AI-assisted Mob Programming in Higher Education: A Study of Team Dynamics and AI Usage Patterns

Luka Četina[1,*], Tilen Hliš[1] and Marjan Heričko[1]

[1]Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška cesta 46, Maribor, 2000, Slovenia

Abstract

Collaborative group work is common in programming courses, and it often follows a recognizable pattern: one student types while the others observe, suggest next steps, and catch errors. With AI coding assistants being widely used by students, we wanted to understand how they integrate into this collaborative dynamic. We report on a pilot study conducted in a graduate-level course at the University of Maribor, where 13 student teams used AI tools across four mob programming lab sessions. Through team-level and individual surveys, we tracked how students interacted with AI, what roles they took on, and how their practices changed over time. When deciding between writing code themselves with AI support and generating code with AI before reviewing it, students generally preferred the latter. Teams typically organised around an "AI operator" who managed prompts while the others reviewed and evaluated the generated output, shifting non-drivers from passive observation to active code review. Across four sessions, completion rates increased while prompt counts decreased, which could suggest students learned to use AI more efficiently.

Keywords

Mob programming, AI-assisted development, software development education

1. Introduction

Collaborative work is common in programming courses, particularly in practical lab sessions [3, 8]. In these settings, we repeatedly observe a familiar pattern: one student types while the others lean in, point at the screen, suggest next steps, and catch mistakes. This group dynamic is called mob programming and is defined as a software practice where the software development team all work together on the same computer on a similar issue by alternating the coder [9]. Having observed this dynamic repeatedly in our own courses, we became curious about what happens when AI coding tools enter the picture. If one student is already writing the code while others advise, how does an AI assistant change the group dynamic?

AI tools like GitHub Copilot, ChatGPT, and Cursor have become a standard part of many students' development workflows. Peng et al. [7] report a 55.8% speedup in task completion when developers used Copilot, and these tools are spreading rapidly through both industry and education. But most empirical research on AI-assisted development examines individual developers working alone [1, 10, 2]. What happens when the same tools are used by a team sharing a single screen is less well explored.

We set up an exploratory study to observe this. In a graduate-level course at the University of Maribor, 13 student teams (3 members each) worked through four weekly lab sessions in a mob programming format. They used one shared computer, were required to use AI tools and received specific development tasks each week. Through surveys after each session, we collected data on the AI usage patterns, the roles they took on, and general team dynamics.

Our paper is positioned as a pilot study exploring AI-assisted mob programming in higher education. Due to a small sample and self-reported data, our reports are preliminary observations about the dynamics that appeared in our particular setting.

We explored three questions:

NWISEd 2026: Workshop on Co-Creating New Ways of Information Systems Education, July 1–3, 2026, Swansea, United Kingdom

*Corresponding author.

luka.cetina@um.si (L. Četina); tilen.hlis@um.si (T. Hliš); marjan.hericko@um.si (M. Heričko)

0009-0000-0267-1092 (L. Četina); 0000-0002-4973-4844 (T. Hliš); 0000-0002-1094-0085 (M. Heričko)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-19

CEUR-WS.org/Vol-4333/paper5.pdf

- **RQ1:** How do students use AI programming tools during mob programming sessions?
- **RQ2:** What team roles and dynamics emerge when students use AI tools during mob programming sessions?
- **RQ3:** How do AI usage patterns and team dynamics change over multiple mob programming sessions?

The rest of the paper is structured as follows: we first review related work, then describe the study design, present the results, discuss the findings and their implications, outline the limitations, and conclude.

2. Related Work

2.1. Pair and Mob Programming in Education

Pair programming has been studied in educational settings since the early 2000s. Williams et al. [11] showed that student pairs produced better code in less overall time than students working alone. Hannay et al. [4] later conducted a meta-analysis and confirmed a small positive effect on quality and a moderate positive effect on duration, though at the cost of roughly 84% more total person-hours. Nosek [6] found that collaborative teams outperformed individuals on complex tasks, particularly those requiring creative problem solving.

