Investigating Remote Pair Programming In Part-Time Distance Education

Adeola Adeliyi
School of Computing and Communications
The Open University
Adeola.Adeliyi@open.ac.uk

Karen Kear
School of Computing and Communications
The Open University
Karen.Kear@open.ac.uk

Michel Wermelinger
School of Computing and Communications
The Open University
Michel.Wermelinger@open.ac.uk

Jon Rosewell
School of Computing and Communications
The Open University
Jon.Rosewell@open.ac.uk

ABSTRACT

Pair programming promotes immediate, informal collaboration over coding activities. The driving developer writes the code and controls the keyboard and mouse; the navigating developer checks the code as it is written by the driver; the developers swap their roles frequently. In agile development, programmers often code in pairs, in order to detect errors faster, produce higher code quality and discover better solutions.

There is substantial research providing evidence of enhanced self-confidence and programming and communication skills if pair programming is used in teaching. However, the use of pair programming in higher education is mostly in co-located settings at campus-based universities. Our overall objective is to investigate how the benefits of pair programming can be brought to students learning to program online at a distance.

This paper presents two initial studies looking at remote pair programming (RPP) also called distributed pair programming, in a part-time distance education setting, where students typically follow an unscheduled self-study style, have little interaction with each other, and have little time for extra activities. We investigated: whether readily available generic communication tools, instead of purpose-built academic prototypes, were sufficient for RPP; whether student pairs 'jelled' (learned to function well together) quickly; whether the ways in which the partners interact, or existing programming experience, affected jelling; and whether students felt positive about, and saw benefits in RPP, despite the overhead on their limited study time.

In the paper, after describing particular challenges encountered, we present and discuss our findings and make recommendations for future implementation. The findings support the use of remote pair programming in teaching, with the majority of students considering it to be beneficial.

CCS CONCEPTS

•Applied computing ~ Education ~ Collaborative learning •Applied computing ~ Education ~ Distance learning

KEYWORDS

Remote Pair Programming, Distance Learning, Teaching Programming

1. Introduction

Pair Programming (PP) is widely used in agile software development. A developer in the driver role writes the code, and controls the keyboard and mouse; the other developer, the navigator, watches for potential defects and assists with design decisions as the code is being written. These roles are switched often as the development progresses. Advocates argue that PP leads to better designed solutions with fewer defects, and reduces silos of knowledge about the codebase [38].

Using PP in education has been advocated for many years. Some of the learning advantages seen in the classroom are:

- Improved academic results for students [25, 29-31]
- Improved efficiency in programming in terms of coding time and quality [11, 12, 27, 32, 35, 42]
- Improved student satisfaction [25, 26, 29, 30, 43]
- Reduced workload for the teachers [30, 33]
- Improved coding productivity for female students [2, 40]

For collaborative coding to be considered PP, both programmers must work at the same computer and continually talk to each other and carry out other physical interactions such as pointing to the screen [38, 41]. These collocation and communication requirements are straightforward to implement in a programming lab but become more challenging in distance learning contexts.

The Open University (OU) provides distance programmes to mainly part-time mature students, who have professional and family commitments. Our Computing students learn to program mostly in their own time, without direct supervision. They expect and prefer unscheduled self-study. In this context, it is not easy to introduce remote pair programming (RPP), also called distributed pair programming, in which partners aren't collocated. We conducted two studies to explore the potential benefits of RPP for our students and to identify whether barriers could be overcome.

For most educational research on RPP, the educators/researchers developed bespoke tools, e.g. IDE plugins. These are often prototypes, difficult to obtain and install, and restricted to particular platforms, IDEs, or programming languages. However, time-poor students need readily available tools that are easy to install and use. Our first research question is:

1. Can some generic collaborative tools be adopted for RPP?

'Pair jelling' happens as both programmers adjust from solitary to collaborative programming. Part-time students in distance education often don't know and don't interact with each other. That could lead to longer jelling times, and in turn less engagement with RPP. We thus want to know how long jelling takes for such students. Individual students interact with their partners in characteristic ways that Kaur Kuttal, *et al.* [20] ascribed to 'leadership style' and this affects their collaboration, e.g. in how they swap roles. To our knowledge, this paper is the first to look at jelling and leadership styles.

