Structuring Collaboration in Programming Through Personal-Spaces

Devamardeep Hayatpur
University of California, San Diego
La Jolla, California, USA
dshayatpur@ucsd.edu

Tehilla Helfenbaum
University of Toronto
Toronto, Ontario, Canada
t.helfenbaum@mail.utoronto.ca

Haijun Xia
University of California, San Diego
La Jolla, California, USA
haijunxia@ucsd.edu

Wolfgang Stuerzlinger
Simon Fraser University
Vancouver, British Columbia, Canada
w.s@sfu.ca

Paul Gries
University of Toronto
Toronto, Ontario, Canada
pgries@cs.toronto.edu

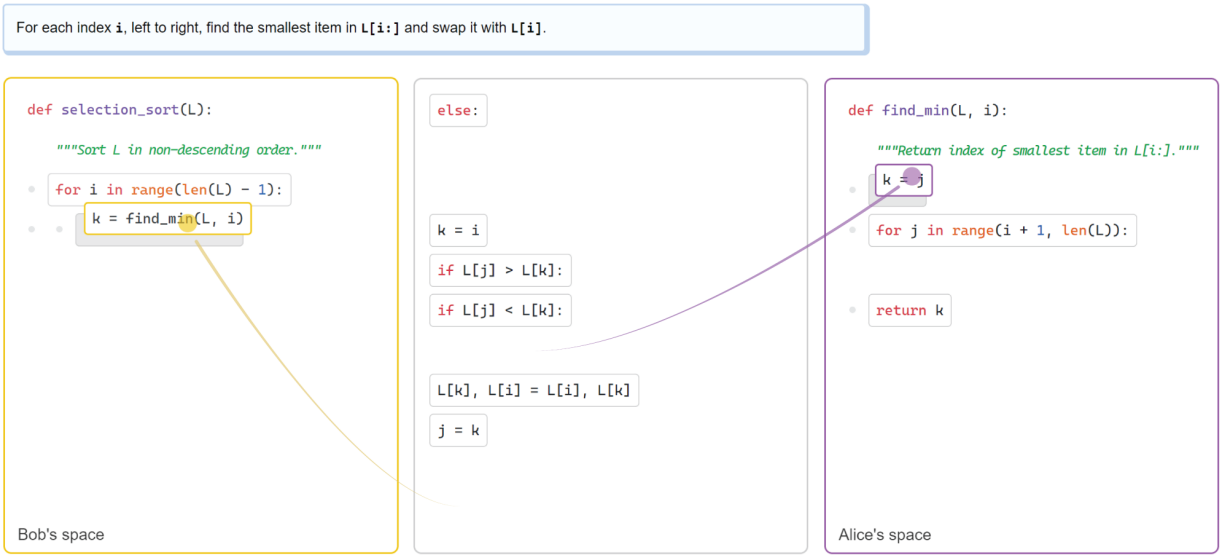

Figure 1: Screenshot of our tool, annotated with cursor movements. We split a Parsons problem into two sub-spaces, assigning one to each student. They must share fragments from the middle space to complete their assigned part.

ABSTRACT

The effectiveness of pair programming in pedagogy depends on the frequency and quality of communication of the driver. We explore an alternative collaboration paradigm that tackles this imbalance through Parsons problems: students are given fragments of code out of order and tasked with re-organizing them into the correct order. We then create an interdependence between students by assigning each to a different sub-problem in their own space, termed Personal-spaces – they must engage in dialog to negotiate, exchange, and share fragments. In an exploratory study with nine pairs of undergraduate students, we find evidence pointing to affordances of different coordination conditions: Personal-spaces promoted ownership and engagement, while Turn-taking (akin to pair programming) helped maintain a consistent train of thought. Our results provide considerations for design of appropriate problem sets and interfaces to structure collaborative learning.

CCS CONCEPTS

• **Human-centered computing** → *Synchronous editors*; • **Social and professional topics** → **Computer science education**.

KEYWORDS

collaborative learning, pair programming, Parsons problems

ACM Reference Format:
Devamardeep Hayatpur, Tehilla Helfenbaum, Haijun Xia, Wolfgang Stuerzlinger, and Paul Gries. 2023. Structuring Collaboration in Programming Through Personal-Spaces . In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems (CHI EA '23), April 23–28, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3544549.3585630

CHI EA '23, April 23–28, 2023, Hamburg, Germany

ACM ISBN 978-1-4503-9422-2/23/04.
https://doi.org/10.1145/3544549.3585630

1 INTRODUCTION

Collaborative learning has shown to be an effective and widely adopted technique in computer science curricula for fostering learning and critical thinking skills [14, 27]. One form of collaborative learning is pair programming, which designates one student to be the driver, who writes code, and the other student to be the navigator, who provides feedback and guidance to the driver. Many studies of pair programming provide evidence for improved student performance [20, 23], as well as long-term academic success [28].

