Is AI the better programming partner? Human-Human Pair Programming vs. Human-AI pAIr Programming*

Qianou Christina Ma[1,*], Sherry Tongshuang Wu[1] and Ken Koedinger[1]

[1]*Carnegie Mellon University (CMU), Pittsburgh, PA, United States*

Abstract

The emergence of large-language models (LLMs) that excel at code generation and commercial products such as GitHub's Copilot has sparked interest in human-AI pair programming (referred to as "pAIr programming") where an AI system collaborates with a human programmer. While traditional pair programming between humans has been extensively studied in both industry and education, it remains uncertain whether its findings can be applied to human-AI pair programming. We compare interaction, measures, benefits, and challenges of human-human and human-AI pair programming. We find that the effectiveness of both approaches is mixed in the literature (the measures used for pAIr programming are not as comprehensive). We summarize moderating factors on the success of human-human pair programming, which provide opportunities for pAIr programming. For example, mismatched expertise makes pair programming less productive, therefore well-designed AI programming assistants may adapt to differences in expertise levels. Finally, we discuss the potential of using LLMs to provide effective pAIr programming learning for students at scale.

Keywords

Pair Programming, Large Language Model (LLM), Copilot, AI-based Programming Assistant

1. Introduction

Pair programming describes the practice of two programmers working together on the same task using a single computer, keyboard, and mouse. One programmer in the pair, the "driver," performs the coding (typing) and implements the task, while the other programmer, the "navigator," aids in planning, reviewing, debugging, and suggesting improvements and alternatives. Now, pair programming is used in a wide range of settings, including education, industry, and open-source software development [1, 2]. For example, in education, human-human pair programming has been adopted from K12 [3], CS1 [4] to higher-level project-based courses [5].

Recent advances in code-generating large-language models (LLMs) have led to the widespread popularity of commercial AI-powered programming assistance tools such as GitHub Copilot [6], which advertises itself as "your AI pair programmer." Instead of two humans working on a single computer, it is the programmer and the LLM-based AI that work together on the same task. The shift in the paradigm raises the questions: *Is the AI programming partner comparable*

Empowering Education with LLMs – the Next-Gen Interface and Content Generation, July 7, 2023, Tokyo, Japan

*Corresponding author.

qianouma@cmu.edu (Q. C. Ma); sherryw@cs.cmu.edu (S. T. Wu); koedinger@cmu.edu (K. Koedinger)

to a human pair programmer? Can they achieve similar or better performance, and should people interact with them in the same way?

The question of whether AI can serve as a better programming partner is crucial. Understanding the comparative performance of AI and human programmers in a pair programming context can guide developers and educators in utilizing the most effective collaboration methods. Understanding the potential role and design of AI in pair programming can help educators design more scalable pedagogical approaches to promote student learning and engagement in programming. Furthermore, identifying the strengths and weaknesses of human and AI programming partners may in turn contribute to the refinement and development of better AI programming tools that augment human programmers' capabilities.

Based on our readings of four existing meta-analysis papers on human-human pair programming and over 50 studies on the topic of pair programming or AI-assisted programming, we dive into comparisons of measurements of success (Section 2), as well as moderators, e.g., pair compatibility factors like expertise (Section 3). We find that (1) prior work on both pair programming paradigms has observed mixed results in *quality*, *productivity*, *satisfaction*, *learning*, and *cost*, (2) human-AI pair programming has yet to develop comprehensive measurements, and (3) key factors to pAIr's success have been largely unexplored.

Building on our exploration, we elaborate on future opportunities for developing best practices and guidelines for pAIr programming (Section 4). First, we argue that moderating factors that bring challenges to human-human pair programming (e.g., compatibility and communication) unveil opportunities to improve human-AI pair programming. It can be promising to exploit the differences between a human and an AI partner (e.g., more customizable expertise level and more adaptable communication styles) to design for more successful pAIr programming experiences. Second, we encourage future research to explore the best deployment environment for pAIr programming, such as education. We hope this paper can inspire better evaluations and designs of code-generating LLMs as a pAIr programmer, especially for students.

2. Mixed Outcomes

Literature reviews have suggested various benefits as well as mixed effects of human-human pair programming [29, 1, 2]. According to Alves De Lima Salge and Berente [1], pair programming improves code quality, productivity, and learning outcomes. However, according to Hannay et al. [29], pair programming improves quality and shortens duration, but it increases effort, higher quality comes at the expense of considerably greater effort, and reduced completion time comes with lower quality. In the education context, pair programming brings benefits including higher quality software, student confidence in solutions, increased assignment grades, exam scores, success/passing rates in introductory courses, and retention [2, 30, 19]. All the reviews acknowledged that even though meta-analyses can show a significant effect size, individual studies could report contradictory outcomes (see examples in Table 1).

