Supporting Collaboration in Introductory Programming Classes Taught in Hybrid Mode: A Participatory Design Study

Lahari Goswami
lahari.goswami@unil.ch
University of Lausanne
Lausanne, Switzerland

Pegah Sadat Zeinoddin
pszeineddin@gmail.com
University of Lausanne
Lausanne, Switzerland

Thibault Estier
thibault.estier@unil.ch
University of Lausanne
Lausanne, Switzerland

Mauro Cherubini
mauro.cherubini@unil.ch
University of Lausanne
Lausanne, Switzerland

ABSTRACT

Hybrid learning modalities, where learners can attend a course in-person or remotely, have gained particular significance in post-pandemic educational settings. In introductory programming courses, novices' learning behaviour in the collaborative context of classrooms differs in hybrid mode from that of a traditional setting. Reflections from conducting an introductory programming course in hybrid mode led us to recognise the need for re-designing programming tools to support students' collaborative learning practices. We conducted a participatory design study with nine students, directly engaging them in design to understand their interaction needs in hybrid pedagogical setups to enable effective collaboration during learning. Our findings first highlighted the difficulties that learners face in hybrid modes. The results then revealed learners' preferences for design functionalities to enable collective notions, communication, autonomy, and regulation. Based on our findings, we discuss design principles and implications to inform the future design of collaborative programming environments for hybrid modes.

CCS CONCEPTS

• **Human-centered computing → Participatory design**; **Collaborative and social computing systems and tools**; • **Social and professional topics → CS1**.

KEYWORDS

collaboration, hybrid classroom, participatory design, programming environment

ACM Reference Format:
Lahari Goswami, Pegah Sadat Zeinoddin, Thibault Estier, and Mauro Cherubini. 2023. Supporting Collaboration in Introductory Programming Classes Taught in Hybrid Mode: A Participatory Design Study. In *Designing Interactive Systems Conference (DIS '23), July 10–14, 2023, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3563657.3596042

1 INTRODUCTION

Learning to program is a complex activity [18]. It requires novices to learn basic programming concepts, develop procedural skills to perform tasks with these concepts [60], and learn to regulate their domain-specific cognitive processes [21]. Introductory programming courses typically introduce beginners to these concepts and teach them how to master programming skills by solving programming exercises during lectures or labs [58]. Especially in universities, these courses also leverage the social setting of classrooms to employ collaborative pedagogical practices to teach programming. One of the most common collaborative learning techniques implemented in introductory programming classrooms involves problem-solving through group activities [37]. In these activities, learners often need to interact collaboratively with their peers, coordinate to varying degrees and work on code together. Learning through collaboration in classrooms is contextualised by its unique social setting involving course activities, curricula, social interactions, teachers' actions and learning tasks [51]. In particular, to successfully solve problems collaboratively, interacting learners need to create and maintain a shared conceptual space. This involves socially negotiating their knowledge elements, goals, problem descriptions and problem solving actions [49]. This also extends to programming classrooms. As such, for students in programming classrooms, their learning environment should also enable team members collaborating on programming problems to co-construct task understanding through interdependent interactions.

In introductory programming courses, collaborative interactions on group programming activities and code can occur synchronously or asynchronously in co-located or remote environments. In addition, following the pandemic, many introductory programming courses shifted from traditional face-to-face teaching to hybrid learning modes. Students can attend a class in person or remotely in these hybrid-taught classrooms. Therefore, members of group activities in hybrid classrooms may sometimes be co-located, sometimes remote, or sometimes partially distributed. [13]. Previous research in the field of computer-supported cooperative work (CSCW) has highlighted the difference between collaborative interactions in co-located and remote modes [43, 56]. In hybrid formats, Neumayr et al. showed these interactions vary further from other modes, as collaboration is not confined to the dichotomies but occurs across the continuum of space (co-located and remote settings) and time (synchronous and asynchronous) [42]. This essentially applies to collaborative learning interactions in group problem-solving activities in hybrid programming courses. Unlike co-located settings, in remote and hybrid modes, learners' primary collaborative interactions on programming tasks take place within programming environments. Critical to this collaboration and subsequent learning is the awareness of group members' actions and intentions across the shared coding space and the ability to maintain a discourse to facilitate this. This necessitates support to be built into programming workspace tools for coordinating, communicating, and collaborating in solving group programming activities.