2. How long does it take for a pair to 'jel' and is it related to leadership style?

RPP entails some overhead (learning a tool, scheduling sessions, etc.) and imposes collaborative work, which many distance learners dislike [21]. We want to know if students in a self-study context nevertheless feel positive about RPP.

3. Do students perceive worthwhile benefits of RPP?

For space reasons, the next section presents only some of the related work on RPP. We then describe the design and results of our two exploratory studies in Sections 3 and 4 and conclude in Section 5.

2. Related Work

There are several studies of RPP in education [7] but most look at outcomes such as productivity (lines of code per hour) and grade improvement: while important, they are not related to our work.

The study by Hughes, *et al.* [17] is also in a part-time distance education setting but their main aim was to see the effect of passive, indirect and direct PP participation on confidence and self-esteem. They asked students to: watch a video of collocated tutors pair programming; attend a live demonstration of RPP; do one RPP session with a student partner. Only 4 students (2 pairs) did the latter, whereas our studies involve more students, pairing over multiple sessions.

Some studies considered partner compatibility. Kaur Kuttal, *et al.* [20] looked at gender and RPP. They found that same-gender pairs were democratic but mixed-gender pairs more often had one authoritative partner and that women did not benefit from the mixed-gender pairing. Hanks [16] asked volunteering students about their schedule and skill level to allocate compatible pairs. However, that study did not investigate whether there was a link between the student's results and the approach to pairing students. Lui [24] and Wray, *et al.* [44] suggested that to achieve the full educational advantage of RPP, skilled students should be matched with novices, but Shaw [34] believes this creates an imbalance between partners where the skilled student ends up doing most of the work. Wanfeng, *et al.* [39] considered students' personality type, skill, gender and time management in analysing requirements for compatibility in RPP. Students showed a preference for another partner with similar or higher technical skills. Students with low technical skills expressed frustration as their lack of programming skills limited the progress of their collaboration. Xinogalos, *et al.* [45] observed that pairs with collective better performance in a prior course maintained strong academic results in a following course supported by RPP, while pairs with poor prior performance performed far worse.

The success of implementing RPP depends on the tools that support the workflow [10]. The communication tool failure faced by students in formal experiments by Canfora, *et al.* [4], [5] led to the dismissal phenomenon (where students stopped collaborating), poor productivity and substandard code quality.

Based on prior research, it is important for RPP tools to support:

1. Synchronous Collaboration – both students should see the driver's screen in real-time, following the WYSIWIS (What You See Is What I See) metaphor [36].
2. Shared and equal access – both students should have shared and equal access to the same code repository [13].
3. Integrated communications – both students can engage in discussion without losing focus on the code. Features that allow the student to make subtle gestures, such as head-shake or finger-pointing, enhance communication and improve the pairing experience [10, 15].

Some tools, like DistEdit [22], allow both students to type and execute code simultaneously, which risks introducing errors if a strict driver-navigator protocol isn't followed. A modified version of the VNC system by Hanks [16] introduced a way for the navigator to request the cursor. This approach makes students aware of each other's roles and activities. FASTDash [3] and Palantir [1] also provide awareness, using a spatial representation of the shared codebase that highlights the team members' current activities. Collece and Collabode are web-based IDEs for Java that share changes instantly, without the need for version control [9, 14]. These tools need an additional audio-visual communication tool, whereas Jimbo [13] and CodeBuddy [23] are IDEs with integrated video and audio calls and text messaging. CodeBuddy also includes automatic and manual role switching, code quality analysis, engagement analysis and code commenting.

Tsompanoudi, *et al.* [37] implemented a scripting mechanism by an Eclipse IDE plugin that automates role switches between paired students within set time limits. Students reported the overall experience as good but they did not like the forced role rotation as it becomes disruptive to the thinking process of the driving student. D'Angelo, *et al.* [6] developed an eye-tracking system to prompt the driver to a location where the navigator is looking at the screen.

3. Study Design

We started with a search for readily available generic collaborative tools that could support RPP, even if they don't provide the advanced features of bespoke tools (Section 2). Using the chosen tool(s), students conducted RPP sessions and were afterwards asked about their experience, including the effectiveness of the tool and the perceived improvement of programming, communication and other skills.