However, pair programming is inherently asymmetric: the driver can iterate on a solution independent of the navigator. Rodríguez et al. found that the learning outcome of pair programming depends on the frequency and quality of communication by the driver [26]. Episodes in which the driver scarcely communicated have a worse learning outcome for the pair. Meanwhile, effective collaboration depends on shared ownership over the problem, which pair programming exercises lack de facto [27].

In this work, we describe the design and evaluation of a novel collaboration paradigm that aims to support shared ownership. Our tool uses Parsons problems as the programming environment, i.e., students re-arrange fragments of code to reach the correct arrangement. We divide a Parsons problem into two dependent subtasks, each of which are completed in their own space, termed a *Personal-space* (Figure 1). Each Personal-space is assigned to a single student, where their partner can view but not modify that space. The two students must then share fragments of code from the same pool to complete their individual problem in their Personal-space and reach an overall solution.

To explore student perceptions of Personal-spaces, we compared it to two other coordination conditions: (1) Shared-control, where both students can modify both spaces; and (2) Turn-taking, where only one of the students is able to modify both spaces at a time, but participants can swap control to alternate the driver and navigator roles, akin to pair programming. We find preliminary evidence of unique affordances of each coordination condition. For example, Turn-taking helped students converge towards a single, shared solution. Meanwhile, Personal-spaces encouraged collaborative dialog to negotiate for blocks and resolve conflicts, and led to a sense of responsibility and ownership over their space. In sum, we contribute (a) the design of a novel collaborative Parsons problem interface, (b) an exploratory study shedding light on the perceived strengths and limitations of these coordination conditions, and (c) design considerations for equitable collaborative interfaces.

2 RELATED WORK

We draw on prior work in cognitive accounts of collaboration and computer science education.

2.1 Collaborative Learning in Computer Science

Collaborative learning occurs when two or more students work together towards the same goal. It encourages students to externalize their mental models, develop shared knowledge, and consider multiple perspectives which all help repair misunderstandings and gaps in their own knowledge [3, 6, 30, 35]. Pair programming has shown to be a successful strategy for conducting co-located synchronous collaborative programming exercises [19, 20, 34]. It promotes engagement [21], self-efficacy [24], increases persistence [2], and reduces the gender gap in computer science courses [13, 33]. Pairs are generally more likely to produce correct code and more likely to succeed in the course [21]. The pedagogical benefits of paired programming have shown to extend to remote environments as well, where it is termed distributed pair programming [4, 11, 29].

The effectiveness of pair programming has shown to depend on the synergy and social dynamics between the two students. In a remote pair programming study, Rodríguez et al. found evidence that effective collaboration, where the pairwise learning outcome is higher, positively correlated with frequent communication from the driver [26]. Although on average pair programming reduces the gender gap between male and female computer science students, Kuttal et al. identified that mixed-gender pairs tend not to be democratic, with one partner dictating the collaboration [16]. Lewis et al. found that regardless of providing explicit instructions designed to promote equity, some students focused on task completion rather than group learning — marginalizing their partner's learning [17].

2.2 Positive Interdependence

The limitations of pair programming in pedagogy stem from its asymmetric interdependencies. The navigator is strongly dependent on the driver to manipulate the code (and for the overall success of the problem), but the driver has a weak dependency on the navigator. A driver can control and dictate the session without communicating or cooperating with the navigator [26]. Pair programming sessions often encourage or make students swap roles frequently, so both members practice being in a position of less dependency, and both have an opportunity to learn. However, frequent switches can be undesirable in shorter sessions. Best practices require a facilitator, like a teaching assistant, to supervise learning [34], which is not practical in many circumstances (e.g. in large classes).

To study effective collaboration in pedagogy, we adopt the underpinnings of positive interdependence, i.e., the belief that *"there is value in working together with anyone in the group"* and both the outcome and the learning will be better if done collaboratively [10, 27]. Positive interdependence is usually either reward-based, i.e., a shared grade, or task-based, i.e., assigned roles or resources [15, 27]. At the same time, an overly engineered interdependency reduces student autonomy and intrinsic motivation, as well as dictates their social interaction [7]. Each member should have enough self-autonomy to feel that they are responsible for completing their own work and contributed to the outcome of the group [1].

2.3 Parsons Problems

We will use Parsons problems: broken-up fragments of code that must be rearranged into the correct order as the basis for our tool [8]. Parsons problems have seen use as a successful learning and evaluation tool [9, 25]. Student success in solving Parsons problems has shown to correlate with success in writing code [9]. Parsons problems lower extrinsic cognitive load, allowing students to work constructively towards problems without needing to deal with the exact semantics of syntax. Common variations of parsons problems include specifying indentation, termed two-dimensional Parsons problems [12], and including distractors, which are fragments that are not part of the solution.

