A Study of Pair Programming Enjoyment and Attendance using Study Motivation and Strategy Metrics

Onni Aarne*
University of Helsinki
Helsinki, Finland
onni.aarne@iki.fi

Petrus Peltola*
University of Helsinki
Helsinki, Finland
petrus.peltola@helsinki.fi

Juho Leinonen
University of Helsinki
Helsinki, Finland
juho.leinonen@helsinki.fi

Arto Hellas
University of Helsinki
Helsinki, Finland
arto.hellas@cs.helsinki.fi

ABSTRACT

We explore educational pair programming in a university context with high student autonomy and individual responsibility. The data comes from two separate introductory programming courses with optional pair programming assignments. We analyze lab attendance and course outcomes to determine whether students' previous programming experience or gender influence attendance. We further compare these statistics to self-reported data on study motivation, study strategies, and student enjoyment of pair programming. The influence of grading systems on pair programming behavior and course outcomes is also examined.

Our results suggest that gender and previous programming experience correlate with participation in pair programming labs. At the same time, there are no significant differences in self-reported enjoyment of pair programming between any of the groups, and the results from commonly used study motivation and strategy questionnaires provide little insight into students' actual behavior.

ACM Reference Format:
Onni Aarne, Petrus Peltola, Juho Leinonen, and Arto Hellas. 2018. A Study of Pair Programming Enjoyment and Attendance using Study Motivation and Strategy Metrics. In *Proceedings of SIGCSE '18: The 49th ACM Technical Symposium on Computing Science Education (SIGCSE '18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3159450.3159493

1 INTRODUCTION

In the last few decades programming has evolved from a small and clear-cut line of work to a popular hobby and an important skill. With this significant increase in popularity, programming courses are being held at many institutes that do not primarily focus on computing-related subjects. The growth of the IT sector [25] has also resulted in booming enrollments for CS programmes [2].

Programming can be intimidating to newcomers and requires persistent practice to learn. Multiple studies have shown that traditional introductory programming courses suffer from high dropout rates, with generally approximately 30% of the students failing the course [11, 12, 18]. One pedagogical approach for combating the drop-out problem is pair programming [36], where students pair up when working on course assignments and alternating roles between one student writing and the other continuously reviewing the code being produced. It has been shown that the use of pair programming in introductory programming courses can improve course retention rates [22, 37] as well as major retention rates [27]. Pair programming has also been shown to have a positive impact on code quality and student enjoyment [10, 24].

Previous studies have identified multiple factors that can have a significant effect on student enjoyment and the effectiveness of pair programming [7, 13]. These factors include concerns such as the difference between students' prior exposure to programming and the instructor's approach to pair programming. Our research presented in this article complements the previous studies, providing new data on pair programming attendance rates as well as attitudes and whether these are connected to previous programming experience, self-efficacy, gender and study motivation and strategies.

Moreover, our context is quite different from the context in which pair programming studies have traditionally been conducted. Many university classes have mandatory attendance, but at the University of Helsinki where our study was conducted, more emphasis is placed on a student's freedom to study flexibly, with only a few classes having compulsory attendance. This enables many students to take a more independent approach to studying. In fact, many students who never attended pair programming labs did well on the course, despite extra points being offered for participating.

This article is organized as follows. In the next Section, we review pair programming and its use in education. Then, in Section 3, we outline our study methodology, data, and research questions, which are followed by the results and discussion in Sections 4 and 5 respectively. Finally, in Section 6, we present our conclusions and outline possible avenues for future work.

*These authors contributed equally to this work.

SIGCSE '18, February 21–24, 2018, Baltimore , MD, USA

ACM ISBN 978-1-4503-5103-4/18/02…$15.00
https://doi.org/10.1145/3159450.3159493

2 BACKGROUND

2.1 Pair programming

Pair programming is a programming technique where two programmers work together on a single task, typically from a single workstation. It was first broadly adopted in professional use with the rise of Extreme Programming [4], and it has since become a common programming method in the software industry.

When a pair of programmers are working together on a task, one programmer is writing code and the other is continously reviewing the new code as it gets produced. The programmer writing the code usually focuses on what is being written, while the observer can review it in the broader context of the whole program. Generally the programmers switch roles regularly.