As noted above, collaborative RPP tools must support synchronous collaboration, equal access, and integrated communications. Doing a web search on "collaborative tools", "screen sharing applications" and "code editors for pair programming", the results were filtered down to 8 tools that meet the three requirements: AdobeConnect, Skype, Stride, GoToAssist, TeamViewer, USETogether (since renamed to Drovio), Microsoft Teams and Zoom. USETogether is specifically aimed at RPP; the others are generic communication tools that can be adopted for RPP.

All these tools allow voice and video calls and sharing desktop screens. Zoom and TeamViewer are cloud-based applications with extra features such as whiteboard and annotation tools. This enables both programmers to make notes, create drawings, and add arrows on the shared screen. USETogether allows both programmers to see each other's cursor moving around on the screen in real-time.

One of the key principles of PP is to swap the driver and navigator roles. AdobeConnect, TeamViewer, GoToAssist, Zoom, and Stride require a restart of a session, with the new driver initiating a session on their computer. Skype, Microsoft Teams and USETogether allow both partners to use the initial driver's computer, as in traditional PP where both partners sit at the same computer. The current driver can pass control of the computer to the navigator without having to restart the session.

Table 1 summarizes the tool evaluation. USETogether and Microsoft Teams were chosen for the pilot study because they provide all features. This phase of the research was carried out in June 2019 and thus doesn't include any improvements made to the tools since then.

Tool	Audio-Visual Communication	Whiteboard & Annotations	Mouse Cursor Control	Ease of swapping Roles
AdobeConnect	x	x		
GoToAssist	x	x	x	
Stride	x			
Skype	x			x
TeamViewer	x	x	x	
USETogether	x	x	x	x
Zoom	x	x	x	
Microsoft Teams	x	x	x	x

Table 1: ***Summary of collaborative tools***

Following the necessary institutional ethical approvals, students on several modules that include programming were invited, by email, to participate. Interested students were asked to complete a consent form, and an initial questionnaire about their programming experience. Pairs were formed from students on the same module. They were asked to collaborate on non-assessed programming activities which they would otherwise have carried out individually as part of their studies.

An internal project website was developed to inform students about PP and RPP and this research, to collect consent forms and to deliver the research instruments. The website also provides resources on communication software installation and usage, as well as a forum for students to ask any questions they had about the research study.

The first session for each pair was an ice-breaking meeting facilitated by the researcher, which included describing the principles of pair programming. Students were encouraged to ask questions about the research and get to know the assigned partner. From the second session, the pairs self-directed the sessions, i.e. they decided when to meet, which activity was to be done and who would drive first. The sessions were recorded for later observation.

The students were asked to complete a reflective journal after each session, to get an insight into their experience as the study progressed. We provided the students with prompts to write about: their feelings before, during and after the sessions, evaluation of the tools and working with their partner, and an overall reflection. For example, we asked "What was good/bad?", "Did you work well with your partner?", "Were there any technical issues?". At the end of the study, participants were invited to complete another questionnaire. A voucher was given to those who did, as a token of appreciation for their time.

These studies did not assess students' code quality or academic performance but rather how the students interacted with each other using the provided collaboration tool, and their overall experience. We conducted two studies, exploring different aspects, to get an idea of the variables that may affect RPP. The studies covered collaborative tools, pairing method, pair jelling, potential benefits (time management, study habits, programming skills, confidence) and overall satisfaction.

In study 1, about 300 students were invited from one 1st year and one 2nd year module, and 16 signed up (low participation rates in research studies are typical in part-time distance learning contexts, partly because students have very little free time available). Of those, 4 dropped out before the study began and 2 could not be paired as they were studying different modules. The remaining 10 students were paired based on their time availability indicated in the pre-study questionnaire. This study examined two collaborative tools (USETogether and Microsoft Teams), students' experience and the pair jelling effect.