Many collaborative coding environments and services have been developed to facilitate collaboration in programming workspaces, especially for remote modes. Some of these environments allow programmers to work together on the same code in an asynchronous manner, while others enable synchronous collaboration. These include environments which are research prototypes [9, 50, 59], and also commercial products like `GitHub` [1], `Studio Live Share` [2] for Microsoft's Visual Studio, Google's `Colab` [3] and, `JupyterLab` [4]. Various features of these applications provide the ability to edit code together, share coding environments, track the actions of other members, have built-in communication channels, and some support real-time collaboration. Despite these platforms' availability, most are aimed towards expert users, and the few designed for educational purposes for beginners are not widely used in teaching. Furthermore, the design of existing collaborative coding applications often does not take into account the specific collaborative work processes of novice learners. Only Ying et al. [62] conducted a study to explore students' perspectives on the use of commercially available collaborative coding tools, and concluded that there is a lack of novice-friendly programming environments. Consequently, there is a need for programming and teaching environments that prioritize needs and goals of student users. In addition, with the recent emergence of hybrid learning modes in introductory programming courses, it is imperative to understand how collaborative learning interactions unfold in group programming activities and how to design programming workspaces that foster effective collaboration in these settings. Although some programming environments for remote collaboration can be used in hybrid settings, research in the field of CSCW, shows that interactions in hybrid collaboration are more nuanced. Our reflections from conducting an introductory programming course (see Sec. 3) also led us to realise that existing set of tools available to students were unable to efficiently support interactions in classrooms designed for hybrid learning. Therefore, we pose the following research question:

> **RQ**. How might we best support students' interactions in a hybrid course teaching introductory programming to first year students?

With the goal of informing design of programming workspaces for novices from a user-centered perspective, we take the approach of Participatory Design. We directly engaged nine students from a hybrid-taught introductory programming course, in a participatory design workshop session consisting of four activities. Through this design workshop, we seek to investigate behaviours and challenges faced by novice programmers when collaborating in hybrid modes and gain an understanding of their perspectives on the designs of collaborative programming workspaces to support their work processes. We qualitatively analysed the challenges participants expressed, and the designs they produced, and described four characteristics students sought in programming platforms. Particularly, we identified their preference for shared contextual resources (e.g., sharing and synchronous editing of code snippets) within the programming workspace to facilitate their collaborative problem-solving processes. At the same time, novices required maintaining their personal space within the larger workspace to self-regulate and adapt to the collaborative activity. Based on our findings, we contribute to the design of programming environments in hybrid-taught programming classrooms by presenting four design considerations.

2 LITERATURE REVIEW

2.1 Introductory Programming

The development of computational thinking and programming skills is becoming increasingly relevant and important for students across a range of disciplines [19]. However, programming is not easy to learn and requires mastering and applying a wide selection of knowledge and skills [18, 28, 45]. Introductory programming courses often expose beginners to programming knowledge and teach them necessary domain-specific skills. Students are required to learn basic programming concepts, code-writing skills [60], and how to self-regulate their progress while programming [21]. This complex process poses many challenges for novices who begin to learn programming in introductory courses. Programming is a practical subject, and mastering the relevant skills requires regular practice of programming activities [15]. To teach these skills to novices, the content of introductory courses typically combines theory with examples, lab exercises, and assignments, which require students to solve programming problems, both in groups and individually [37]. These courses and assignments are typically structured to reflect the concept of deliberate practice that contributes to gaining expertise in a particular skill [20].