For human-AI pair programming, existing works mainly focus on quality, productivity, and satisfaction, and already demonstrated mixed results in quality and productivity [9, 31, 12] (see examples in Table 1). Additionally, some measures are arguably too simplified as evaluation metrics. For example, Imai [9] used the number of lines of added code as the measure of

Table 1
Comparison of Outcome Variables and Moderators for Human-Human Pair Programming vs. Human-AI pAIr Programming

Outcomes	**Human-Human vs. Human Solo**	**Human-AI (Copilot)**
Quality	🙂 significantly lower defect density for complex code [7] 😐 no difference for simpler code [7] 🙂 significantly higher percentage of test cases passed [8]	☹ vs. Human-Human: more lines of code deleted in next session (lower quality) [9] 🙂 vs. Human Solo: significantly improve correctness score and reduce encountered errors for novice students [10] 😐 vs. Human Solo: no significant difference in task success [11] or task success rate in given time [12]
Productivity	☹ significantly fewer lines of code per person hour writing simpler code [7] 😐 no significant difference writing more complex code [7] 🙂 29% shorter time to complete task (pair speed advantage = 1.4) [13]	🙂 vs. Human-Human: more lines of added code [9] 🙂 vs. Human Solo: 55.8% reduction in completion time [11] 🙂 vs. Human Solo: significantly increase task completion and reduce task completion time for novice students [10] 😐 vs. Human Solo: no significant difference in the task completion rate in given time [12]
Satisfaction	🙂 higher self-ratings of satisfaction [14] ☹ students with greater self-confidence and self-efficacy less enjoy the pair programming experience [15]	🙂 vs. Human Solo: higher self-ratings of satisfaction [12, 16, 17]
Learning	🙂 higher grades, exam scores [18], and retention [19] 🙂 significantly higher gains in exam performance in female students than male students [20]	😐 vs. Human Solo: no significant difference in immediate and retention post-test performance of novices, students with more prior experiences have more learning gains from AI code generator [10]
Cost	☹ increased management workload to match, schedule a pair, resolve collaboration conflict, assess individual contributions, etc. [21] 🙂 reduced teaching staff workload (grading one assignment from a pair) [8]	No experiment yet. Vaithilingam et al. [12], Bird et al. [16] hypothesized that human-AI may lead to more unnecessary debugging vs. Human Solo

Moderators	**Human-Human vs. Human Solo**	**Human-AI (Copilot)**
Task Types & Complexity	Complex task improve quality, simple one does not [7]; debugging is perceived as less enjoyable or effective than comprehension or refactoring [22]	N/A
Compatibility (E.g., Expertise)	Random pairing led to incompatible partners and conflicts during work [18]. Expertise: improve quality more effectively if pair is similarly skilled [14]; less-skilled students learn more and enjoy more [20, 22]; if knowledge gap is large, less-skilled programmers may tend to be more passive and disengaged [23]	N/A
Communication	Conversations with intermediate-level details contribute to pair programming success [24]; different types of discourse lead to more attempts or more debug success [25]	N/A
Collaboration	Over-reliance leads to conflicts and impedes satisfaction and learning, as work is entirely burdened on one partner [4, 18]; educators recommend regular role-switching to ensure equitable learning in collaboration [2]	N/A
Logistics	Scheduling difficulties [26], teaching & evaluating individual responsibility and accountability are important to collaboration success [27], but can lead to increased management costs [21, 28]	N/A

productivity; however, the nature of interaction with Copilot (tab to accept suggestions) is likely to contribute to more added lines in the human-Copilot condition, and how valid would it represent the notion of productivity is questionable.

Since there is not enough research for a comprehensive review of human-AI pair programming, we cannot reach any conclusion on pAIr effectiveness yet. It is also hard to compare the human-human and human-AI pair programming literature, as they differ in what outcomes and measurements they adopt. Therefore, in the top rows of Table 1, we listed the most common outcome variables in both literature (*quality*, *productivity*, *satisfaction*, *learning*, and *cost*) and some sample works to demonstrate mixed outcomes and example measures that operationally define the outcomes. For more details, please refer to our more elaborated work [32].

Note that researchers can use different words to characterize similar constructs. For example, duration, effort, and productivity are all types of "efficiency" outcomes that involve time and accomplishment. Productivity can be measured in terms of the number of completed tasks in a fixed unit of time, duration can be measured as the amount of elapsed or total time used to complete a fixed number of tasks to a certain standard, and effort can be measured as twice the duration, the person-hours required, etc. [1]. We use productivity as an aggregated outcome variable of different measures, for consistency with the human-AI literature.

3. Moderators

In search of the explanations of the cost-benefit of human-human pair programming experiences, researchers have found moderators such as *task type & complexity* [29], *compatibility* factors like expertise [27, 33], *communication* [34, 24, 23], *collaboration* factors like over-reliance and role-switching [4, 35, 14], and *logistics* difficulties including scheduling and training [26, 29] (as shown in the bottom rows of Table 1). For human-AI pair programming's moderators, much was unexplored – we do not know what could make human-AI pair programming more or less effective. Therefore, in this section, we discuss the key moderators that are examined in the human-human pair programming literature, and individual examples of moderating effects are provided in Table 1.