In study 2, 1769 students were invited, of whom 122 signed up and 24 completed the study, i.e. filled in the end survey. Most students dropped out due to the impact of COVID 19 and the difficulty in scheduling a convenient time for the sessions. The pairing was carried out without regard to gender, skills or availability. This study narrowed its examination to one tool (Microsoft Teams) and to investigating students' reported experience.

4. Analysis And Results

We examined qualitative data gathered from the students' reflective journals, as well as quantitative and qualitative data from the end survey questions, which were based on a Likert scale varying from 1 (strong disagreement with a given statement) to 5 (strong agreement). Text boxes allowed students to elaborate their responses. We used the median to measure the central tendency of the survey responses and interquartile range (IQR) to assess their degree of dispersion. In the tables that follow we have highlighted results that are either negative (median ≤ 2) or positive (median ≥ 4) and also suggest consensus (IQR < 2) rather than polarised views (IQR ≥ 2) about the statement.

4.1 Using Collaboration Tools

To address research question 1, we asked students to rate their experiences of using the tool. Students in study 1 started using USETogether and switched to Microsoft Teams towards the end due to issues with the former. Study 2 participants used Teams only.

Student Experience Statement	Study 1 (n = 10)				Study 2 (n = 24)	
	USETogether		Microsoft Teams		Microsoft Teams	
	MED	IQR	MED	IQR	MED	IQR
It was easy to install	4.5	1	5	0.5	4	1
It was easy to use with my programming partner	4	1	4.5	1	4	1
It has a high impact on my computer systems' resources	2	2	2	1.5	2.5	1
It froze or crashed sometimes	2	2	3	2	2	2
The audio-visual quality is adequate	4	0	4	0.5	4	0.5
We were able to record our sessions	3	0.5	4	1.5	4	1.5

Table 2: Students' experience of using collaborative tools.

As Table 2 shows, most students agreed that USETogether and Microsoft Teams were easy to install, easy to use with a partner and have adequate audio-visual quality. Microsoft Teams supports the recording functionality well (i.e. students can create an audio-visual recording of their RPP session) while USETogether performed less well in this regard. There was a mix of experiences (IQR = 2) with regard to the computer becoming unresponsive or crashing.

USETogether is specifically developed for RPP, whereas Teams isn't, and yet responses are similar. While the number of responses is too small to draw general conclusions, they seem to indicate that generic collaboration tools can be adopted for RPP.

4.2 Pair jelling period and leadership style

The first author observed the sessions, noted the interactions and categorised each partner according to the leadership styles used in the study by Kaur-Kuttall *et al.* [20]:

i. Democratic style (shares the decisions with the other partner)
ii. Authoritative style (dominates the interactions)
iii. Laissez-faire style (all decision-making is delegated to the other partner)
iv. Paternalistic style (instructs the other partner)

The observed style and the self-reported skills level in the pre-study questionnaire were taken into account to investigate their effects on pair jelling. The end survey asked the students how long it took to jel. The last column of Table 3 is the number of sessions they reported until perceived jelling, which agrees with observations of the online sessions.

The table suggests that partners with the same style may jel the fastest. In pair Gamma, the novice student P2 had gone further in the module activities and thus tended to teach the more advanced student P1, thus appearing paternalistic.

Beta and Epsilon are the only pairs with equal skill levels. Based on Melnik, *et al.* [28] reporting that students prefer to collaborate with a partner of equal skill and experience, one would have expected these pairs to jel more quickly. This preference seems to have been overridden by the pairs' styles: it took three sessions for Beta (democratic/authoritative) to jel while Epsilon (both democratic) jelled in one. Remote teams can feel a "swift" level of trust but this trust seems very frail and it is critical to collaboration success or failure [18].

The study seems to indicate that leadership style affects how paired programmers jel. All partners' styles remained unchanged until jelling. Once jelling had taken place, the student pairs were able to accommodate each other's style in a more fluid interaction.

Pair	Gender		Leadership Style		Skills Level		Sessions before jelling
	P1	P2	P1	P2	P1	P2	
Alpha	Male	Male	Democratic	Authoritative	Novice	Intermediate	2
Beta	Male	Male	Democratic	Authoritative	Novice	Novice	3
Delta	Male	Male	Democratic	Democratic	Intermediate	Novice	1
Epsilon	Female	Female	Democratic	Democratic	Novice	Novice	1
Gamma	Male	Female	Democratic	Paternalistic	Advanced	Novice	2

Table 3: Students' jelling factors and period for study 1

4.3 Benefits

With a significant amount of research reporting the benefits of PP on co-located students, we included 4 statements in the end survey for students to appraise the benefits for their learning (see Table 4).