Introductory courses, in universities or Massive Open Online Courses (MOOCs) orient their content and activity designs in several ways to facilitate this practice-based learning. In MOOCs, content and activities are mostly asynchronous, learner-paced, and although they incorporate some degree of collaborative pedagogy, support for learner collaboration is limited [23, 63]. In contrast, in the formal setup of universities courses and activities are delivered synchronously to students, who are usually from similar pedagogical backgrounds, regardless of whether they are co-located or remote. It entails simultaneous interactions between peers, teachers and activities. As such courses in university classrooms often leverage its unique social setting to instil learning through collaborative programming pedagogies—like peer assessment [55], pair programming [44], and project-based collaborative learning [2]. Some courses' designs also replace traditional lecture formats with an 'inverted' or 'flipped' classroom approach, including more in-class activities such as practice exercises, collaborative problem-solving, and teamwork. These different modes have been shown

[1] https://github.com/, last accessed May 2023.
[2] https://visualstudio.microsoft.com/services/live-share/, last accessed May 2023.
[3] https://colab.research.google.com/, last accessed May 2023.
[4] https://jupyterlab.readthedocs.io/en/stable/user/rtc.html, last accessed May 2023.

to increase student participation [11], foster their sense of community [33] and better performance [32, 46, 61]. Additionally in recent times, post the pandemic, introductory programming courses in universities are also adapting to hybrid modes of learning. A full review of *hybrid learning* is beyond the scope of this paper, as it encompasses a range of pedagogical approaches distributed across different modes [1]. In the context of formal university classrooms, hybrid modes of programming learning, allow learners to attend and follow the activities in a class in person or remotely, usually requiring them to engage in simultaneous interactions during the course time for in-class programming activities. As such, integrating collaborative teaching strategies into different modes of introductory programming courses requires enabling and fostering effective collaborative learning interactions between students specific to their learning context.

2.2 Collaborative Learning And Regulation

The classroom as a learning environment is a social setting, where learning is contextualised through interrelated activities, curricula, social interactions, teacher actions and learning tasks [51]. Likewise, introductory programming courses in classrooms form a social setting for learning. Collaborative learning is a prevalent pedagogical approach that emphasises interactions between learners in social settings as the key factor in their learning process [16]. Learning through collaboration is shaped by how team members construct shared understanding through interaction with each other, where the members are engaged in shared goals and problem-solving [49]. In this context, contemporary educational psychology grounds learning regulation as a social phenomenon. In their theory, Hadwin et al. [26] defines three modes of social regulation of learning in collaboration: (i) *self-regulation* as individual learner's regulatory process in a collaborative context, which is the precursor to optimal productive collaboration in the context of a group task, (ii) *co-regulation* as transitional and flexible stimulation of regulation through interpersonal interactions and exchanges between learners in a collaborative context, and (iii) *socially shared regulation* as interdependent processes by which group members, who work towards a co-constructed or shared outcome regulate their own and collective activity.

Prior research in programming establishes, *self-regulation* of learning as one of the fundamental skills crucial for success in programming learning [3, 24, 54]. Loksa et al. [36] defines programming self-regulation as the process in which learners are aware of their thoughts and actions while evaluating their progress toward writing a program to solve a computational problem. In group work, shared regulation is created through interactive and interdependent processes where individual reasoning builds on, and relates to, reasoning shared by other group members. Thus, during productive collaboration on a group task, self-, co-, and shared regulation of learning occur simultaneously and reciprocally over time in physical and social contexts. Therefore in introductory courses, to program collaboratively, the social learning setting needs to support interactions between learners, such that their programming self-regulation, as well as other social modes of programming regulation are equally supported.