Mob programming extends the pair programming idea to the whole team: everyone works on one task at one computer, rotating who sits at the keyboard. It grew out of Agile and Extreme Programming communities, but unlike pair programming, it has seen very little empirical study. Shiraishi et al. [9] conducted a systematic literature review and found a handful of studies, none in educational contexts. Most classroom research on collaborative coding is focused on pair programming or general Agile practices [3, 8].

2.2. AI-Assisted Software Development

How do developers actually interact with AI code generation tools? Barke et al. [1] ran a grounded theory study of Copilot users and identified two distinct modes. In *acceleration mode*, the developer already knows what they want and uses AI to produce it faster. In *exploration mode*, the developer is less sure and uses AI to try out options. This distinction turned out to be useful for interpreting our own data (see Section 5).

Not all evidence on AI assistance is positive. Vaithilingam et al. [10] found that Copilot did not actually improve task completion time in their study, even though participants preferred using it. The main issue was comprehension: developers struggled to understand and debug AI-generated code. Dakhel et al. [2] showed that Copilot can be a liability for novices, because less experienced developers lacked the judgement to evaluate whether a suggestion was correct.

2.3. AI in Software Engineering Education

AI tools are spreading rapidly through computer science courses. Kasneci et al. [5] reviewed LLM applications across education and found growing adoption, but most studies examined individual use. AI-assisted software development has become widespread and preventing or controlling how students use AI-tools for course assignments is often not feasible. Therefore we wanted to focus on understanding what collaborative work looks like when using AI, to better support the students and make sure they achieve the required learning outcomes. Our study does not fill this gap comprehensively, but it offers a first set of observations from our particular setting.

3. Study Design

3.1. Course Context

The study took place as part of "Development of Information Solutions and Services", which is a graduate-level course in a Master's programme at the University of Maribor. It covers software requirements, design, implementation, testing, and CI/CD automation, and runs for 15 weeks with one 90-minute session each week.

3.2. Participants

Students were organised into 13 teams of 3 members. Participants were in their 2nd-5th year of study. Among those who reported on work experience ($N = 31$), the large majority had at least some professional IT experience through internships or part-time work; only two students reported none.

A baseline survey ($N = 39$) showed that prior AI experience was widespread: 57% rated their AI development skill as "some," 21% as "a lot," and 19% as "proficient." Only one student reported "very little" experience. Self-rated teamwork ability averaged 4.13 out of 5.

3.3. Procedure

The study ran over four sessions from October to November 2025, one per week. Each session consisted of approximately 90 minutes of active mob programming in the lab, followed immediately by a survey.

The instructions to students were straightforward: complete the assigned development tasks, use an AI coding tool, and work together at one shared computer. Each team received a different but structurally comparable assignment for a small web information system. The assignment description contained the application idea, prioritised user stories with acceptance criteria and wireframes. All assignments required the users to implement user registration/login, searching/filtering, detail views, CRUD operations, notifications as well as administrator views. What we did not prescribe was how to use AI. Students could write code manually and consult AI for suggestions, generate entire blocks of code through prompts, or anything in between. They could also chose their AI tools freely and use multiple ones in parallel.

When teams first received the assignments, they broke them down into four approximatelly equal increments, each corresponding to about 25% of the overall required functionallity. Each week, they were expected to implement one 25% increment and were told to stop working once they had finished the tasks assigned for that session.

3.4. Data Collection

We collected data through three online surveys administered after each session:

1. A **baseline survey** ($N = 39$, before the first session): programming experience, AI experience/skill self-assessment and typical AI coding strategies.
2. A **team survey** (13 per session, 52 total): one response per group, covering coding time, keyboard rotations and their duration, number of AI prompts used, self-reported task completion percentage, and a free-text description of how the team organised its work adn which AI tools they used. This survey also included four items about coding practices: how often the team accepted shortcuts (technical debt), fixed AI-generated errors, refactored code, and wrote tests.
3. An **individual survey** (36-38 per session, 149 total): each team member reported how frequently they occupied four roles during the session (AI operator, advisor, manual coder, idle), how frequently they used each of five AI coding strategies (manual with AI advice, manual with AI completion, mixed, mostly AI, full AI generation), and how they rated the development process, AI effectiveness, and teamwork. Some students did not complete the individual survey in every session therefore the total number is sligthly lower than expected.