Student impact	Study 1 (n= 10)		Study 2 (n = 24)	
	MED	IQR	MED	IQR
My coding and debugging skills are better than before I used pair programming	4.5	1	4	1
Participating in RPP has improved my confidence level when communicating my thought process	5	1	4	1.5
My RPP sessions helped improve my time management skills	4	1	3	2
My study habits were positively affected because of the RPP sessions	4.5	3	3.5	1.5

Table 4: Impact of RPP on students

All students who had completed more than 4 sessions (the minimum required) were invited to complete the survey, even if they subsequently dropped out from the study. As Table 4 shows, most students agreed or strongly agreed that RPP improved their learning experience. The exception is in Study 2 groups, where the median opinion on the impact on time management skills was neutral. While groups in Study 1 were paired based on their indicated time availability, the groups in Study 2 were paired randomly, irrespective of time availability. This is reflected in how students manage their schedules as they struggled to find a suitable time to accommodate each other. Two study 2 students commented:

"I found it difficult to arrange a mutually convenient time to take part in sessions with my partner."

"*the peer cancelled session 3 with a five minutes notice and I never heard from her again. No reason was given and there was no attempt to reschedule."*

As mentioned in Section 1, we probed student satisfaction in the face of any issues. In the end survey we asked "What is your overall feeling about your experience of using RPP in your module?" using a Likert scale from 1 (very dissatisfied) to 5 (very satisfied). Study 1 responses had median 4 and IQR 2 (n = 10); study 2 responses had median 4 and IQR 1 (n = 24). We also asked "Would you recommend RPP to other students and its wider use within computing modules?" and most replied yes (93% in study 1 and 83% in study 2). Overall, there is a positive attitude towards remote pair programming, confirming previous research on pair programming. Looking at the survey's comments, students who were satisfied or very satisfied mentioned the following:

- Improved feeling of inclusion and social interaction
- Improved team working experience, e.g. confidence in communication, motivation, skill for employment
- A positive impact from peer pressure, e.g. keep up with studies before sessions, time management and discipline.
- Improved coding skills from learning from pair, i.e. knowledge transfer.

Some comments from the students are:

"During the RPP sessions I have noticed I was more focused and engaged with the task than I normally would by myself".

"My coding skills have definitely improved, but also my understanding of how to code and what you can and shouldn't do. It was also really nice to have someone else to talk to who is studying the same module as me. Distant learning can get very lonely and this has massively helped alleviate this feeling"

"I am very satisfied with the experience because I feel it brought to light the best part of me in a programming context. I always felt problem-solving includes creativity and ideas come much easier when I can bounce them with a partner, our minds are more agile and personally, I felt empowered to know I am not alone in solving it."

Students who were dissatisfied or neutral mentioned the following factors: the partner didn't show up for sessions; they felt their module's exercises were not suitable for pair programming. It is worthwhile to note that in the overall assessment of the benefits of RPP, some students who reported technical challenges in their journal (inability to record sessions, disconnection from the internet, computer crashes, etc.) nevertheless have positive feelings towards RPP for online learning.

In study 1, where students were paired by availability, there were fewer incidents of a partner not showing up in comparison with study 2 where the pairing was randomised. Katira, *et al.* [19] found that collaboration is successful even when students are paired randomly, but our studies suggest this is not true for part-time students. A student noted in their journal that *"the whole point of [the] OU is being able to fit around individuals, so having a pair [imposed] can make it difficult to schedule time*". Co-located full-time students have a structured learning process, e.g. with timetabled laboratory sessions. This contrasts with the flexibility on which most part-time remote teaching and learning is based [8].

4.4 Limitations And Future Research

The studies reported in this paper represent initial investigations of RPP in a distance learning context. As such, they are subject to limitations. The small number of self-selecting students does not necessarily represent our student population, or distance learners generally. We do not claim any statistical significance or general applicability of our results. However, the studies uncover RPP barriers and benefits perceived by these keener students, and we assume that these barriers and benefits may be felt even more acutely by other students.