3.1. Task Types & Complexity

For task type and task complexity, Chaparro et al. [22] found that debugging tasks lead to less satisfaction and perceived efficacy compared to comprehension and refactoring tasks. Hannay et al. [29] found that the duration is shorter for low complexity tasks, at the expense of lower quality results, and quality is higher when complexity is higher, but it requires considerably greater effort. Arisholm et al. [33] found that the moderating effect of complexity also depends on the expertise of the pair, where "benefits of correctness on complex system apply mainly to juniors, whereas the reductions in duration to perform the tasks correctly on the simple system apply mainly to intermediates and seniors."

3.2. Compatibility

Salleh et al. [14] listed multiple factors for pair compatibility, such as personality, perceived skills, actual skills (expertise), self-esteem, gender, and work ethic. Thomas et al. [15] found that paired students with similar self-confidence levels produce their best work. Hannay et al. [35] found that Big Five personality traits only have modest predictive value on pair programming performance, in comparison to expertise, task complexity, and country. There also seems to be evidence that women benefit from pair programming more than men [27, 30].

Expertise as a compatibility factor has been extensively studied. For example, researchers found that a student pair performs the best when their expertise is similar [14] and students preferred to be paired with similarly skilled partners [22]. However, in industry, Jensen [36] reported that when both members were near the same capability level and strongly opinionated, the collaboration was counter-productive and troublesome.

In the introductory programming context, Lui and Chan [37] found that pairing up novices results in a larger improvement in productivity than pairing up experts. However, there are concerns about "the blind leading the blind" if they don't have an expert to consult with [21]. Researchers also found that less-skilled students learn and enjoy more than more-skilled students in pair programming [22, 20]. However, when the knowledge gap is too large, students can be less satisfied and the benefits of quality may be smaller [13]. Chong and Hurlbutt [23] reported that a novice programmer collaborating with an expert may become disengaged, have lower self-esteem, and be afraid of slowing down or annoying their more-skilled partner [21].

3.3. Communication

According to Freudenberg et al. [24], "the key to the success of pair programming [is] the proliferation of talk at an intermediate level of detail in pair programmers' conversations." Researchers found that pair programming eliminates distracting activity and enables programmers to focus on productive activity [38], which could be why engaging communications contribute to successful pair programming. Murphy et al. [25] used transactive analysis to break down communication by different types of transactions and found that attempting more problems associated with more completion and debugging success correlated with more critique transactions. Some other works pointed out the social support aspect of communication [23] and an explanation effect where the verbalization of the thought process makes thinking clearer [16].

In human-human pair programming, programmers spend about 1/3 of the time primarily focusing on communication [34], which forces them to concentrate, rationalize, and explain their thoughts [38, 29]. In human-AI pair programming, Mozannar et al. [39] has shown that an analogous 1/3 amount of time is spent communicating with Copilot, such as thinking and verifying (22.4%) Copilot's suggestion, which may be replicating the self-explanation effects in some ways, and prompt crafting, which takes 11.56% of the time. These activities are arguably efforts to understand and communicate with Copilot. However, there is no other human to co-verify the answers, and there is no study that evaluate the communicative nature of human-Copilot interaction as human-human pair programming.

3.4. Collaboration

Collaboration can fail in various ways in a human-human pair. For example, the free-rider problem, where the entire workload is on one partner while the other remains a marginal player, can result in less satisfaction and learning [4, 18]. In human-AI pair programming, educators are worried that easily available code-generation tools may lead to cheating, and over-reliance on AI may hinder students learning [40]. However, no study has formally evaluated it yet.

For human-human pair programming, there is a suggested collaboration pattern of role-switching – two software developers periodically and regularly switch between writing code (driver) and suggesting code (navigator), aiming to ensure that both are engaged in the task and alleviate the physical and cognitive load borne by the driver [1, 34]. Some researchers

Freudenberg et al. [24] argue that the success of pair programming should be attributed to communication rather than “the differences in behavior or focus between the driver and navigator,” as they found both driver and navigator worked on similar levels of abstraction. Nevertheless, instructors still recommend drivers and navigators to regularly alternate roles to ensure equitable learning experiences [2].

In human-AI interaction, given Copilot’s amazing capability to write code in different languages, some have argued that Copilot can take on the role of the “driver” in pair programming, allowing a solo programmer to take on the role of the “navigator” and focus on understanding the code at a higher level [9]. However, while it is possible for humans to offload some API lookup and syntax details to Copilot, humans still need to jump back into the driver’s seat frequently and fluidly switch between the thinking and writing activities [39]. It is ultimately the human programmer’s sole responsibility to understand the code at the statement level [41].

3.5. Logistics

Logistical challenges, including scheduling difficulties, teaching and evaluating collaboration for the pair, and figuring out individual accountability and responsibility [26, 27], can add to the management cost of human-human pair programming [21, 28].

In human-AI pair programming, some may argue that the human is solely responsible in the human-AI pair [41], but the accountability of these LLM-based generative AI is still under debate [40]. There may be new logistics issues for the human-AI pair, such as teaching humans how to best collaborate with Copilot. There could also be unique challenges as in every human-AI interaction scenario, such as bias, trust, and technical limitations – much to be explored. More study would be needed to empirically and experimentally verify the moderating effects of different variables in human-AI pair programming.