2.3 Collaborative Working In Programming Learning

In introductory programming, collaborative learning facilitates synergy among peers and develops their critical thinking and programming abilities in classrooms or remote learning environments [37]. Collaborative programming activities in introductory courses require learners to engage in various degrees of collaboration to solve or debug programming problems along with their peers, often requiring them to code together on the same problem. For example, in peer review [55], learners evaluate code written by their peers and provide feedback, in pair programming [44], pairs of learners work on the same code on the same computer in a turn-taking manner, and in project-based collaborative learning [2], learners work on a programming project in groups over a period of time. In hybrid modes of learning, these collaborative interactions on code can occur synchronously with programmers working in real-time or can be asynchronous. These interactions may also occur in co-located learning settings or remote contexts.

Many Computer-Supported Cooperative Work (CSCW) researches have established the inherent difference between collaborative paradigms between remote and co-located collaborations and investigated remote collaborative work and problem solving [43, 56]. To collaborate on problems in hybrid modes, Neumayr et al. [42] explain that in groups, members may be co-located, partially co-located, partially remote, or completely remote, and interactions may also occur synchronously via shared editing or screen sharing for any duration, or asynchronously via forums and comment sections. As a result, collaborators' interactions in hybrid mode span the continuum of space (i.e. remote and co-located formats) and time (i.e. synchronous and asynchronous modes), especially in partially distributed groups where at least one group member is located elsewhere and connected to the rest of the team via computer-mediated channels [10]. Therefore, collaborative coding processes for programming problem-solving during group activities in these hybrid learning modes will be fundamentally different from co-located classroom scenarios.

As noted in the previous section, in order to coordinate effectively on a collaborative learning task, team members need to gain a shared understanding of the problem, which necessitates collaborating learners to know peers' actions, interaction history and intentions [25]. When learners work together in a co-located environment, coding on the same laptop like in pair programming or on separate laptops, they can engage in spontaneous communication, use gestural cues to coordinate, and gain interpersonal awareness of each other's context to build a shared understanding while coding together. However, collaborative work processes between programmers in remote and hybrid modes require support for coordination, communication and awareness through programming workspace tools. In this context, workspace awareness is a critical factor in educational collaborative software systems, not only for effective collaboration, but also to create opportunities for collaborative learning [25]. It allows collaborating learners to easily perceive and gain knowledge of the interactions taking place with other people in the shared workspace. Furthermore, support is required to sustain learners' coordination through facilitating collaborative dialogue and discourse within these shared workspaces.

2.4 Collaborative Coding Environments

Many collaborative coding environments have been designed and developed to facilitate collaboration in programming workspaces, especially in a remote context to work synchronously. Most of these platforms allow learners to work together on the same problem, with some support for providing awareness of other members' actions. Examples include: Saros [50], an `Eclipse` plug-in that displays each coder's text cursor to communicate its location in the source code and provides a follow mode for coders to synchronise their programming platform viewports during pair programming and code walkthroughs; COLLECE [9], a groupware system that supports collaborative editing, compilation and execution of programs in a synchronously distributed manner with mechanisms for communication, coordination and workspace awareness; RIPPLE [6], a distributed synchronous open-source software tool, specifically for educational contexts, that enables two programmers to work remotely on the same program at the same time; CodePilot [59], an Integrated Development Environment (IDE) for novices which integrates coding, testing, bug reporting, and version control management into a real-time collaborative system for situational awareness and impromptu collaboration. Real-time collaboration support for coding with peers is also available in many commercial programming environments like `Studio Live Share` for Microsoft's Visual Studio, `Code With Me` [5] in JetBrains IDE, and also in web-based collaborative working environments like `CoCalc` [6], Google's `Colab` and recently in open-source solution JupyterLab. Few commercial platforms that support non-real time code collaboration include version control systems like `Subversion`[7] and `GitHub`.