3 COLLABORATIVE PARSONS PROBLEMS

Our thesis is that collaborative programming interfaces can be designed with more equal and organic (i.e. avoid over-structuring their social interactions) learner dependency. To this end, we inform our design through two key guidelines following principles of positive interdependence:

(1) Both students must equally communicate and depend on one another to reach a solution.
(2) The interdependence mechanic must have enough 'room to maneuver' for agency and self-expression.

3.1 Design Outcome

Based on these goals, we designed Personal-spaces, which split a two-dimensional Parsons problem into two dependent sub-problems and assign each sub-problem to one learner. Our design can be characterized by two main attributes:

3.1.1 A personal and a shared space. The solutions to the two sub-problems are each assembled in their own Personal-space, giving each learner ownership and responsibility over a part of the solution. Initially, a shared space in the middle is populated with fragments of code. We pre-populate each individual space with documentation to describe its purpose, but ensure that the personal spaces are otherwise empty at the start. We hypothesize that learners will have the autonomy and opportunity to engage with the problem in their personal space, and use the shared space to resolve misunderstandings, communicate actions, and reach common ground. In addition, each person's cursor is represented as a colored dot which is synchronized with the color of the space they are responsible for.

3.1.2 A shared pool of resources. Our key insight is that since both learners must grab fragments from the same pool, they will need to share fragments with each other, in turn stimulating dialog and collaboration. And, since the two sub-problems depend on each other, the learners are incentivized to communicate what the code in their space does to reach a correct solution. Note that Parsons problems themselves have interesting implications to collaborative scenarios (versus writing code from scratch together). They provide a smaller 'space of actions' between the two learners, which may help establish common ground faster than typical synchronous code-writing tasks.

3.2 Problem Set Design

To evaluate our approach, we produced a problem set containing three introductory sorting algorithms: Bubble sort, Insertion sort, and Selection sort. Anecdotally, we find these algorithms to be approachable by students in introductory programming classes yet still non-trivial for experienced programmers. Since each sorting algorithm has two loop-invariants, they can be split into a parent and a helper function, each of which maintain the corresponding invariant. We also add distractors containing common error types: i.e., code with off-by-one errors in loop range or in list index, reversed assignments, reversed swaps, see, e.g., Figure 1. In early pilot studies, we identified that without the ambiguity introduced by the distractors, some participants trivially solved the problems solely based on the syntax.

4 EXPLORATORY STUDY

To explore the effects of introducing a resource interdependence to collaboration, we compare Personal-spaces against two other coordination conditions: Turn-taking and Shared-control in a within-subjects study.

In Turn-taking, only one person can move the code fragments, while the other person observes and provides feedback. To avoid any connotations associated with terms 'driver' and 'navigator', we adopted Bigman et al.'s terminology of pilot (who can edit the code) and co-pilot for the study [4]. To provide users with the option of swapping control, either person can press a button to reverse the roles at any time in this condition. In addition to our verbal instructions, the interface also briefly describes each persons role, e.g., *"As the Co-pilot, you should help guide the Pilot. Ask clarifying questions and help the Pilot think through the code."* In the Shared-control condition, both participants can use both spaces.

While both Turn-taking and Shared-control techniques are novel in the context of Parsons Problems, their general coordination strategies serve as proxies for existing collaborative techniques: Turn-taking is a proxy for pair programming, and Shared-control is a proxy for synchronous programming.

4.1 Participants

We recruited undergraduate students across two well-known North American universities that had completed or were taking an introductory computer science course (N=18; 6 male; 11 female; 1 prefer not to say; ages 18 to 23). Participants were paired solely based on their availability. Participant pairs are referred to as P1–P9. We refer to individual participants by sub denoting with an 'a' or 'b'. I.e., P1a and P1b refer to the two participants in pair 1. Each session took under one hour, and participants were compensated $20 USD for their participation. On average, participants had 2.6 years of programming experience (range: 0–6 years, std. dev.: 1.95 years).

4.2 Study Procedure

The order of conditions and problems used were randomized across pairs of participants. The first and second authors supervised and directed the web session. The study was structured as follows:

(5 min) Tutorial. Participants were given a tutorial to familiarize themselves with rearranging code fragments and having control over spaces.

(2 min) Introduction to modules. Participants were provided a brief introduction to the upcoming problem set. We emphasized the 10-minute time limit was to ensure that participants are able to view all collaboration conditions, but that there was no expectation to complete each problem in 10 minutes. They were encouraged to communicate and collaborate on each problem.

(10 min x 3) Tasks. Each task began with a brief description of the current condition (Free-for-all, Personal-spaces, or Turn-taking). At the start of the Turn-taking condition, participants were instructed to swap control frequently to ensure that both arrived at an understanding of the code. In the other two conditions, participants were generally instructed to communicate, think out loud, and work through the problem together.

(5 min) Post questionnaire. In the post-questionnaire, we collected demographics information, previous experience in programming,

and the participants' perception of how much they and their partner contributed in each condition, as well as a ranking of the most effective conditions for collaboration.