All frequency items used a 5-point ordinal scale: Never (0), Rarely (1), Sometimes (2), Often (3), Always (4). Satisfaction items used a scale from Poor to Very Good. Because Poor and Very Poor responses were absent in the satisfaction ratings, we report means on a 1-3 scale where OK = 1, Good = 2, and Very Good = 3.

Table 1 summarises the constructs.

Table 1
Constructs measured across the study's data collection instruments.

Construct	Measurement	Instrument	Scope
Programming experience	Years	Baseline survey	Individual
Typical AI workflow	Single-choice (4 styles)	Baseline survey	Individual
AI experience/skill	Ordinal self-rating (1-5)	Baseline survey	Individual
Task completion	Self-reported % of plan	Post session survey	Team
AI prompt count	Self-reported count	Post session survey	Team
Coding practices	4 items, 0-4 freq.	Post session survey	Team
Keyboard rotation	Count and avg. duration (min)	Post session survey	Team
AI tool used	Free-text	Post session survey	Team
AI coding strategy	5 approaches, 0-4 freq.	Post session survey	Individual
Team role frequency	4 roles, 0-4 freq.	Post session survey	Individual
Satisfaction	3 items, 1-3 scale	Post session survey	Individual

4. Results

4.1. AI Usage Patterns (RQ1)

Students were required to use AI, so the finding that they did is not surprising. What is worth reporting is how they chose to use it, since the manner of use was entirely their choice. We asked each participant to rate how often they used five different approaches during a session:

1. **Full AI generation**-prompting the AI to produce entire code blocks, used with minimal manual editing.
2. **Mostly AI**-generating code through AI, then modifying it.
3. **Mixed**-combining AI-generaion and manual coding in varying proportions.
4. **Manual with AI completion**-writing code manually, using AI autocomplete.
5. **Manual with AI advice**-writing code manually, consulting AI only for tips or explanations.

Note that these are not mutually exclusive: a student could report using several approaches during the same session.

Full AI generation scored highest (mean 2.77), followed by mostly-AI (2.16). The three manual-oriented approaches all scored below 1.2 (mixed: 1.18, manual with AI completion: 0.92, manual with AI advice: 0.72). Figure 1 shows the distribution.

The most common AI tools used were ChatGPT, GitHub Copilot, and Cursor, though Claude, DeepSeek, and several others also appeared. Multiple teams used two or more tools in parallel during a single session (Table 2).

The pattern was consistent across all four sessions. Students mostly used a "generate-and-review" workflow where they prompt the AI to produce a block of code, check whether it works, and move on.

The team survey also included four items about coding practices, each measured on the 0-4 frequency scale. Testing was essentially absent (mean 0.12, between "never" and "rarely"). Refactoring was infrequent (0.83). Teams reported accepting shortcuts (technical debt) at a moderate level (1.52) and fixing AI-generated errors somewhat more frequently (1.79).

Table 2
AI assistants used by teams across sessions (team-level counts, $N = 13$ teams per session). Counts can exceed N because teams could use more than one tool in a session.