Pair programming has been found to have multiple positive effects on the coding process. A study by Cockburn & Williams found that while pair programming increased development time costs by about 15%, it produced all-around better code, which in return reduced the amount of testing and bug fixing required [10]. In addition to raising the quality of the code, it spreads the understanding of what is being produced to at least two persons instead of one, which makes the end product considerably easier to maintain. Programmers also generally seem to prefer pair programming over solitary work [10, 21, 36].

Some factors that can have an unfavorable effect if ignored still exist in the realm of pair programming. Begel & Nagappan found that programmers consider having a complimentary skillset as the most important attribute of a programming partner [5], while the results of Choi et al. suggest that programmers with differing personalities are more productive [9]. Thus, there should be a strong emphasis on forming compatible pairs.

Scheduling is a problem for some [5], since pair programming usually takes place at a set location and time. The problem can be reduced by using online tools to collaborate. Correctly conducted online pair programming can be as effective as collocated pair programming [3, 29].

Another pitfall that was found by Vanhanen & Lassenius was overconfidence in the pair-review process, which led to omitting some of the testing process [32]. In their study, the end products delivered by the pair programing teams actually contained more defects than those that were created by a single person. Solo programmers were also more productive than pairs, but the effect was caused by the pairs having to first get accustomed to the pair programming procedure.

2.2 Pair programming in education

While pair programming made its debut in the software industry, it has also gained popularity in education. Numerous studies have shown its effectiveness as a learning activity [14, 20, 23, 24]. Pair programming has also been used as part of or alongside various pedagogical methods, such as the coding dojo [15], media computation, and peer instruction [27]. One of the largest demonstrated benefits of pair programming is its approachability [33]. It makes programming less intimidating to a newcomer, and by its nature lowers the bar for asking questions. This results in higher retention rates [6].

Pair programming also helps to curtail the so-called code warrior mentality, where a student sees coding as a solo activity and themselves "as a sort of code warrior, fighting with the enemy compiler, forcing it to assent to their glorious code and to produce a program that obeys their every desire" [34]. Such a mentality is harmful, since building the high-quality large-scale programs professional programmers usually work on requires collaboration.

When a student is not just coding by themselves, it creates a positive "pair pressure", which makes pair programming more effective [39]. Williams & Kessler report students feeling more invested in their work as a result of the shared responsibility, thus resulting in fewer drop-outs [38]. Pair programming also forces the students to continuously examine their views on programming and how it is learned, since their pair might have differing views. This expands the learning strategies students have for learning programming, which helps to repel the fixed mindsets some students have [17].

Many studies that have been conducted on pair programming in an educational context have studied the correlation between self-reported preference and course performance, with other factors occasionally explored. The previous studies were predominantly from American universities, which results in a large cultural difference between our study and the ones preceding it. In the American system, students sometimes do not choose their major before the end of the first year, which leads to more participants trying out different courses and later dropping them. In the Finnish educational system, students always apply for a specific programme, and the programming courses are mandatory for computer science students. Additionally, attendance is generally mandatory at American universities, while our system generally only requires presence for exams. Grading schemes do usually incentivize lab attendance, but they vary from course to course and can influence student behavior and outcomes considerably.

Student attitudes towards pair programming and the factors that affect them still remain mostly unexplored, and some of the existing studies have yielded mixed results. When examining the relationship between student attitudes towards pair programming and self-efficacy, Hanks found a positive correlation [13], whereas Thomas et al. found an opposite effect [30]. The results obtained by Thomas et al. may have however suffered from low sample size and convoluted survey design. Hanks on the other hand noted that different instructors also had a major impact on the results. He also surveyed his students on their confidence in their returned assignments, rather than on broader feelings of confidence and programming skill.

Computer science has one of the largest gender gaps of all the sciences, and the proportional number of women in computer science has only been declining since the mid-1980s [28]. Reasons for the apparent disparity and solutions for closing the gap have been studied, and although pair programming improves course performance equally for both genders [22], it has been argued that it benefits women more than men. In their article Werner et al. argue that pair programming specifically addresses factors that limit women's participation in CS [35]. Hanks found that women generally enjoy pair programming more than men, which tangentially supports the argument presented by Werner et al. [13].

3 METHODOLOGY

3.1 Context

The data for this study comes from two introductory Java programming courses organized in the fall of 2015 and the spring of 2016 at the University of Helsinki. The course material was almost identical between the courses, but the population, grading and course structure differed.