(10 min) Post interview. We conducted an interview inquiring about the participant's perception of each condition using the dimensions of engagement, frustration, conversation, and value-added by the partner, as well as the effectiveness of Parsons problems compared to previous collaborative experiences (if applicable).

4.3 Implementation and Data Collection

We implemented a client-side web prototype through JavaScript, which displays Python-based Parsons problems. Web sockets were used to implement synchronous collaboration. A Node.js backend server managed client connections and simulated code for evaluating solutions against test cases. All interactions with the web app, such as mouse movement, grabbing of a code fragment, verifying the solution, and swapping control were recorded. The study was conducted remotely, with Zoom for voice communication.

4.4 Study Limitations

The problem difficulty, length, or the order in which the problems were administered may have had an impact on the observed interaction patterns. The task duration at 10 minutes per problem is likely lower than typical collaboration episodes in real pair programming. Moreover, the difficulty of each problem varied: bubble sort was solved 8/9 times, insertion sort 4/9 times, and selection sort 6/9 times. Our chosen problem set was asymmetrical: the parent function was shorter than the helper function. The majority (12/18) of participants had two or more years of programming experience, as such these results may not generalize to early novices. Lastly, these results are based on a small sample, and so provide at best anecdotal evidence, which cannot be generalized to learning outcomes or compared quantitatively to other collaboration paradigms.

5 RESULTS

Three our of our nine pairs were successful in solving all problems (P1, P5, P9), three solved only two of the three problems (P2, P3, P8), and another three were successful in solving only one of three problems (P4, P6, P7). Overall, participants perceived collaboration to be most effective in Shared-control (10/18) and least effective in Turn-taking (9/18), with Personal-spaces perceived as mostly neutral (12/18).

We conducted a thematic analysis of post-interview reflections with a focus on comparing the coordination conditions. The first author coded the transcript, deriving 52 codes (e.g., *"Did not swap control because they did not receive a request to"*). These codes were then used to synthesize eight primary themes, which were reviewed and refined by the first and second authors. Below, we report the results of the thematic analysis.

5.1 General Results

5.1.1 Emergence of new roles. Self-perception of roles plays a crucial part in effective collaboration [5]. Across all conditions, participants perceived themselves in roles that were not designated by the task structure. Roles sometimes emerged via expertise, e.g., P4b conceived of themselves as an "initiator" who got a task in motion – a task more comfortable for their beginner-level programming experience, and distinguished this role from the "leader" which they attributed to their partner. Some other unexpected roles arose through circumstance: P7b identified their testing-based role as a result of the fact that *"P7a had a bigger role because we started with [them] having control first"* (P7b). Some roles were socially driven, e.g., P6b described themselves as the conversation-starter, and began verbally walking through a concrete example in attempt to *"start the conversation, so that we can both try thinking out loud together."*

5.1.2 Parsons problems may help to establish common ground. When asked to compare their experience in this session to previous synchronous collaborative sessions, multiple participants noted that these puzzles help bridge the gap between different starting skill levels *"it's sort of like both people are at the same baseline... it's harder for one person to realize 'Oh that's how we do it' and just finish everything"* (P3b). In addition, Parsons problems may help constrain the space of possible solutions; both partners are working with the same solution components, and so share a starting point (P6b). Instead of an infinite number of lines of code for each participant to try on their own, there are concrete pieces of potential code to discuss and *"you can't like write whatever you want"* (P5b).

5.2 Strategy Specific Results

5.2.1 Personal-spaces promote collaborative dialog. P7a explained that Personal-spaces *"force you to interact with the other person. Because you absolutely can't control their personal space... if you had an idea, you had to communicate it in a way they would understand."* P1b noted that the dependencies of the two methods gave rise to conversation, *"you could say 'Okay my method does this. How does it affect your method?'."* Some participants found this added friction to task completion, P8b felt *"bottlenecked by the other person. I would ask like 'What part of this code do you want?'... I would need to wait for the other person to engage."* P5a was frustrated by being unable to manipulate the other space *"because I wasn't able to drag blocks into [their] section, so I kind of had to like tell [them] what to do."* The requirement for dialogue also appeared to increase the amount of agency participants had individually in forming the solution, as even in cases where participants tried to micromanage the spaces they could not access, they would still rely on the other participant's cooperation. Participants noted that this allowed them to experience agency in the solutions since *"[they're] the one doing it, and not [their partner] invading [their] space and doing it for [them]"* (P4b).