We didn't develop bespoke programming exercises for the studies. Instead, we asked students to work in pairs through their module's activities, so as not to impose extra workload on already time-poor students. The responses suggest this approach isn't always suitable. Overall, the reasons indicated for dissatisfaction suggest that better integration of RPP into the course design process is needed to reap the potential benefits of RPP.

Questions around the effect of leadership style on the pair jelling period deserve a more systematic examination. In our study 1, a definitive conclusion cannot be drawn due to the small sample size and various variables involved during the investigation. Data from study 2, which is yet to be analysed, should shed more light on this issue. Further long-term studies could indicate if pair jelling has an impact on the effectiveness of RPP. If so, monitoring pair jelling could help spot conflicts early and drive a methodology for conflict management to ensure collaboration does not break down.

Future studies using larger samples could shed further light on the other aspects of pairing we investigated: prior skills level and gender. We also plan to carry out more quantitative investigations of the learning effectiveness of RPP, using before-and-after measures of students' programming skills, and making comparisons with individual programming practice.

5. Conclusion

Many studies show evidence that pair programming in Computing education can have positive effects on technical and soft skills. Pair programming is relatively easy to embed into scheduled full-time study. This paper presents exploratory studies into the possible benefits of, and barriers to, remote pair programming in unscheduled part-time distance education.

While many RPP studies use specially built tools, these may be difficult to obtain or install. We asked students to use generic communication tools to work on their module's exercises with a partner in several RPP sessions. We analysed students' pre-and post-study questionnaires, recorded sessions and reflective journals. In the first study, students were paired according to their declared time availability; in the second, they were randomly paired. We also carried out the first, as far as we know, investigation of pair jelling and leadership style.

Students found the tools easy to use and install, and broadly adequate for RPP, despite occasional technical problems. Students agreed that RPP improved their coding and debugging skills and their confidence. Responses as to whether RPP improved their time management or study habits varied. Almost all students would recommend RPP for inclusion in distance learning computing modules. From our limited amount of data, leadership style seemed to be a more dominant factor for pair jelling than programming skills level. Students were less satisfied or even stopped sessions if there were scheduling difficulties with their partner or if they felt their module's exercises were not suitable for pair programming.

Our research suggests that RPP can enhance the learning experience in part-time distance education, provided that potential barriers are dealt with by: providing guidance on how best to use existing communication tools instead of developing bespoke ones, taking time availability and leadership style into account when pairing students, and choosing appropriate programming activities.

ACKNOWLEDGMENTS

We thank the participants and the funding support of eSTEeM, the OU's Centre for STEM pedagogy.