AI assistant	S1	S2	S3	S4
GitHub Copilot	6	7	5	6
Cursor	3	2	3	1
ChatGPT	9	5	6	8
Warp	0	1	0	0
Hercules	0	0	1	0
DeepSeek	0	0	1	1
Claude	0	0	0	1

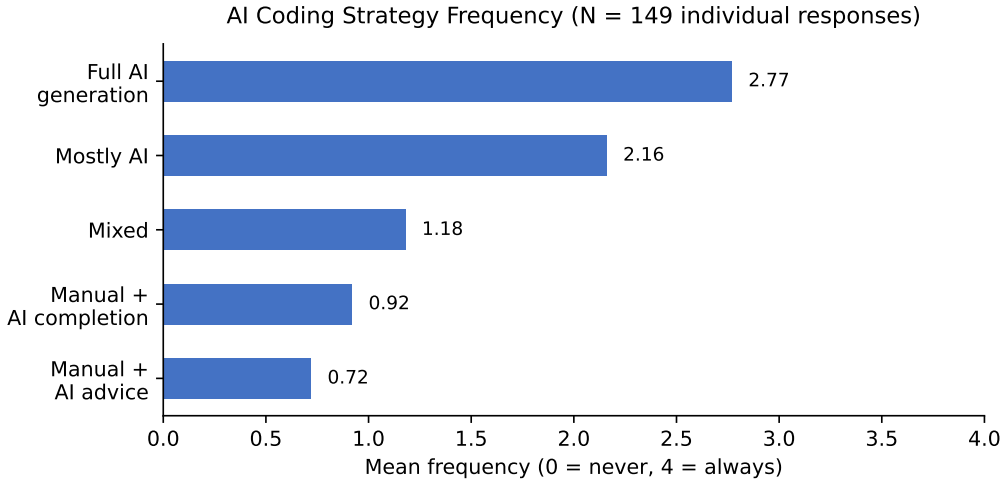


Figure 1: Frequency of AI coding strategies ($N = 149$ individual responses across all sessions). Scale: 0 = never, 4 = always.

4.2. Team Roles and Dynamics (RQ2)

In traditional mob programming, the driver types code and the navigators observe and advise. We were curious whether AI tools would change this dynamic. We asked each team member to rate how frequently they occupied four roles during a session: *AI operator* (writing and refining prompts), *advisor* (commenting on code, spotting errors), *manual coder* (writing code by hand), and *idle* (waiting, not actively contributing).

As Figure 2 shows, the AI operator role scored highest (mean 2.66 across all sessions and individuals), with advisor close behind at 2.43. Manual coding was less frequent (1.56), and idle time was low (1.16). The free-text responses from the team survey flesh out these numbers. In the first session, one team wrote (translated from Slovene): "*The person at the computer gave instructions to the AI, the other two advised him and looked for potential errors*". Another described their workflow: "*One person set the prompts, the others watched and gave comments, together we looked for solutions*". By the fourth session, teams described similar structures in shorter, more matter-of-fact terms: "*One typed, the other two advised*". Keyboard rotation averaged 2.6 swaps per session. The average turn length grew from 20.0 minutes in session 1 to 32.7 minutes by session 4-teams rotated somewhat less frequently over time, but each turn lasted longer.

We also looked at whether the AI operator role was held by one person or rotated. For each team we compared individual members' mean AI operator scores across sessions. In 8 of 13 teams the gap between the highest and lowest scorer was under 1.5 points, meaning all members engaged with the role at roughly similar levels. In the other five teams one person scored noticeably higher than the rest, suggesting someone informally took the lead. Even in those teams the lower-scoring members still

averaged above 1.5, so nobody was completely absent from the role. By session 4, every member in every team rated their AI operator involvement at 2 or higher.