In the studied context, the students choose their major during the admission phase. Thus, nearly all CS majors take the introductory programming course in the fall of their freshman year, and therefore make up most of the population of the fall course. In the spring on the other hand, very few CS majors are present. This means that the course demographics differ between the semesters. When only considering students included in this study, the percentages of women were 32.1% and 46.2% in each course respectively. Contrary to our expectations, the two course populations did not differ significantly in terms of previous programming experience according to the Pearson's Chi-squared test ($p < 0.01$).

Both of the courses emphasized learning by doing, which meant that they had more assignments and fewer lectures. Both of the courses had lab sessions four times a week, where students could freely complete the assignments with an instructor ready to help them if needed. The instructors were students who had already passed the course, and usually many instructors were present at any given time. This meant that the influence of a single instructor should not have a large effect on the students. The fall course also had a weekly lecture, while the spring course only had one lecture at the beginning of the course. The teaching methods used are further explained by Kurhila & Vihavainen [19]. Both of the courses had similar exams, but they used different grading. The fall course was only graded as passed or failed. The spring course on the other hand was graded as 5/pass/fail, where 5 signifies the highest grade in the 1–5 grading system used at the university.

3.2 Data and Preprocessing

Data for this study was collected using three methods: (1) a feedback survey that was administered as a part of the course exam, (2) a voluntary Motivated Strategies for Learning Questionnaire (MSLQ) [26], and (3) course grading system. The Motivated Strategies for Learning Questionnaire [26] is used to assess students' use and preference of learning strategies such as peer learning and self-regulation.

The feedback survey included questions on the time invested into the course, amount of exercises completed, etc. as well as questions on the pair programming activity in the course. The questions on the pair programming activity were based on items used by Hanks in [13][1]. The full statements along with response statistics can be found in Table 2. The items were scored using a Likert scale from 1 for completely agree, to 7 for completely disagree. A zero option was available if the student felt they could not provide an answer.

In total, 138 students filled all the surveys and were included in this study. This means that the overall inclusion rate was approximately 50%.

[1]Our survey differed from Hanks' in that the scale was reversed and the zero option was added. In addition, the question 5 was added for the spring course.

Prev. Exp.	Mean	n
None	3.14	78
A Little	1.90	31
By myself	1.55	11
With others on a course	4.20	11
With others on a job	6.00	2
Other	3.80	5

Table 1: Mean number of attended pair programming labs and sample sizes of students of different levels of previous experience.

3.3 Research Questions

Our research questions for this study are as follows.

- RQ1: How actively do students attend pair programming labs?
- RQ2: What are student attitudes towards pair programming like?
- RQ3: Do pair programming attendance rates or attitudes vary by gender or previous programming experience?
- RQ4: How do students' motivation and learning strategies affect their preference towards pair programming?
- RQ5: Do students' feelings of self-efficacy affect their attitudes towards pair programming?

4 RESULTS

Unless otherwise noted, these results were obtained by treating the two courses as a single population. Results were considered statistically significant at $p < 0.0025$ due to a Bonferroni correction for multiple comparisons. Seven students were missing data on pair programming lab attendance, so they were not included in tests where that information was relevant. Additionally, when studying the effect of gender, five students were excluded as they did not disclose their gender.

4.1 Pair programming attendance

Attendance in the pair programming labs was recorded by the instructors to whom the pair programming assignments were returned. When analyzing the attendance data, we found that 74.5% of students participated in at least one pair programming lab of the six held throughout each course. This percentage was 86.8% for women and 67.3% for men.

The overall mean number of pair programming labs attended was 3.02 out of six. Women attended 3.58 labs on average whereas men only attended 2.69. In addition to being more likely to attend at least once, women were more likely to attend every session. A Kruskal-Wallis rank sum test indicated that the differences in pair programming activity between male and female students were statistically significant.