These programming environments facilitate both synchronous and asynchronous collaborative interaction between programmers regardless of their locations. As such these environments have the potential to support hybrid collaboration in programming. Despite the availability of these tools, most are primarily designed for industrial needs and aimed at expert users. Also, the design of collaborative programming environments for educational contexts, especially for beginners, is limited and existing tools are not widely used. Some of these educational collaborative programming environments take into account novices' collaborative work processes and design to support that. However they do not include novice end-users first-hand perspectives in the design of these environments. Only a study by Ying et al. [62], investigated students' perspectives on using available programming tools for coding collaboratively. They conducted a large-scale survey with novices and showed that current available programming tools do not completely support novices' needs. The findings revealed that using version control systems and live-shared coding platforms were hard for novices to learn and integrate in their collaborative work processes. Furthermore, with the recent adoption of hybrid modes of learning in introductory classrooms, it is critical to understand how students' collaborative problem-solving processes unfold in such settings. As reflected from CSCW literature, interactions in hybrid workspaces are more nuanced than other learning settings. This establishes a requirement for novice-friendly programming environments that foster effective collaborations in these new modes of learning.

[5]https://www.jetbrains.com/code-with-me/, last accessed May 2023.
[6]https://cocalc.com/, last accessed May 2023
[7]https://subversion.apache.org/, last accessed May 2023

3 STUDY CONTEXT

The research question of this study stems from our lessons and reflections on a hybrid-taught introductory programming course which took place between February and June 2022. The pedagogical goal of this course was to introduce Computer Science (CS) programming syntax, semantics, the basics of the Python programming language, and the most common programming concepts to first-year undergraduate students pursuing a Bachelor of Science in either Economics or Management. Thus, most of the students in this course had no background in CS. Given the course was delivered at a time when the COVID-19 pandemic was coming to an end, the university where it was conducted adapted to a hybrid teaching format. As a result, the course was designed so that students could attend classes in person or follow remotely via live streaming. The course used `Jupyter Notebook`[8], which is hosted on `JupyterHub`[9], as the programming environment to teach and practice Python programming to the students. This platform allowed the instructors to easily distribute the notebooks and other pedagogical materials easily to the large number of students enrolled in the course. The course had ∼600 students registered, of which ∼400 typically attended the course in presence, and ∼200 online. To allow students to follow the course, these were given early access to the course resources, typically a week before each class. The resources includes the session slides, videos of theoretical explanations (typically 3 to 5, of ∼10 minutes each), and exercises to be practised during the class. Classes were scheduled over 4 hours during the week. Class sessions were devoted to lectures, discussion of guided examples and hands-on activities. During these activities, students were encouraged to work collaboratively either with co-located colleagues or with colleagues attending the course online. To facilitate interaction between people onsite and people online, we provided a plugin (Supplementary 1 (Supp. 1))[10] for Jupyter that enabled students to share cells of their active notebook and to chat with their colleagues. In addition to the plugin, students were also free to communicate and share code using the medium of their choice (e.g., `WhatsApp`, `Telegram`, `Skype`). Finally, students were also encouraged to work on an optional capstone project to further strengthen their programming solving abilities, on which they had to collaborate with six to ten classmates of their choice. Collaboration for this group project happened outside of class hours and students were free to choose modes and frequencies of interaction with their classmates. At the end of the semester, students were evaluated with a final exam, which consisted of multiple-choice questions that required them to solve programming problems.

The authors of this study were directly involved in conducting of this course as either the instructors or teaching assistants. As a result, the researchers were able to gain first-hand insights on how students tried to regulate their learning behaviour within the context of the course and its setting. Specifically, throughout the

[8]https://jupyter.org/, last accessed May 2023.
[9]https://jupyter.org/hub, last accessed May 2023.
[10]Details about this intervention with figures of the plugin, and all other supplementary materials of our study are available on the Open Science Framework (OSF) repository. See here: https://doi.org/10.17605/osf.io/985FQ.

semester observations and interviews were conducted to understand how students interacted to complete the different pedagogical activities and how the different tools that were available to them were appropriated to support collaboration. From these activities we derived the following:

- Little to no interaction between students onsite and students online was observed.
- While we expected the Jupyter plugin to support both students online interacting with students onsite (and onsite with onsite and online with online), it was seldom used. We also observed little use of other communication apps during the classes.
- Successful collaboration happened mostly for co-located students who sat together in the classrooms. These students typically had their laptops open and stared at each other screens while discussing and working on the exercises.