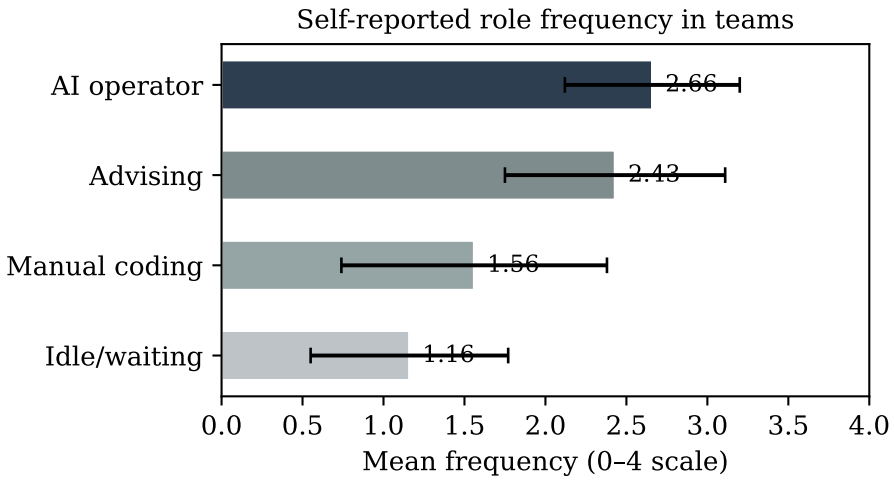


Figure 2: Self-reported role frequency across all team members and sessions ($N = 149$). Error bars show standard deviation.

4.3. Changes Across Sessions (RQ3)

We also looked at how the metrics changed over the four sessions. Table 3 and Figure 3 show the key metrics.

Table 3
Key metrics per session (team-level data, $N = 13$ per session).

Metric	S1	S2	S3	S4
Avg. num. of keyboard rotations	2.3	2.9	2.2	3.2
Completion rate (%)	55.5	62.1	66.6	71.5
AI prompts per session	23.0	20.8	11.7	13.9
Rotation duration (min)	20.0	24.6	29.6	32.7
Satisfaction: process (1-3)	2.03	2.05	2.26	2.28
Satisfaction: AI tools (1-3)	1.97	2.03	2.27	2.28
Satisfaction: teamwork (1-3)	2.46	2.44	2.53	2.62
Teams slower than planned	5/13	3/13	4/13	3/13

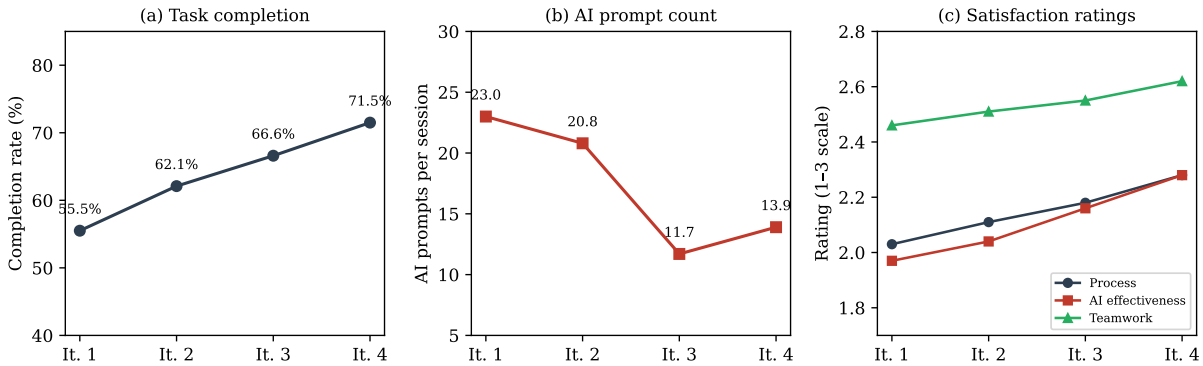


Figure 3: Trends across four sessions: (a) self-reported task completion, (b) AI prompt count per session, (c) satisfaction ratings (1-3 scale).

Self-reported task completion rose by 16 percentage points (55.5% to 71.5%). The number of AI prompts dropped from 23.0 in session 1 to 11.7 in session 3, then rose slightly to 13.9 in session 4. The overall pattern shows more completed with fewer prompts over time, which suggests that teams became more efficient at interacting with AI. We cannot rule out that the later portions of the project happened to be easier to implement, but since the scope was planned to be evenly distributed, we consider a learning effect plausible.