4. Discussion and Future Work

4.1. LLM, A Better pAIr Programmer?

As reviewed in Section 2, previous literature has explored a variety of measures to evaluate different aspects of human-human pair programming, while the current exploration in human-AI pair programming is quite limited. Murillo and D’Angelo [42] have proposed evaluation metrics for LLM-based creative code writing assistants in software engineering. More works could use more valid measures in the human-human pair programming literature to explore how to best help humans and LLM-based AI programming assistant collaborate together. It would also be interesting to have a study setup with three conditions – human-human, human-AI, and human solo – working on the same task.

Note that in this paper, we mostly covered studies using the VSCode Extension Copilot. Tools like ChatGPT may support the communication aspect better than Copilot [43], and there are also Bard developed by Google [44] and an experimental version of Copilot Labs by Github [45], which support more functionalities such as fix bug, clean, and customizable prompts. Those tools may already improve the human-AI pair programming interaction in some ways, so future studies could also compare across a variety of LLM-based programming tools.

Table 2
Challenges in Human-Human Pair Programming *Yield* Opportunities for Human-AI pAIr Programming

Moderating Factors	Human-Human Challenges	Human-AI Opportunities
Task Types & Complexity: pair work better if the task is not too simple and good for collaboration [7, 22]	Hard to design suitable tasks of appropriate complexity level	AI may be used to generate collaboration tasks and adjust tasks complexity
Compatibility: pairs with similar skill levels and compatible working styles work better [14, 22]	Hard to find a similarly skilled or compatible partner	AI partner should adjust to human skill level and adapt to be compatible with different people
Communication: pairs work better with productive conversations [24], and critiques lead to more debugging success [25]	Hard to teach effective communication and constructive criticism	AI partner should support productive conversations and provide critiques
Collaboration: pairs work better with positive interdependence [27] and clear and balanced responsibilities [18]	Hard to teach collaboration and prevent free riders	AI should support positive social interactions and collaboration and avoid over-assist that eliminates human's need to engage
Logistics: pair programming is costly to implement because of management challenges [21, 28]	Hard to schedule and assess individual contributions in a pair	Scheduling is no longer a problem, but humans should be accountable and responsible when using AI-generated code

Previous literature suggested some key factors in the success of human-human pair programming, as summarized in Table 1. These moderators that cause challenges for human-human pair programming may yield opportunities to explore in human-AI pair programming (Table 2). For example, self-efficacy can lead to a difference in satisfaction [15] and gender can lead to a difference in learning [20], do these compatibility moderators influence pAIr too? Can we improve pAIr outcomes using insights derived from human-human literature (e.g., simulate an AI partner with similar self-efficacy levels and the same gender)? We discuss more details on how and why might LLM be used in ways presented in Table 2 in our later work [32].

Therefore, in general, we can ask the following questions for future works: could these factors be implemented for human-AI pair programming; would they make human-AI pair programming more effective, less effective, or have no influence, and why?

4.2. LLM, Students' pAIr Programmer?

Most current studies that evaluate the efficacy of Copilot are conducted with experienced software developers. If we estimate Copilot's problem-solving abilities as an average student in introductory programming classes, evaluating its performance when pairing up with a professional software developer with much more expertise may not bring enough benefit to the professional. Therefore, working with LLM's current capabilities, it seems like a student-AI pair programming setup would be the most promising to explore, so the next question is: how should we best support student-AI pair programming?

Re-prioritize programming skills. First of all, co-working with AI requires a special skill set, and future work could explore how to support students to better develop these crucial skills.

Bird et al. [16] argued that the popularity of LLM-based programming assistants will result in the growing importance of reviewing code as a skill for developers. Nonetheless, in Perscheid et al. [46]'s interview, none of the professional developers remembered training on debugging at school. There is already rich literature on debugging and testing instructions [47, 48, 49], but logistical challenges like the lack of instructional time still exist [49, 50], and educators need to better prepare students with debugging and testing skills needed to work with unreliable AI.

Integrate AIEd frameworks. Holstein et al. [51] developed a framework to map ways to mutually augment humans and AI in education, for example, by augmenting interpretation, action, scalability, and capacity. Future works can use existing theories in the AI education space to improve the design of the AI pAIr programming partner, and further investigate if LLMs bring new focus and affordances to previous human-AI education frameworks.

Support explanation and communication with students. Previous attempts of using AI agent as pair programming partner have shown some preliminary success in knowledge transfer and retention [52, 53], and the limitation discussed was the lack of discussion and explanation [54]. Nowadays, as an LLM-based agent can support more natural interaction and provide good quality explanations in the introductory programming context [55], it would be interesting to explore if LLM-based AI could resolve some limitations mentioned in pedagogical and conversational agent works before. Self-reflection and explanation techniques may also be adopted to make up for the communication aspect as in human-human pair programming.