As shown in Table 1, students with no previous programming experience were among the most active in attending pair programming labs in addition to being the most numerous. The least active were the students who reported having some programming experience by themselves. The most active pair programmers were the

Item	Statement	Mean	Median	Mode
Q1	I like pair programming.	3.28	3	2
Q2	I would like to pair program again on another course or in my job.	3.66	4	3
Q3	I learned more on this course because I pair programmed.	3.78	4	3
Q4	I had more fun on this course because I pair programmed.	4.06	4	4
Q5	I would prefer to pair program with someone I know.	2.81	2	1

Table 2: Mean, median and mode values from the Likert items in the range from 1 to 7. Lower numbers signify greater agreement. Any zero responses were excluded.

students who reported having previous experience programming on a course or job. However, some of these results are undermined by small samples for certain categories, especially "on a job", as can be seen from Table 1.

No statistically significant differences in exam scores between the students with different levels of previous programming experience was identified. Similarly, pair programming activity was not significantly correlated with exam scores on either course according to Pearson's product-moment correlation tests.

A Pearson's Chi-squared test showed that there were statistically significant differences in the distribution of previous programming experience between male and female students with male students being more likely to have at least some previous exposure to programming.

When considering attendance in pair programming sessions, female students significantly out-attended their male counterparts on average (3.51 vs 2.42 for no previous programming experience and 3.29 vs 1.47 for some programming experience). This indicates that the differences in lab attendance between genders can not be explained by differences in previous experience.

4.2 Attitudes towards pair programming

The survey questions and their respective mean, median and mode values can be seen in Table 2. A detailed explanation of how these results were acquired can be found in Section 3.2. For the Spearman correlation tests, zero-level responses were removed from the data, and the signs of the responses were flipped in the Likert data in order to make more positive responses have higher ranks, thus making the sign of the obtained ρ what one would expect it to be.

No statistically significant differences between men and women were found in responses to Q1. It is worth noting, however, that the response counts differed significantly between genders. For this specific Likert item, the female students included in this study had a response rate of 88.2%, with the same statistic being only 69.2% for men. These numbers closely mirror the aforementioned differences in rates of students participating in at least one pair programming lab. Interestingly, more students had opinions about pair programming than ever attended the pair programming labs.

Associations between self-reported attitudes and performance statistics were also examined. A Spearman's rank correlation test indicated that response levels to the question on liking pair programming exhibited a statistically significant positive correlation with lab attendance with $\rho = 0.322$. The same test showed no statistically significant correlation between the Likert responses and exam scores.

Responses to Q5 were highly polarized, though most students did agree with the statement.

Our and Hanks' survey results by gender are summarized in Table 3. Hanks' results indicated clearly favorable attitudes toward pair programming, whereas our students gave mixed responses. None of our mean values were even one point off the neutral value of four. It should be noted that Hanks only surveyed volunteers, whereas our survey was shown to all students who came to the final exam, and they were incentivized to fill it by offering a single point toward their course score.

4.3 Student study motivation and strategies

We tested a handful of MSLQ motivation and learning strategies scales for associations with pair programming lab attendance and self-reported attitudes. These scales were as follows: peer learning, metacognitive self-regulation, task value and self-efficacy. Correlations between these values and lab attendance were checked with Pearson's product-moment correlation tests, whereas Spearman's rank correlation tests were used to check correlations with responses to Q1.

No statistically significant correlation was found between any of the factors examined and the questionnaire items in Table 2, with the exception of the MSLQ Peer Learning scale and lab attendance, which had a moderate correlation. That is, the students with a higher preference to learning with peers were also more likely to attend the pair programming labs. However, at the same time, even though peer learning correlated with pair programming attendance, no statistically significant correlation between the Likert Responses of the questions in Table 2 was identified.

5 DISCUSSION

Our results complement the previous mixed results between student confidence metrics and their pair programming preferences. Thomas et al. [30] and Hanks [13] both identified statistically significant correlations between different student confidence and pair programming preference metrics, though in opposing directions. Our study provides a third viewpoint that suggests that there is no statistically significant "global truth", and that the results are at least partially influenced by the context.

It is interesting to note that a positive correlation between exam scores and pair programming attendance in the fall course would have been considered statistically significant at $p = 0.0064$ if not for our Bonferroni correction. This could be due to the fact that the fall course was graded as pass/fail, whereas the spring course was graded as 5/pass/fail, in a system where 5 is the highest achievable grade.