REFERENCES

1. Al-Ani Ban, Erik Trainer, Roger Ripley, Anita Sarma, André van Der Hoek, and David Redmiles, 2008. Continuous coordination within the context of cooperative and human aspects of software engineering. In *Proc. Ws on Cooperative and Human Aspects of Software Engineering*. ACM, 1-4.
2. Berenson Sarah, Kelli Slaten, Laurie Williams, and Chih-Wei Ho, 2004. Voices of women in a software engineering course: reflections on collaboration. *Journal on Educational Resources in Computing (JERIC)* 4(1): 3-es.
3. Biehl Jacob, Mary Czerwinski, Greg Smith, and George Robertson, 2007. FASTDash: A visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI Conference on human factors in computing systems*. ACM**.** 1313-1322.
4. Canfora Gerardo, Cimitile Aniello and Visaggio Corrado Aaron, 2005. Empirical Study on the Productivity of the Pair Programming. In *Extreme Programming and Agile Processes in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 92–99. DOI: https://doi.org/10.1007/11499053_11.
5. Canfora Gerardo, Cimitile Aniello and Visaggio Corrado Aaron, 2003. Lessons learned about distributed pair programming: what are the knowledge needs to address? In *Proceedings Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. WET ICE, 314–319.
6. D'Angelo Sarah and Andrew Begel, 2017. Improving Communication Between Pair Programmers Using Shared Gaze Awareness. In *Proceedings of the SIGCHI Conference on human factors in computing systems*, ACM**,** 6245-6290.
7. da Silva Estácio, Bernardo José and Rafael Prikladnicki, 2015. Distributed pair programming: A systematic literature review. *Information and Software Technology*. 63, 1-10
8. Daniel Weinbren, 2015. *The Open University: A history*. Manchester University Press, Manchester.
9. Duque Rafael and Bravo Crescencio, 2008. Analyzing Work Productivity and Program Quality in Collaborative Programming. In *Proc Int'l Conf. on Software Engineering Advances***:** 270-276. DOI: https://doi.org/10.1109/ICSEA.2008.82
10. Flor Nick, 2006. Globally distributed software development and pair programming. *Communications of the ACM* 49(10), 57-58.
11. Gehringer Edward, 2003. A pair-programming experiment in a non-programming course. *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications.* ACM**,** 187-190.
12. Ghorashi Soroush and Carlos Jensen, 2017. Integrating Collaborative and Live Coding for Distance Education. *Computer (Long Beach, Calif.)*, 50(5), 27-35.
13. Ghorashi Soroush and Carlos Jensen, 2016. Supporting Learners in Online Courses Through Pair Programming and Live Coding. *IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)***.** 738-747.
14. Goldman Max, 2011. Role-based interfaces for collaborative software development. In *Proceedings of the 24th annual ACM symposium adjunct on user interface software and technology*. ACM, 23-26.
15. Hanks Brian, 2005. Student performance in CS1 with distributed pair programming. *ACM SIGCSE Bulletin*, 37(3), 316-320 DOI: https://doi.org/10.1145/1067445.1067532.
16. Hanks Brian, 2008. Empirical evaluation of distributed pair programming. In *International Journal of Human-Computer Studies* 66(7), 530-544. DOI: https://doi.org/10.1016/j.ijhcs.2007.10.003
17. Hughes Janet, Ann Walshe, Bobby Law and Brendan Murphy, 2020. Remote pair programming. In *Proc.12th Int'l Conf. on Computer Supported Education,* Vol. 2, SciTePress, 476-483.
18. Jarvenpaa L. Sirkka and Dorothy E. Leidner, 1999. Communication and Trust in Global Virtual Teams. *Organization Science* 10(6), 791-815.
19. Katira Neha, Laurie Williams, Eric Wiebe, Carol Miller, Suzanne Balik and Ed Gehringer,2004. On understanding compatibility of student pair programmers. In *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)*, 36(1), 7–11.
20. Kaur K. Sandeep, Kevin Gerstner and Alexandra Bejarano, 2019. Remote Pair Programming in Online CS Education: Investigating through a Gender Lens. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE**,** 75-85.
21. Kear Karen and Helen Donelan, 2018. Creating and collaborating: students' and tutors' perceptions of an online group project. In *International Review of Research in Open and Distributed Learning* 19(2), 38-54.
22. Knister Michael and Atul Prakas, 1990. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of the 1990 ACM conference on computer-supported cooperative work*. ACM**,** 343-355.
23. Leelanupab Teerapong and Tiwipab Meephruek, 2019. CodeBuddy (Collaborative Software Development Environment): In- and Out-Class Practice for Remote Pair-Programming with Monitoring Coding Students' Progress. In *Technical Symposium on Computer Science Education*, ACM**,** 1290-1290.