5.2.2 Personal-spaces work towards balanced engagement. P1b noted that even though they were mostly learning from P1a, *"I felt like I did do my part with the Personal-spaces just because I can focus on one small thing and digest it a little more."* Personal-spaces also made learning more intentional compared to other conditions: *"sometimes, if you are not paying attention to what the other person is doing, then you didn't learn that part... when you have your own little space, you're forced to do that part, so you get to learn more"* (P2b). In addition, making participants engage in dialog discouraged one person from taking over, unlike in *"Shared-control one and the Turn-taking [which] were the least for communication and*

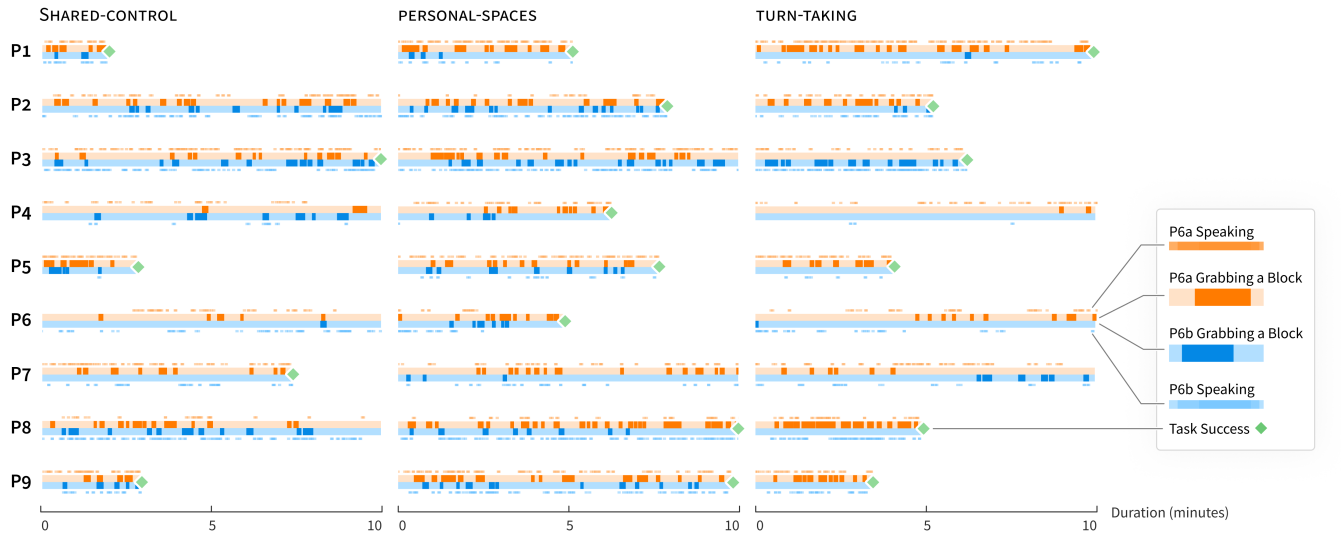

Figure 2: Frequency of participant dialogue and tool use over time grouped by coordination condition. There were no significant differences in the average time taken or in the length of verbal dialog between the conditions. Turn-taking sessions were by far the least symmetric in terms of interactions with the tool with only P7 swapping control to give both partners an even amount of control time. Note, for visibility purposes, tool use visual marks have added thickness, their size does not correspond to the exact time taken.

collaboration. Just because it seems like one person can do most of the heavy lifting." (P4b). However, the push to involve each participant towards engaging in their section highlighted imbalance in workload assigned due to the helper function being more difficult than the parent function: *"I felt like one method required only little code while another method required a lot more code"* (P1b).

5.2.3 Personal-spaces can fragment collaboration. Multiple participants noted that Personal-spaces encouraged divide and conquer, with communication occurring at the end to synthesize the results. P7b noted that instead of actively collaborating, they *"work[ed] on the task that's given to us [in our individual sections] and after a while we come back together and talk about what we just did."* P4a *"just ignore[d] some details in helper function... that part [is] more [their] responsibility, so I only need to take care of my part."* This fragmentation of workload can also fragment the conversation, and P6a noted that *"as soon as we figure out what different roles [our spaces] have, I'll just start focusing on my own role. That's why I tend to talk a little bit less."*

5.2.4 Turn-taking encourages a single train of thought. Turn-taking supported participants in synchronizing their approaches and maintaining joint-attention. In Turn-taking, P3a felt like they and their partner were on the *"same page the whole time... in the other two, I felt like I didn't have the assurance because it felt like [they were] looking at the other side."*. However, a downside of the single train of thought is that it allows for a single participant to take over, or conversely, become extremely passive e.g., P3b did not actively engage with the problem because *"I kind of forgot about bubble sort, so I was in the role of observing"*. Moreover, the condition does not necessitate any dialogue in the event where the driver chooses to solve the problem alone. P7b described this experience as *"both thinking in our heads instead of talking about it more."* (P7b)