		Q1		Q2		Q3		Q4		Q5	
Survey	Gender	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Hanks	Male	2.84	2	2.75	2	3	3	2.51	2	N/A	N/A
	Female	2.43	1	2.49	1	2.71	2	2.29	1	N/A	N/A
This	Male	3.27	2	3.69	4	3.91	4	4.01	4	2.67	2
	Female	3.28	2	3.59	3	3.65	3	4.16	4	3	2

Table 3: A comparison between our and Hanks' results from the same survey. Hanks' results have been converted to our reversed scale.

In the spring course that had the possibility for the highest grade for students who amassed over 90% of the course points, no correlation whatsoever was observed between lab attendance and exam scores. This indicates that, in the spring course, the extra points offered for participating in the labs were significantly more valuable to the students than for the students in the fall course. This means that it is possible that some students attended purely to target the largest possible amount of points.

This suggests that the (non-significant) correlation observed in the fall course can be attributed to more committed students both attending more labs and performing better in the exam. Therefore, one may question whether pair programming labs are the actual factor that explains better course performance, or if there are some underlying variables that we do not know of. In other words, in so far as the exams are considered to be a valid test of skill acquisition, pair programming does not appear to increase skill acquisition over other methods of study.

The difference between our and Hanks' results might be explained by reduced sampling biases, since we offered a point for filling the survey whereas Hanks' survey was only filled by volunteering students. This could result in our sample being more representative of the course population, since Hanks' volunteers might have very well been more active or opinionated students. Hanks showed that instructors can have a significant impact on student attitudes towards pair programming, which could explain why our students exhibited more ambivalent attitudes.

Cultural differences in Likert response styles could also influence some of our results, as for example, Chen et al.[8] showed that East Asian participants were less likely to use the extremes of a Likert scale when compared to American participants. They proposed that differences on the individualism/collectivism spectrum might explain some of these biases. Finnish culture could be considered to be less individualistic compared to American culture, which might be a contributing factor in the differences between our results and others. It is also worth noting that because the course was conducted in Finnish, the Likert items had been translated. This could have led to subtle differences in how the statements were interpreted. Most notably, the specific translation we used for "like" (pitää) in Q1 could be perceived as a slightly more serious expression than the original. This could also have had a mild neutralizing effect on the response levels.

When considering possible explanations for the low attendance of those with a little experience, we hypothesize that at least some had completed some basic programming tutorials online, which may have led them to overestimate their own abilities and underestimate their need for practice and instruction. Alternatively, students who had programmed before on their own could have simply been more independent learners and therefore perhaps did not prioritize the lab sessions.

An environmental factor that could have influenced our sample is that although the course the data comes from is mandatory for all CS majors, our university provides experienced programmers the option to skip the studied course if they are able to pass an exam equivalent to the final exam of the course studied here. This could have made our sample less representative of CS majors in general, but perhaps better representative of those just starting to learn programming.

6 CONCLUSIONS

In this work, we studied attitudes towards pair programming and whether those attitudes are related to pair programming activity, gender, and various study strategy and motivation metrics from the MSLQ Questionnaire. Our work differs from the trend of pair programming studies through the context in which it was conducted; in our context, students have high autonomy, lectures (or attendance) is not mandatory, and students choose their major already during enrollment to the studies at the university.

Our results indicate that students with little programming experience by themselves are less likely to attend lab sessions compared to their peers who either have no previous programming experience or who have previous experience of programming with other people. Furthermore, we found that women attended more sessions on average than men, and that this effect was not explainable by differences in previous programming experience. In addition, we compared students' attitudes towards pair programming in our context with the results from previous research by Hanks [13] and Thomas et al. [31] using the same questionnaire as Hanks.

Our results add to Hanks' and Thomas et al.'s results as – somewhat surprisingly – MSLQ answers did not have statistically significant correlations with pair programming attitudes in our context. The conflicting results with B. Hanks [13] and Thomas et al. [30] show that further research into student attitudes towards pair programming, self-efficacy and the factors affecting them needs to be done, since no consensus has been reached.

Moreover, when studying pair programming attendance from the perspective of the grading of the course, we found that pair programming is more likely to explain students' exam performance if the grading is simply pass and fail. That is, the external motivations

provided by mechanisms such as the pass / fail / best grade system can increase lab attendance, but not necessarily in a productive way.