24. Lui K. Man, Barnes K. Atikus and Chan C. Keith, 2010. Pair Programming: Issues and Challenges. *Agile Software Development: Current Research and Future Directions*. Springer, Berlin, Heidelberg, 143-163. https://doi.org/10.1007/978-3-642-12575-1_7
25. McDowell Charlie, Linda Werner, Heather Bullock and Julian Fernald, 2003. The impact of pair programming on student performance, perception and persistence. In *Proceedings of 25th International Conference on Software Engineering*. IEEE**,** 602-607.
26. McDowell Charlie, Linda Werner, Heather Bullock and Julian Fernald, 2006. Pair programming improves student retention, confidence, and program quality. In *Communications of the ACM* 49(8), ACM, 90-95.
27. Mehmet Celepkolu and Kristy E. Boyer, 2018. The Importance of Producing Shared Code Through Pair Programming. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education.* ACM, 765-770. DOI: https://doi.org/10.1145/3159450.3159516
28. Melnik Grigori and Frank Maurer, 2002. Perceptions of agile practices: A student survey. In *Proc. Conf. on XP and AGile Methods*, Springer, LNCS 2418**,** 241-250.
29. Mendes Emilia, Lubna Al-Fakhri and Andrew Luxton-Reilly, 2005. Investigating pair-programming in a 2nd-year software development and design computer science course. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ACM, 296-300.
30. Mendes Emilia, Lubna Al-Fakhri and Andrew Luxton-Reilly, 2006. A replicated experiment of pair-programming in a 2nd-year software development and design computer science course. In *Proceedings of the 11th annual SIGCSE conference on innovation and technology in computer science education*. ACM, 108-112.
31. Nagappan Nachiappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller and Suzanne Balik, 2003. Improving the CS1 experience with pair programming. *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)*, 35(1). ACM, 359-362.
32. Nosek John, 1998. The case for collaborative programming. *Communications of the ACM*, 41(3). ACM, 105-108.
33. Porter Leo, Mark Guzdial, Charlie McDowell and Beth Simon, 2013. Success in introductory programming: what works? *Communications of the ACM***,** 56(8). ACM, 34-36.
34. Shaw C. Alan 2009. Extending the Pair Programming Pedagogy to Support Remote Collaborations in CS Education. In *2009 Sixth International Conference on Information Technology: New Generations*. IEEE**,** 1095-1099.
35. Sison Raymund, 2009. Investigating the Effect of Pair Programming and Software Size on Software Quality and Programmer Productivity. In *2009 16th Asia-Pacific Software Engineering Conference*. IEEE**,** 187-193.
36. Stefik Mark, Bobrow Daniel, Foster Gregg, Lanning Stan and Tatar Deborah, 1987. WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Transactions on Information Systems (TOIS)* **5**(2). ACM, 147-167. DOI: https://doi.org/10.1145/27636.28056
37. Tsompanoudi Despina, Satratzemi Maya and Xinogalos Stelio, 2013. Exploring the effects of collaboration scripts embedded in a distributed pair programming system. In *Annual Joint Conference Integrating Technology into Computer Science Education*, ACM**,** 225-230.
38. Vanhanen Jari. and Korpi Harri, 2007. Experiences of Using Pair Programming in an Agile Project. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07).* IEEE, 274b–274b.
39. Wanfeng Dou. and Wei He, 2010. Compatibility and Requirements Analysis of Distributed Pair Programming. In *2010 Second International Workshop on Education Technology and Computer Science*, IEEE, 467-470.
40. Werner L. Linda., Hanks Brian and McDowell Charlie, 2004. Pair-programming helps female computer science students. In *Journal on Educational Resources in Computing* 4(1): 4-es. DOI: https://doi.org/10.1145/1060071.1060075
41. Williams A. Laurie and Kessler Robert, 2003. *Pair Programming Illuminated.* Addison-Wesley Longman Publishing Co., Inc., USA..
42. Williams A. Laurie and Kessler Robert, 2000. The effects of 'pair-pressure' and 'pair-learning' on software engineering education. In *Thirteenth Conference on Software Engineering Education and Training*, IEEE, 59–65. DOI: https://doi.org/10.1109/CSEE.2000.827023.
43. Williams A. Laurie, Kessler Robert, Cunningham Ward and Jeffries Ron, 2000. Strengthening the case for pair programming. *IEEE Software*, 17(4), IEEE, 19-25. DOI: https://doi.org/10.1109/52.854064
44. Wray Stuart, 2010. How Pair Programming Really Works. *IEEE Software*, 27(1). IEEE, 50-55. DOI: https://doi.org/10.1109/MS.2009.199
45. Xinogalos Stelio, Satratzemi Maya, Chatzigeorgiou Alexander and Tsompanoudi Despina, 2019. Factors Affecting Students' Performance in Distributed Pair Programming. In *Journal of educational computing research* 57(2), 513-544.