Satisfaction generally improved, with the largest gain in perceived AI effectiveness (+0.31 points on a 1-3 scale from session 1 to session 4). The line was not perfectly straight as teamwork satisfaction dipped slightly in session 2 before rising in sessions 3 and 4. But the overall direction was positive. Time estimation also improved: in session 1, 5 of 13 teams finished slower than planned whereas by session 4, only 3 did.

5. Discussion

5.1. RQ1: AI Usage Patterns

RQ1 explores how the students used AI to finish the given task. Figure 1 shows a clear hierarchy of strategies: full AI generation had the highest mean frequency (2.77), followed by mostly AI (2.16), while the more manual strategies remained below 1.2. This indicates that teams treated AI primarily as a code-generation engine and only secondarily as an assistant for completion or advice.

This pattern is consistent with Barke et al.'s distinction between exploration and acceleration modes [1]. In our setting, many interactions appear closer to acceleration as students delegated he initial implementation to AI, then reviewed and edited. At the same time, non-zero scores for mixed and manual-with-completion suggest that teams did not rely blindly on generation and still switched to manual coding when outputs were unclear, unsuitable, or needed refinement.

The preference for generate-and-review is also plausible in a 90-minute lab context with completion-oriented tasks. Prior work notes that AI output still requires comprehension and correction [10, 2]. Our data suggest students addressed this by combining rapid generation with ongoing peer review rather than by abandoning manual work entirely.

5.2. RQ2: Team Roles and Dynamics

The pattern we found most interesting for RQ2 was how the team dynamic restructured itself around the AI tool.

In traditional mob programming, the person at the keyboard is the *code writer*-they translate the group's ideas into actual code, character by character. The navigators watch this process unfold in real time, catching typos and suggesting improvements as the code takes shape. This is inherently a slow, incremental process, and one of the known challenges of mob programming is keeping the non-drivers engaged while someone else types [9].

In our setting, the person at the keyboard became a prompt operator. Instead of writing code line by line, they mostly prompted the AI to produce code which everyone (driver and navigators) read and evaluated simultaneously. The nature of the non-drivers' contribution changed as instead of watching code being typed and catching errors in real time, they were reviewing finished blocks and asking whether the output matched what the team intended.

This is a meaningful shift as it changes the cognitive load distribution within the team. The prompt operator needs to translate the group's intent into a clear request for the AI, which is often more akin to writing specifications than programming. The navigators then need to assess whether a block of generated code is correct and complete which is very similar to non-AI mob programming except that here, code is reviewed in chunks as it is generated and not written line by line.

The low idle time we observed (1.16 on a 0-4 scale) is consistent with this interpretation: reviewing a block of generated code is a more concrete and engaging task than watching someone type. But we note that idle time is self-reported, and students may have been reluctant to admit inactivity. The qualitative

responses, however, independently describe active reviewing, which gives us some confidence that the pattern was real.

Another notable point is that the AI-operator role was shared rather than dominated by one student. With keyboard rotation averaging 2.6 swaps per session, prompt writing naturally moved from one team member to another. This likely spread prompt-engineering practice across the whole team and may have contributed to the improvement we observed over time.

5.3. RQ3: Changes Across Sessions

The prompt count data-dropping from 23.0 per session to 11.7 in session 3 while completion rates rose-suggests that teams became more efficient at using AI over time. This connects to Barke et al.'s [1] framework of exploration versus acceleration modes. In early sessions, teams may have been *exploring*: trying different prompts, seeing what works, discarding failed attempts. By session 3, they appear to have shifted toward *acceleration*: knowing what to ask for, getting it right with fewer attempts.

What makes this potentially interesting from an educational perspective is that this learning was collective. The whole team was present for every prompt and every response. When one person's prompt failed and another's approach succeeded, the entire team saw why. This shared experience may accelerate the learning process compared to individual trial and error, though we have no data to confirm this directly.

The growing rotation duration (from 20 to 33 minutes) could reflect growing trust: teams let each driver work longer because the group had developed shared expectations about prompting and reviewing.