Finally, the results presented in this work provide additional evidence on the recent discussion in computing education research that has highlighted the importance of replication studies [1, 16] for examining whether results from one context generalize to others as well as for building stronger theoretical foundations on which to base future research as well as teaching.

REFERENCES

[1] Alireza Ahadi, Arto Hellas, Petri Ihantola, Ari Korhonen, and Andrew Petersen. 2016. Replication in computing education research: researcher attitudes and experiences. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM, 2–11.

[2] Computing Research Association et al. 2017. Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006. (2017).

[3] Prashant Baheti, Edward Gehringer, and David Stotts. 2002. Exploring the efficacy of distributed pair programming. *Extreme Programming and Agile Methods—XP/Agile Universe 2002* (2002), 387–410.

[4] Kent Beck. 2000. *Extreme programming explained: embrace change*. addison-wesley professional.

[5] Andrew Begel and Nachiappan Nagappan. 2008. Pair programming: what's in it for me?. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 120–128.

[6] Jeffrey C Carver, Lisa Henderson, Lulu He, Julia Hodges, and Donna Reese. 2007. Increased retention of early computer science and software engineering students using pair programming. In *Software Engineering Education & Training, 2007. CSEET'07. 20th Conference on*. IEEE, 115–122.

[7] Edgar Acosta Chaparro, Aybala Yuksel, Pablo Romero, and Sallyann Bryant. 2005. Factors affecting the perceived effectiveness of pair programming in higher education. In *Proc. PPIG*. 5–18.

[8] Chuansheng Chen, Shin-Ying Lee, and Harold W. Stevenson. 1995. Response Style and Cross-Cultural Comparisons of Rating Scales among East Asian and North American Students. *Psychological Science* 6, 3 (1995), 170–175. http://www.jstor.org/stable/40063010

[9] Kyungsub S Choi, Fadi P Deek, and Il Im. 2008. Exploring the underlying aspects of pair programming: The impact of personality. *Information and Software Technology* 50, 11 (2008), 1114–1126.

[10] Alistair Cockburn and Laurie Williams. 2000. The costs and benefits of pair programming. *Extreme programming examined* (2000), 223–247.

[11] Mark Guzdial and Elliot Soloway. 2002. Teaching the Nintendo generation to program. *Commun. ACM* 45, 4 (2002), 17–21.

[12] Hossein Hakimzadeh and James Wolfer. 2009. Introducing a CS0 class to ease transition to CS1: Another attempt at retention. In *INTERTECH 2009–International Conference on Engineering an Technology Education, Buenos Aires, Argentina*.

[13] Brian Hanks. 2006. Student attitudes toward pair programming. In *ACM SIGCSE Bulletin*, Vol. 38. ACM, 113–117.

[14] Brian Hanks, Charlie McDowell, David Draper, and Milovan Krnjajic. 2004. Program quality with pair programming in CS1. In *ACM SIGCSE Bulletin*, Vol. 36. ACM, 176–180.

[15] Kenny Heinonen, Kasper Hirvikoski, Matti Luukkainen, and Arto Vihavainen. 2013. Learning Agile Software Engineering Practices Using Coding Dojo. In *Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education (SIGITE '13)*. ACM, New York, NY, USA, 97–102. https://doi.org/10.1145/2512276.2512306

[16] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, et al. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM, 41–63.

[17] Shamim Khan, Lydia Ray, Aurelia Smith, and Angkul Kongmunvattana. 2010. A pair programming trial in the cs1 lab. In *Proc. Annual International Conference on Computer Science Education: Innovation and Technology (CSEIT)*. 6–7.

[18] Päivi Kinnunen and Lauri Malmi. 2006. Why students drop out CS1 course?. In *Proceedings of the second international workshop on Computing education research*. ACM, 97–108.

[19] Jaakko Kurhila and Arto Vihavainen. 2011. Management, Structures and Tools to Scale Up Personal Advising in Large Programming Courses. In *Proceedings of the 2011 Conference on Information Technology Education (SIGITE '11)*. ACM, New York, NY, USA, 3–8. https://doi.org/10.1145/2047594.2047596

[20] Charlie McDowell, Brian Hanks, and Linda Werner. 2003. Experimenting with pair programming in the classroom. In *ACM SIGCSE Bulletin*, Vol. 35. ACM, 60–64.