Match expertise with students. Furthermore, as discussed in Section 3, matching expertise is a tricky problem. Lui and Chan [37] found that expert-expert pair may not gain as much of an advantage over an expert solo programmer, in comparison to novice-novice pair vs. a solo novice. Meanwhile, pairing two novices together raise concerns of "the blind leading the blind," but pairing a novice with an expert may lead to lower self-esteem of the novice [21]. Given all these complexities, when it comes to a student-AI pair and when we only care about the student's learning gains, there are a lot of research questions to ask. If we have full control of the perceived skill level of the AI partner, should we configure it to be similar to the student, slightly higher skilled, or a lot better? Would it be beneficial to have both a peer AI agent but also a tutor AI agent to assist if students get stuck?

Avoid over-helping students. Note that for programming learners, it would be important to configure the LLM-based programming assistant to avoid over-help. In the few studies that examined novice interaction with Copilot [56] or a customized programming environment based on LLM-based code generation model Codex [10]. Prather et al. [56] found that novices do have unique interaction patterns with Copilot and a tendency to rely on and trust the generated code too much. Kazemitabaar et al. [10] discussed design implications including control over-use and support complete novices. There have also been concerns about academic integrity and changing perception of learning when LLM-based programming tools become easily accessible to students [40, 57, 56], which are issues to further explore for student-AI pair programming.

Boost students' self-confidence. Pair programming has been shown to benefit students with lower self-efficacy and self-confidence levels [15] and women [20] more, which could make it a pedagogical tool to engage more vulnerable or underrepresented populations in CS. When an AI is introduced in pair programming, would the same benefits retain? How should we present the AI differently to make it compatible with students with different confidence levels? How do we mitigate the risks of unreliable but seemingly authoritative AI?

Address critical risks in using LLM. Last but not least, LLMs may be an opportunity to address some existing challenges that student-student pair programming has (as summarized in Table 2), but there are critical risks associated with using generative AI in education, such as bias, trust, and transparency [58]. Evaluation on existing benchmarks shows that ChatGPT exhibits ethnically risky behaviors and presents inaccurate information [59]. Therefore, to ethnically and responsibly use AI as a student's pair programming partner, besides keep improving model designs, we also need to keep human instructors' supervision and education students about AI's limitations.

5. Conclusion

This paper has discussed the concept of human-AI pair programming (pAIr programming). Research has yet to pinpoint which of these supposed advantages of human-AI pair programming yields the largest benefits in efficiency and learning. Human-human pair programming literature yield insights on what outcomes and measures should researchers use to evaluate their pAIr programming work (e.g., use more valid quality and productivity measurements, and further investigate cost), and what moderators should researchers consider to further analyze and improve pAIr programming's process and design (e.g, compatibility, communication, etc.).

In conclusion, more valid and comprehensive measurements are needed to evaluate pAIr programming, more comparisons can be drawn between human-human vs. human-AI pair programming, and more works can explore how to best support LLM-assisted programming with insights from the rich literature on human-human pair programming.

Acknowledgments

Thanks to Ken's lab members for giving feedback on this work. Thanks to Stephen MacNeil for coming up with the creative "pAIr" keyword for this project.

References

[1] C. Alves De Lima Salge, N. Berente, Pair programming vs. solo programming: What do we know after 15 years of research?, in: 2016 49th Hawaii International Conference on System Sciences (HICSS), 2016, pp. 5398–5406. URL: http://dx.doi.org/10.1109/HICSS.2016.667. doi:10.1109/HICSS.2016.667.

[2] K. Umapathy, A. D. Ritzhaupt, A Meta-Analysis of Pair-Programming in computer programming courses: Implications for educational practice, ACM Trans. Comput. Educ. 17 (2017) 1–13. URL: https://doi.org/10.1145/2996201. doi:10.1145/2996201.

[3] X. Wei, L. Lin, N. Meng, W. Tan, S.-C. Kong, Others, The effectiveness of partial pair programming on elementary school students' computational thinking skills and self-efficacy, Comput. Educ. 160 (2021) 104023. URL: https://www.sciencedirect.com/science/article/pii/S0360131520302219?casa_token=BCC-5YvaAgAAAAAA:lIOmfQPTBGpeL296DBtelqwPs7pNeHKuugy8gsVK0Nn6bha9pdJIMm2G3JLUNXTwTUZavVA2Fw.

[4] L. Williams, E. Wiebe, K. Yang, M. Ferzli, C. Miller, In support of pair programming in the introductory computer science course, Comput. Sci. Educ. 12 (2002) 197–212. URL: http://www.tandfonline.com/doi/abs/10.1076/csed.12.3.197.8618. doi:10.1076/csed.12.3.197.8618.

[5] S. Xu, V. Rajlich, Pair programming in graduate software engineering course projects, in: Proceedings Frontiers in Education 35th Annual Conference, IEEE, 2006. URL: http://ieeexplore.ieee.org/document/1612027/. doi:10.1109/fie.2005.1612027.

[6] GitHub, Your AI pair programmer: Copilot, https://github.com/features/copilot, 2021. URL: https://github.com/features/copilot, accessed: 2022-10-5.