5.2.5 Pairs rarely utilize the ability to swap control in order to balance involvement. Despite instructions to do so, participants rarely swapped control (Figure 2) in the Turn-taking condition. When asked how participants decided to swap, only one pair was concerned with balance and thus swapped control *"after five minutes, just to give my partner another five minutes."* (P2a). Instead, swaps were motivated by practical factors like if the other person has an *"idea, while I haven't"* (P4b). However, practical considerations would not always result in a swap. As the Co-pilot, P7b was *"too scared to ask to swap control... [and so] just let the other person take over"*, and as a Pilot, P8b felt like they were *"on a good train of thought... [they] didn't feel the need to swap control or receive a request to swap control."*

5.2.6 Shared-control provides complete agency. Many participants expressed their preference for Shared-control since they were able to execute their own ideas without being 'held back' by their partner. P1a found Shared-control to be the most engaging because it gave them *"the freedom to do what [they thought] should be done"*. Some participants utilized this freedom vary their extent of involvement. P5b shared that there was a give and take between them and their partner as *"there was parts that I wasn't sure about and then [they] help[ed] me with that and... I was able to figure out parts that maybe [they weren't] sure about and then explain it to [them]"*. P5a agreed that in the Shared-control condition *"you complete the parts that you're sure about that I'm not sure about and then contribute that way"*.

However, this independence came at the cost of disorganized collaboration. Shared-control was *"chaotic and you can accidentally change something that your partner thinks you're not going to touch... it could break someone's train of thought"* (P1b). It was difficult to repair conversations: e.g., *"I think the least talking was Shared-control, because we were like looking at the code and moving things, and then just the whole conversation thing kind of felt like a mess"* (P3a). Although some participants felt compelled to describe their thought process to avoid *"making the other one feel that I'm doing things without involving them"* (P6a), they did not feel as if it was required: *"[In Shared-control] I didn't feel obligated to explain because I could just move everything myself"* (P7a).

6 DISCUSSION

Each coordination condition showed strengths and weaknesses with regards to effective collaboration. Shared-control allowed for more agency but can lead to one participant dominating and disorganized communication. Turn-taking promoted a single train of thought but lead to one participant taking the lead. Personal-spaces encouraged both participants to contribute – as the lack of ability to control the other participant's space necessitates dialogue to execute ideas in the other user's space – but lead to division and dominant participants still verbally micromanaged. Considering the potential gains and losses discussed above, we believe that Personal-spaces is a promising approach to allow *both* participants to gain insight into the chosen problem. In Personal-spaces, both participants will have interacted to some extent with the problem, and will have more incentive to discuss the problem.

6.1 Design Considerations and Future Work

Our technique is inherently tied to the characteristics of Parsons Problems (i.e., there being a shared and limited resource with code blocks) and as such it would not directly translate to more general coding environment. We still hope that our guiding design goals and subsequent evaluation can serve as a promising start to investigate how collaborative learning can be structured beyond Parsons Problems. Through our work we have identified three key areas of investigation for collaborative programming interfaces.

6.1.1 Problems should be divided evenly and non-trivially. We observed instances where where participants integrated their solutions at the end with Personal-spaces rather than actively engaging with each other's spaces, which is not ideal for collaborative learning. Designing sub-problems that (1) have dependence, and (2) have enough uncertainty that communication is required to successfully implement them, is crucial to encourage cooperation. Additionally, in the problem sets we constructed, there was a clear imbalance of workload: the helper functions were more challenging than the parent function. By attempting to fix the asymmetry in Turn-taking by giving each participant their own tasks, there is an obligation to keep these separate workloads (at least roughly) balanced. One option to approach this would be to use less strictly defined problems, which may encourage sharing of the workload between finding and communicating requirements. Variations in Parsons problems, such as faded Parsons problem, which add ambiguity to the fragments [32], or sub-goals [22], which are at a finer level of granularity than functions, can also enable new and interesting designs of such divided problem sets.

6.1.2 Structuring the task structures the conversation. We found that structuring the task, either with Turn-taking or with Personal-spaces, helped structure the dialog between participants. Turn-taking enabled pursuit of a more consistent 'vision' of the overall solution while Personal-spaces emphasized negotiations over egocentric solutions. However, this work has only investigated participant *perceptions* of each condition. Future conversational analysis is needed to shed light on the discourse between learners: such as the kinds of dialog (e.g. handling conflicts, thinking-out-loud, [31]), conversational turn-taking and synchronization that occurred in each condition. While our analysis found no pair-wise effects, observing participants' conversations in-situ can illuminate potential effects of interpersonal dynamics and learning styles (e.g., as follower or leader).

6.1.3 Remote collaboration tools can leverage new kinds of interactions to help reach common ground. Participants organized and re-organized the shared space to spatially reason together about the problem. They frequently used their cursor to point out objects and locations, as well as mimic their thought process through gestures with their cursor. Adding expressivity, such as annotations, to the tool may thus make it easier to repair gaps in understanding, speaking, or hearing. Additionally, a visualization that helps direct the conversation could also be a promising direction. For example, visualizing how long each person was in control, or visualizing the frequency of conversational turn-taking may help learners reflect on and encourage more democratic collaboration [18].