[21] Charlie McDowell, Linda Werner, Heather E Bullock, and Julian Fernald. 2003. The impact of pair programming on student performance, perception and persistence. In *Proceedings of the 25th international conference on Software engineering*. IEEE Computer Society, 602–607.

[22] Charlie McDowell, Linda Werner, Heather E. Bullock, and Julian Fernald. 2006. Pair Programming Improves Student Retention, Confidence, and Program Quality. *Commun. ACM* 49, 8 (Aug. 2006), 90–95. https://doi.org/10.1145/1145287.1145293

[23] Emilia Mendes, Lubna Basil Al-Fakhri, and Andrew Luxton-Reilly. 2005. Investigating pair-programming in a 2 nd-year software development and design computer science course. In *ACM SIGCSE Bulletin*, Vol. 37. ACM, 296–300.

[24] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. 2003. Improving the CS1 experience with pair programming. *ACM SIGCSE Bulletin* 35, 1 (2003), 359–362.

[25] Bureau of Labor Statistics. 2017. Careers in the growing field of information technology services. https://www.bls.gov/opub/btn/volume-2/careers-in-growing-field-of-information-technology-services.htm. (2017). Accessed: 2017-08-28.

[26] Paul R Pintrich et al. 1991. A manual for the use of the Motivated Strategies for Learning Questionnaire (MSLQ). (1991).

[27] Leo Porter and Beth Simon. 2013. Retaining Nearly One-third More Majors with a Trio of Instructional Best Practices in CS1. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 165–170. https://doi.org/10.1145/2445196.2445248

[28] Linda J Sax, Kathleen J Lehman, Jerry A Jacobs, M Allison Kanny, Gloria Lim, Laura Monje-Paulson, and Hilary B Zimmerman. 2017. Anatomy of an enduring gender gap: The evolution of women's participation in computer science. *The Journal of Higher Education* 88, 2 (2017), 258–293.

[29] David Stotts, Laurie Williams, Nachiappan Nagappan, Prashant Baheti, Dennis Jen, and Anne Jackson. 2003. Virtual teaming: Experiments and experiences with distributed pair programming. In *XP/Agile Universe*. Springer, 129–141.

[30] Lynda Thomas, Mark Ratcliffe, and Ann Robertson. 2003. Code warriors and code-a-phobes: a study in attitude and pair programming. In *ACM SIGCSE Bulletin*, Vol. 35. ACM, 363–367.

[31] Lynda Thomas, Mark Ratcliffe, John Woodbury, and Emma Jarman. 2002. Learning styles and performance in the introductory programming sequence. In *ACM SIGCSE Bulletin*, Vol. 34. ACM, 33–37.

[32] Jari Vanhanen and Casper Lassenius. 2005. Effects of pair programming at the development team level: an experiment. In *Empirical Software Engineering, 2005. 2005 International Symposium on*. IEEE, 10–pp.

[33] V Venkatesan and A Sankar. 2010. Adoption of pair programming in the academic environment with different degree of complexity in students perspective–an empirical study. *International Journal of Engineering Science and Technology* 2, 9 (2010), 4791–4800.

[34] Debora Weber-Wulff. 2000. Combating the code warrior: A different sort of programming instruction. *ACM SIGCSE Bulletin* 32, 3 (2000), 85–88.

[35] Linda L Werner, Brian Hanks, and Charlie McDowell. 2004. Pair-programming helps female computer science students. *Journal on Educational Resources in Computing (JERIC)* 4, 1 (2004), 4.

[36] Laurie Williams, Robert R Kessler, Ward Cunningham, and Ron Jeffries. 2000. Strengthening the case for pair programming. *IEEE software* 17, 4 (2000), 19–25.

[37] Laurie Williams, Charlie McDowell, Nachiappan Nagappan, Julian Fernald, and Linda Werner. 2003. Building pair programming knowledge through a family of experiments. In *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*. IEEE, 143–152.

[38] Laurie A Williams and Robert R Kessler. 2000. The effects of" pair-pressure" and" pair-learning" on software engineering education. In *Software Engineering Education & Training, 2000. Proceedings. 13th Conference on*. IEEE, 59–65.

[39] Laurie A Williams and Robert R Kessler. 2001. Experiments with industry's "pair-programming" model in the computer science classroom. *Computer Science Education* 11, 1 (2001), 7–20.