[7] R. Sison, Investigating the effect of pair programming and software size on software quality and programmer productivity, in: 2009 16th Asia-Pacific Software Engineering Conference, 2009, pp. 187–193. URL: http://dx.doi.org/10.1109/APSEC.2009.71. doi:10.1109/APSEC.2009.71.

[8] L. Williams, R. L. Upchurch, In support of student pair-programming, in: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, ACM, New York, NY, USA, 2001. URL: https://collaboration.csc.ncsu.edu/laurie/Papers/WilliamsUpchurch.pdf. doi:10.1145/364447.364614.

[9] S. Imai, Is GitHub copilot a substitute for human pair-programming? an empirical study, in: 2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), ieeexplore.ieee.org, 2022, pp. 319–321. URL: http://dx.doi.org/10.1145/3510454.3522684. doi:10.1145/3510454.3522684.

[10] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, T. Grossman, Studying the effect of AI code generators on supporting novice learners in introductory programming (2023). URL: http://arxiv.org/abs/2302.07427. arXiv:2302.07427.

[11] S. Peng, E. Kalliamvakou, P. Cihon, M. Demirer, The impact of AI on developer productivity: Evidence from GitHub copilot (2023). URL: http://arxiv.org/abs/2302.06590. arXiv:2302.06590.

[12] P. Vaithilingam, T. Zhang, E. L. Glassman, Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models, in: Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems, number Article 332 in CHI EA '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 1–7. URL: https://doi.org/10.1145/3491101.3519665. doi:10.1145/3491101.3519665.

[13] V. V. K. Padmanabhuni, H. P. Tadiparthi, S. M. Muralidhar Yanamadala, Effective pair programming practice-an experimental study, Journal of Emerging Trends in Computing and Information Sciences 3 (2012) 471–479. URL: http://www.agilemethod.csie.ncu.edu.tw/agileMethod/download/2012papers/2012%

20Effective%20Pair%20Programming%20Practice-%20An%20Experimental%20Study/Effective%20Pair%20Programming%20Practice-%20An%20Experimental%20Study.pdf.

[14] N. Salleh, E. Mendes, J. Grundy, Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review, IEEE Trans. Software Eng. 37 (2011) 509–525. URL: http://dx.doi.org/10.1109/TSE.2010.59. doi:10.1109/TSE.2010.59.

[15] L. Thomas, M. Ratcliffe, A. Robertson, Code warriors and code-a-phobes: a study in attitude and pair programming, SIGCSE Bull. 35 (2003) 363–367. URL: https://doi.org/10.1145/792548.612007. doi:10.1145/792548.612007.

[16] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, I. Gazit, Taking flight with copilot: Early insights and opportunities of AI-powered pair-programming tools, Queueing Syst. 20 (2023) 35–57. URL: https://doi.org/10.1145/3582083. doi:10.1145/3582083.

[17] E. Kalliamvakou, Research: quantifying GitHub copilot's impact on developer productivity and happiness, https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/, 2022. URL: https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-producti accessed: 2022-10-13.

[18] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, S. Balik, Improving the CS1 experience with pair programming, in: Proceedings of the 34th SIGCSE technical symposium on Computer science education, ACM, New York, NY, USA, 2003. URL: https://dl.acm.org/doi/10.1145/611892.612006. doi:10.1145/611892.612006.

[19] C. McDowell, L. Werner, H. E. Bullock, J. Fernald, Pair programming improves student retention, confidence, and program quality, Commun. ACM 49 (2006) 90–95. URL: https://dl.acm.org/doi/10.1145/1145287.1145293. doi:10.1145/1145287.1145293.

[20] P. Maguire, R. Maguire, P. Hyland, P. Marshall, Enhancing collaborative learning using pair programming: Who benefits?, AISHE-J 6 (2014). URL: https://ojs.aishe.org/index.php/aishe-j/article/view/141.

[21] M. Ally, F. Darroch, M. Toleman, A framework for understanding the factors influencing pair programming success, in: Extreme Programming and Agile Processes in Software Engineering, Lecture notes in computer science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 82–91. URL: http://link.springer.com/10.1007/11499053_10. doi:10.1007/11499053_10.

[22] E. A. Chaparro, A. Yuksel, P. Romero, S. Bryant, Factors affecting the perceived effectiveness of pair programming in higher education, Annual Workshop of the Psychology of Programming Interest Group (2005). URL: https://www.semanticscholar.org/paper/c095f0d9b17cd9c2851000534740e7cc087253fa.

[23] J. Chong, T. Hurlbutt, The social dynamics of pair programming, in: 29th International Conference on Software Engineering (ICSE'07), ieeexplore.ieee.org, 2007, pp. 354–363. URL: http://dx.doi.org/10.1109/ICSE.2007.87. doi:10.1109/ICSE.2007.87.

[24] S. Freudenberg, P. Romero, B. Du Boulay, Talking the talk: Is intermediate-level conversation the key to the pair programming success story?, in: AGILE 2007, unknown, 2007, pp. 84–91. URL: https://www.researchgate.net/publication/4270516_Talking_the_talk_Is_intermediate-level_conversation_the_key_to_the_pair_programming_success_story. doi:10.1109/AGILE.2007.1.