7 CONCLUSION

We have presented the design of a novel collaboration technique, Personal-spaces, which addresses the imbalance in pair programming for collaborative learning by dividing the problem into two parts and assigning each to one learner. We provide preliminary evidence of the usefulness of this paradigm: Personal-spaces were unique in their insistence that both participants construct (parts of) the solution. This level of interaction contributed to a sense and responsibility and ownership over the solution and discouraged dominant members from completely taking over the solution. However, in worse scenarios, participants would divide and conquer to focus purely on their portion of the code, or become over-involved in the other person's space and instruct them how to complete their solution. Our results highlight the need for structure in collaborative tools, use of balanced but ambiguous problem sets, and provide promising leads for interaction design of future remote collaboration systems.

REFERENCES

[1] Praveen Aggarwal and Connie L O'Brien. 2008. Social loafing on group projects. *J. Mark. Educ.* 30, 3 (Dec. 2008), 255–264.
[2] Carolina Alves De Lima Salge and Nicholas Berente. 2016. Pair Programming vs. Solo Programming: What Do We Know After 15 Years of Research?. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*. 5398–5406. https://doi.org/10.1109/HICSS.2016.667
[3] Brigid Barron. 2003. When Smart Groups Fail. *The Journal of the Learning Sciences* 12, 3 (2003), 307–359. http://www.jstor.org/stable/1466921
[4] Maxwell Bigman, Ethan Roy, Jorge Garcia, Miroslav Suzara, Kaili Wang, and Chris Piech. 2021. PearProgram: A More Fruitful Approach to Pair Programming. In

Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 900–906. https://doi.org/10.1145/3408877.3432517

[5] Susan Brewer and James D. Klein. 2006. Type of Positive Interdependence and Affiliation Motive in an Asynchronous, Collaborative Learning Environment. *Educational Technology Research and Development* 54, 4 (2006), 331–354. http://www.jstor.org/stable/30220464

[6] Clark A. Chinn, Angela M. O'Donnell, and Theresa S. Jinks. 2000. The Structure of Discourse in Collaborative Learning. *The Journal of Experimental Education* 69, 1 (2000), 77–97. http://www.jstor.org/stable/20152650

[7] Pierre Dillenbourg. 2002. Over-scripting CSCL: The risks of blending collaborative learning with instructional design.

[8] Barbara J. Ericson, Paul Denny, James Prather, Rodrigo Duran, Arto Hellas, Juho Leinonen, Craig S. Miller, Briana B. Morrison, Janice L. Pearce, and Susan H. Rodger. 2022. Parsons Problems and Beyond: Systematic Literature Review and Empirical Study Designs. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (Dublin, Ireland) *(ITiCSE-WGR '22)*. Association for Computing Machinery, New York, NY, USA, 191–234. https://doi.org/10.1145/3571785.3574127

[9] Barbara J. Ericson, Lauren E. Margulieux, and Jochen Rick. 2017. Solving Parsons Problems versus Fixing and Writing Code. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '17)*. Association for Computing Machinery, New York, NY, USA, 20–29. https://doi.org/10.1145/3141880.3141895

[10] Stanley M Gully, Kara A Incalcaterra, Aparna Joshi, and J Matthew Beauien. 2002. A meta-analysis of team-efficacy, potency, and performance: interdependence and level of analysis as moderators of observed relationships. *J. Appl. Psychol.* 87, 5 (Oct. 2002), 819–832.

[11] Brian Hanks. 2005. Student Performance in CS1 with Distributed Pair Programming. *SIGCSE Bull.* 37, 3 (jun 2005), 316–320. https://doi.org/10.1145/1151954.1067532

[12] Petri Ihantola and Ville Karavirta. 2011. Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations. *Journal of Information Technology Education: Innovations in Practice* 10 (01 2011), 1–14. https://doi.org/10.28945/1394

[13] Lindsay Jarratt, Nicholas A. Bowman, K.C. Culver, and Alberto Maria Segre. 2019. A Large-Scale Experimental Study of Gender and Pair Composition in Pair Programming. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 176–181. https://doi.org/10.1145/3304221.3319782

[14] David W. Johnson and Roger T. Johnson. 2009. An Educational Psychology Success Story: Social Interdependence Theory and Cooperative Learning. *Educational Researcher* 38, 5 (2009), 365–379. http://www.jstor.org/stable/20532563

[15] David W. Johnson, Roger T. Johnson, and Karl Smith. 2007. The State of Cooperative Learning in Postsecondary and Professional Settings. *Educational Psychology Review* 19, 1 (2007), 15–29. http://www.jstor.org/stable/23363866

[16] Sandeep Kaur Kuttal, 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)*. 75–85. https://doi.org/10.1109/VLHCC.2019.8818790