5.4. Implications for Educators

The mob programming format, combined with AI tools, did not lead to the disengagement we might have feared. Teamwork was consistently the highest-rated satisfaction dimension (mean 2.51 on a 1-3 scale), and both completion rates and satisfaction improved across sessions. This suggests the format is at least a viable option in higher education programming courses.

The emergence of the prompt operator role raises a practical question for instructors: should keyboard rotation be enforced more strictly to ensure all students practise interacting with AI? Our data shows 2.6 rotations per session on average, which means each student in a 3-person team got roughly one turn per session. Whether this is enough practice is an open question.

The absence of testing and refactoring, while unsurprising given that neither was required, does highlight that AI-augmented mob programming is unlikely to produce quality-conscious habits on its own. If quality practices matter for the learning objectives, they need to be explicitly built into the task requirements.

6. Limitations

All data is self-reported, as we did not record screens, log prompts, or run automated code analysis. There was no control group that worked without AI, so we cannot isolate AI as a causal factor. This was largely due to course constraints, as assigning one part of the class to a non-AI condition could have created different requirements and difficulty across teams. It would also have been difficult to reliably enforce a strict no-AI condition, as students were likely to try to use AI tools anyway. Although teams were instructed to stop once they finished the tasks assigned for each session, we cannot verify that all teams worked strictly within the intended per-session scope; some may have completed more or less than planned. The sample is small (39 students, 13 teams) and comes from a single institution and course.

7. Conclusion

We observed 13 student teams using AI tools in a mob programming format over four sessions. The study was small and all data is self-reported, so what we offer are preliminary observations rather than generalisable conclusions.

The most notable pattern was a restructuring of team roles around the AI tool: the traditional mob programming driver became a prompt operator, and the navigators became reviewers of generated code blocks. This appeared to keep non-drivers more engaged than we had expected. Teams also appeared to become more efficient at prompting over time, completing more while querying the AI less.

Future work should incorporate more objective measures, larger samples, and comparisons with other collaborative formats to test whether these findings are generaliseable.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly for grammar, readability, and spelling checks and Prism to help style and fix LaTeX errors. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of this publication.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research and Innovation Agency (Research Core Funding No. P2-0057).

References

[1] Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7 (OOPSLA1):85:1–85:27, 2023. doi: 10.1145/3586030.

[2] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming Jiang. GitHub Copilot AI pair programmer: Asset or liability? *Journal of Systems and Software*, 203:111734, 2023. doi: 10.1016/j.jss.2023.111734.

[3] Vladan Devedžić and Saša R. Milenković. Teaching agile software development: A case study. *IEEE Transactions on Education*, 54(2):273–278, 2011. doi: 10.1109/TE.2010.2052104.

[4] Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag I. K. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7):1110–1122, 2009. doi: 10.1016/j.infsof.2009.02.001.

[5] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Saber, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Joana Weidlich, Jochen Kuhn, and Gjergji Kasneci. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023. doi: 10.1016/j.lindif.2023.102274.

[6] John T. Nosek. The case for collaborative programming. *Communications of the ACM*, 41(3): 105–108, 1998. doi: 10.1145/272287.272333.

[7] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590*, 2023.

[8] Andreas Schroeder, Annabelle Klarl, Philip Mayer, and Christian Kroiß. Teaching agile software development through lab courses. In *Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON)*, 2012. doi: 10.1109/EDUCON.2012.6201194.

[9] Makoto Shiraishi, Hironori Washizaki, Yoshiaki Fukazawa, and Joseph Yoder. Mob programming: A systematic literature review. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 616–621, 2019. doi: 10.1109/COMPSAC.2019.10276.

[10] Priyan Vaithilingam, Tianyi Zhang, and Elena 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*, CHI EA '22, 2022. doi: 10.1145/3491101.3519665.

[11] Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. Strengthening the case for pair programming. *IEEE Software*, 17(4):19–25, 2000. doi: 10.1109/52.854064.