[25] L. Murphy, S. Fitzgerald, B. Hanks, R. McCauley, Pair debugging: a transactive discourse analysis, in: Proceedings of the Sixth international workshop on Computing education research, ICER '10, Association for Computing Machinery, New York, NY, USA, 2010, pp. 51–58. URL: https://doi.org/10.1145/1839594.1839604. doi:10.1145/1839594.1839604.

[26] A. Begel, N. Nagappan, Pair programming: what's in it for me?, in: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, ACM, New York, NY, USA, 2008. URL: https://dl.acm.org/doi/10.1145/1414004.1414026. doi:10.1145/1414004.1414026.

[27] D. Preston, Using collaborative learning research to enhance pair programming pedagogy, SIGITE Newsl. 3 (2006) 16–21. URL: https://doi.org/10.1145/1113378.1113381. doi:10.1145/1113378.1113381.

[28] W. Sun, G. Marakas, The true cost of pair programming: Development of a comprehensive model and test, Americas Conference on Information Systems (2009). URL: https://www.semanticscholar.org/paper/647fc48650e4f19962c8a6feb87f3bdedde9dd04.

[29] J. E. Hannay, T. Dybå, E. Arisholm, D. I. K. Sjøberg, The effectiveness of pair programming: A meta-analysis, Information and Software Technology 51 (2009) 1110–1122. URL: https://www.sciencedirect.com/science/article/pii/S0950584909000123. doi:10.1016/j.infsof.2009.02.001.

[30] B. Hanks, S. Fitzgerald, R. McCauley, L. Murphy, C. Zander, Pair programming in education: a literature review, Comput. Sci. Educ. 21 (2011) 135–173. URL: https://www.tandfonline.com/doi/full/10.1080/08993408.2011.579808. doi:10.1080/08993408.2011.579808.

[31] S. Barke, M. B. James, N. Polikarpova, Grounded copilot: How programmers interact with Code-Generating models (2022). URL: http://arxiv.org/abs/2206.15000. arXiv:2206.15000.

[32] Q. Ma, T. Wu, K. Koedinger, Is AI the better programming partner? Human-Human pair programming vs. Human-AI pAIr programming (2023). URL: http://arxiv.org/abs/2306.05153. arXiv:2306.05153.

[33] E. Arisholm, H. Gallis, T. Dyba, D. I. K. Sjoberg, Evaluating pair programming with respect to system complexity and programmer expertise, IEEE Trans. Software Eng. 33 (2007) 65–86. URL: http://dx.doi.org/10.1109/TSE.2007.17. doi:10.1109/TSE.2007.17.

[34] L. Plonka, J. Segal, H. Sharp, J. van der Linden, Collaboration in pair programming: Driving and switching, in: Agile Processes in Software Engineering and Extreme Programming - 12th International Conference, XP 2011, Madrid, Spain, May 10-13, 2011. Proceedings, volume 77, unknown, 2011, pp. 43–59. URL: https://www.researchgate.net/publication/221592723_Collaboration_in_Pair_Programming_Driving_and_Switching. doi:10.1007/978-3-642-20677-1_4.

[35] J. E. Hannay, E. Arisholm, H. Engvik, D. I. K. Sjoberg, Effects of personality on pair programming, IEEE Trans. Software Eng. 36 (2010) 61–80. URL: http://dx.doi.org/10.1109/TSE.2009.41. doi:10.1109/TSE.2009.41.

[36] R. W. Jensen, A pair programming experience, ACCU - professionalism in programming Overload 13 (2005). URL: https://accu.org/journals/overload/13/65/jensen_254/.

[37] K. M. Lui, K. C. C. Chan, Pair programming productivity: Novice–novice vs. expert–expert, Int. J. Hum. Comput. Stud. 64 (2006) 915–925. URL: https://linkinghub.elsevier.com/retrieve/pii/S1071581906000644. doi:10.1016/j.ijhcs.2006.04.010.

[38] A. Sillitti, G. Succi, J. Vlasenko, Understanding the impact of pair programming on

developers attention: A case study on a large industrial experimentation, in: 2012 34th International Conference on Software Engineering (ICSE), IEEE, 2012, pp. 1094–1101. URL: http://dx.doi.org/10.1109/ICSE.2012.6227110. doi:10.1109/ICSE.2012.6227110.

[39] H. Mozannar, G. Bansal, A. Fourney, E. Horvitz, Reading between the lines: Modeling user behavior and costs in AI-assisted programming, ArXiv (2022). URL: http://dx.doi.org/10.48550/ARXIV.2210.14306. doi:10.48550/ARXIV.2210.14306.

[40] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, E. A. Santos, Programming is hard - or at least it used to be: Educational opportunities and challenges of AI code generation, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023, Association for Computing Machinery, New York, NY, USA, 2023, pp. 500–506. URL: https://doi.org/10.1145/3545945.3569759. doi:10.1145/3545945.3569759.