[17] Colleen M. Lewis and Niral Shah. 2015. How Equity and Inequity Can Emerge in Pair Programming. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, Nebraska, USA) *(ICER '15)*. Association for Computing Machinery, New York, NY, USA, 41–50. https://doi.org/10.1145/2787622.2787716

[18] Jialang Victor Li, Max Kreminski, Sean M Fernandes, Anya Osborne, Joshua McVeigh-Schultz, and Katherine Isbister. 2022. Conversation Balance: A Shared VR Visualization to Support Turn-Taking in Meetings. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 181, 4 pages. https://doi.org/10.1145/3491101.3519879

[19] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. 2002. The Effects of Pair-Programming on Performance in an Introductory Programming Course. *SIGCSE Bull.* 34, 1 (feb 2002), 38–42. https://doi.org/10.1145/563517.563353

[20] C. Mcdowell, L. Werner, H.E. Bullock, and J. Fernald. 2003. The impact of pair programming on student performance, perception and persistence. In *25th International Conference on Software Engineering, 2003. Proceedings.* 602–607. https://doi.org/10.1109/ICSE.2003.1201243

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

[22] Briana B. Morrison, Lauren E. Margulieux, Barbara Ericson, and Mark Guzdial. 2016. Subgoals Help Students Solve Parsons Problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 42–47. https://doi.org/10.1145/2839509.2844617

[23] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. 2003. Improving the CS1 Experience with Pair Programming. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Navada, USA) *(SIGCSE '03)*. Association for Computing Machinery, New York, NY, USA, 359–362. https://doi.org/10.1145/611892.612006

[24] John T. Nosek. 1998. The Case for Collaborative Programming. *Commun. ACM* 41, 3 (mar 1998), 105–108. https://doi.org/10.1145/272287.272333

[25] Dale Parsons and Patricia Haden. 2006. Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (Hobart, Australia) *(ACE '06)*. Australian Computer Society, Inc., AUS, 157–163.

[26] Fernando J. Rodríguez, Kimberly Michelle Price, and Kristy Elizabeth Boyer. 2017. Exploring the Pair Programming Process: Characteristics of Effective Collaboration. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 507–512. https://doi.org/10.1145/3017680.3017748

[27] Karin Scager, Johannes Boonstra, Ton Peeters, Jonne Vulperhorst, and Fred Wiegant. 2016. Collaborative Learning in Higher Education: Evoking Positive Interdependence. *CBE—Life Sciences Education* 15, 4 (2016), ar69. https://doi.org/10.1187/cbe.16-07-0219 arXiv:https://doi.org/10.1187/cbe.16-07-0219 PMID: 27909019.

[28] Max O. Smith, Andrew Giugliano, and Andrew DeOrio. 2018. Long Term Effects of Pair Programming. *IEEE Transactions on Education* 61, 3 (2018), 187–194. https://doi.org/10.1109/TE.2017.2773024

[29] Despina Tsompanoudi, Maya Satratzemi, Stelios Xinogalos, and Leonidas Karamitopoulos. 2019. An Empirical Study on Factors related to Distributed Pair Programming. *International Journal of Engineering Pedagogy (iJEP)* 9 (04 2019), 61. https://doi.org/10.3991/ijep.v9i2.9947

[30] Simon Veenman, Eddie Denessen, Anneriet van den Akker, and Janine van der Rijt. 2005. Effects of a Cooperative Learning Program on the Elaborations of Students during Help Seeking and Help Giving. *American Educational Research Journal* 42, 1 (2005), 115–151. http://www.jstor.org/stable/3699457

[31] Astrid J S F Visschers-Pleijers, Diana H J M Dolmans, Bas A de Leng, Ineke H A P Wolfhagen, and Cees P M van der Vleuten. 2006. Analysis of verbal interactions in tutorial groups: a process study. *Medical Education* 40, 2 (2006), 129–137. https://doi.org/10.1111/j.1365-2929.2005.02368.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2929.2005.02368.x

[32] Nathaniel Weinman, Armando Fox, and Marti A. Hearst. 2021. Improving Instruction of Programming Patterns with Faded Parsons Problems. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 53, 4 pages. https://doi.org/10.1145/3411764.3445228

[33] Linda L. Werner, Brian Hanks, and Charlie McDowell. 2004. Pair-Programming Helps Female Computer Science Students. *J. Educ. Resour. Comput.* 4, 1 (mar 2004), 4–es. https://doi.org/10.1145/1060071.1060075

[34] Laurie Williams, D. Scott McCrickard, Lucas Layman, and Khaled Hussein. 2008. Eleven Guidelines for Implementing Pair Programming in the Classroom. In *Agile 2008 Conference*. 445–452. https://doi.org/10.1109/Agile.2008.12

[35] Merlin C. Wittrock. 1989. Generative Processes of Comprehension. *Educational Psychologist* 24, 4 (1989), 345–376. https://doi.org/10.1207/s15326985ep2404{_}2 arXiv:https://doi.org/10.1207/s15326985ep2404_2