[41] A. Sarkar, A. D. Gordon, C. Negreanu, C. Poelitz, S. S. Ragavan, B. Zorn, What is it like to program with artificial intelligence? (2022). URL: http://arxiv.org/abs/2208.06213. arXiv:2208.06213.

[42] A. Murillo, S. D'Angelo, An engineering perspective on writing assistants for productivity and creative code, The Second Workshop on Intelligent and Interactive Writing Assistants (2023). URL: https://cdn.glitch.global/d058c114-3406-43be-8a3c-d3afff35eda2/paper1_2023.pdf.

[43] H. H. Thorp, ChatGPT is fun, but not an author, Science 379 (2023) 313. URL: http://dx.doi.org/10.1126/science.adg7879. doi:10.1126/science.adg7879.

[44] Google, Bard, https://bard.google.com/, ???? URL: https://bard.google.com/, accessed: 2023-5-19.

[45] Github, GitHub copilot labs, https://githubnext.com/projects/copilot-labs/, ???? URL: https://githubnext.com/projects/copilot-labs/, accessed: 2023-5-19.

[46] M. Perscheid, B. Siegmund, M. Taeumel, R. Hirschfeld, Studying the advancement in debugging practice of professional software developers, Software Quality Journal 25 (2017) 83–110. URL: https://doi.org/10.1007/s11219-015-9294-2. doi:10.1007/s11219-015-9294-2.

[47] W. Ahrendt, R. Bubel, R. Hähnle, Integrated and Tool-Supported teaching of testing, debugging, and verification, in: Teaching Formal Methods, Springer Berlin Heidelberg, 2009, pp. 125–143. URL: http://dx.doi.org/10.1007/978-3-642-04912-5_9. doi:10.1007/978-3-642-04912-5_9.

[48] J. Smith, J. Tessler, E. Kramer, C. Lin, Using peer review to teach software testing, in: Proceedings of the ninth annual international conference on International computing education research, ICER '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 93–98. URL: https://doi.org/10.1145/2361276.2361295. doi:10.1145/2361276.2361295.

[49] R. McCauley, S. Fitzgerald, G. Lewandowski, L. Murphy, B. Simon, L. Thomas, C. Zander, Debugging: A review of the literature from an educational perspective, Computer Science Education 18 (2008) 67–92. URL: http://www.informaworld.com/openurl?genre=article&id=doi:10.1080/08993400802114581. doi:10.1080/08993400802114581.

[50] S. Fitzgerald, R. McCauley, B. Hanks, L. Murphy, B. Simon, C. Zander, Debugging from the student perspective, IEEE Trans. Educ. 53 (2010) 390–396. URL: http://dx.doi.org/10.1109/TE.2009.2025266. doi:10.1109/TE.2009.2025266.

[51] K. Holstein, V. Aleven, N. Rummel, A conceptual framework for Human–AI hybrid adaptivity in education, Artificial Intelligence in Education 12163 (2020) 240. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7334162/. doi:10.1007/978-3-030-52237-7_20.

[52] P. Robe, S. K. Kuttal, Designing PairBuddy—A conversational agent for pair programming, ACM Trans. Comput.-Hum. Interact. 29 (2022) 1–44. URL: https://doi.org/10.1145/3498326. doi:10.1145/3498326.

[53] K.-W. Han, E. Lee, Y. Lee, The impact of a Peer-Learning agent based on pair programming in a programming course, IEEE Trans. Educ. 53 (2010) 318–327. URL: http://dx.doi.org/10.1109/TE.2009.2019121. doi:10.1109/TE.2009.2019121.

[54] S. K. Kuttal, B. Ong, K. Kwasny, P. Robe, Trade-offs for substituting a human with an agent in a pair programming context: The good, the bad, and the ugly, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, number Article 243 in CHI '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1–20. URL: https://doi.org/10.1145/3411764.3445659. doi:10.1145/3411764.3445659.

[55] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, A. Hellas, Comparing code explanations created by students and large language models (2023). URL: http://arxiv.org/abs/2304.03938. arXiv:2304.03938.

[56] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, E. A. Santos, "it's weird that it knows what I want": Usability and interactions with copilot for novice programmers (2023). URL: http://arxiv.org/abs/2304.02491. arXiv:2304.02491.

[57] B. Puryear, G. Sprint, Github copilot in the classroom: learning to code with AI assistance, J. Comput. Sci. Coll. 38 (2022) 37–47. URL: https://dl.acm.org/doi/pdf/10.5555/3575618.3575622.

[58] D. Mhlanga, Open AI in education, the responsible and ethical use of ChatGPT towards lifelong learning, 2023. URL: https://papers.ssrn.com/abstract=4354422. doi:10.2139/ssrn.4354422.

[59] T. Y. Zhuo, Y. Huang, C. Chen, Z. Xing, Red teaming ChatGPT via jailbreaking: Bias, robustness, reliability and toxicity (2023). URL: https://www.researchgate.net/publication/368476294. doi:10.48550/ARXIV.2301.12867. arXiv:2301.12867.