Reflections from conducting the course led to realise that existing set of tools that was available to students was unable to efficiently support interactions in programming classrooms designed for hybrid learning. Particularly, questions arose around whether specific features could be added to the tools available to the students or whether it was a question of the contextual environment (e.g., regulations of the class, logistic setups) which did not allow successful interactions between students online and students onsite. Therefore, to understand how to better support first-year students' interactions in hybrid programming courses and classrooms, we conducted the study described in the following section.

4 METHODOLOGY

We conducted a qualitative user study to gain insights into novice collaborative behaviours in hybrid programming contexts and uncover effective ways to support them. To do so we employed a participatory design workshop (see Fig. 1) involving nine participants. The study was conducted with the students from the first year programming course, as described in Sec. 3 and was approved by our institution's IRB.

4.1 Participants and Recruitment

To recruit participants for the user study, we invited the students of the same Elementary Programming course in 2022. We wanted to involve students in the study who had already been exposed to the specific context of the course and could therefore provide relevant insights in the design workshop. We invited a total of N=244 students who actively engaged in the course activities during the classes. Students active engagement was already determined during the run-time of the course from their activity log on the exercises provided they attended and engaged in more than 80% of the classes in the course. The invitations for the participatory design workshop were sent out three months after the end of the course, when the students had already moved on to the next semester of their programme. This led to a very low response rate to our invitations with only 13 responses. Out of them, nine participants registered and attended the design workshop. There were 6 males and 3 females, with a mean age of M = 20.33 (SD = 1.41). Except for one of these participants, the Elementary Programming 2022 course (see Sec. 3) was everyone's first introduction to programming. During the course, all of them have attended at least one class remotely. Each of the participants received a monetary incentive of 54 USD for their participation in the study. During the process of the participatory design workshop (as explained later in Sec. 4.2.2), the participants were randomly divided in to groups. We summarise the participants demographics information across the groups in Table. 1

Table 1: Summary of participants demographics in each group during the participatory design workshop

Group	Participant	Age (in years)	Gender
G1	P1	19	m
	P2	20	m
	P3	21	f
G2	P4	18	f
	P5	20	f
	P6	23	m
G3	P7	21	m
	P8	20	m
	P9	21	m

4.2 Participatory Design Workshop

Participatory design (or PD) is a design philosophy that places people who might benefit from a given technology at the center of the design process aiming at designing this technology. By leveraging the knowledge and expertise of potential users, it aims at designing product and services that better meet their needs and expectations. PD uses an ethnographic lens to comprehensively understand needs and preferences [31, 34, 52]. Numerous studies have utilised participatory design methods to shape the development of learning pedagogies and technologies. Some of these studies focused on teachers involvement [14, 35, 40, 48], while others focused on students' perspectives [5, 27, 47]. Although there are many collaborative interventions in programming environments for both experts and students, their designs often do not directly involve end-users in the design process. Our study focuses on prioritising the perspectives of novices in the design of pedagogical interventions. By adopting a PD approach, we aim to uncover the nuances of their collaborative interactions, and their needs in hybrid programming learning environments. Our research protocol and materials used to facilitate the workshop are available respectively in Supp. 2 and Supp. 3.

4.2.1 Apparatus. The design workshop took place in a large laboratory room to accommodate all participants, equipped with tables and chairs for them to sit together as groups (explained later in Sec. 4.2.2). The entire workshop was conducted in one session by three researchers – the first author as the moderator, and the second author and a research assistant who welcomed and guided the participants across the session and took notes. The participants had access to the Jupyter Notebooks for coding that was available to them during the course. The examples of these notebooks along with the Jupyter environment used during the course are available in Supp. 1.

4.2.2 Procedure. Our design workshop lasted roughly 3 hours. At the beginning of the participatory design session, all participants

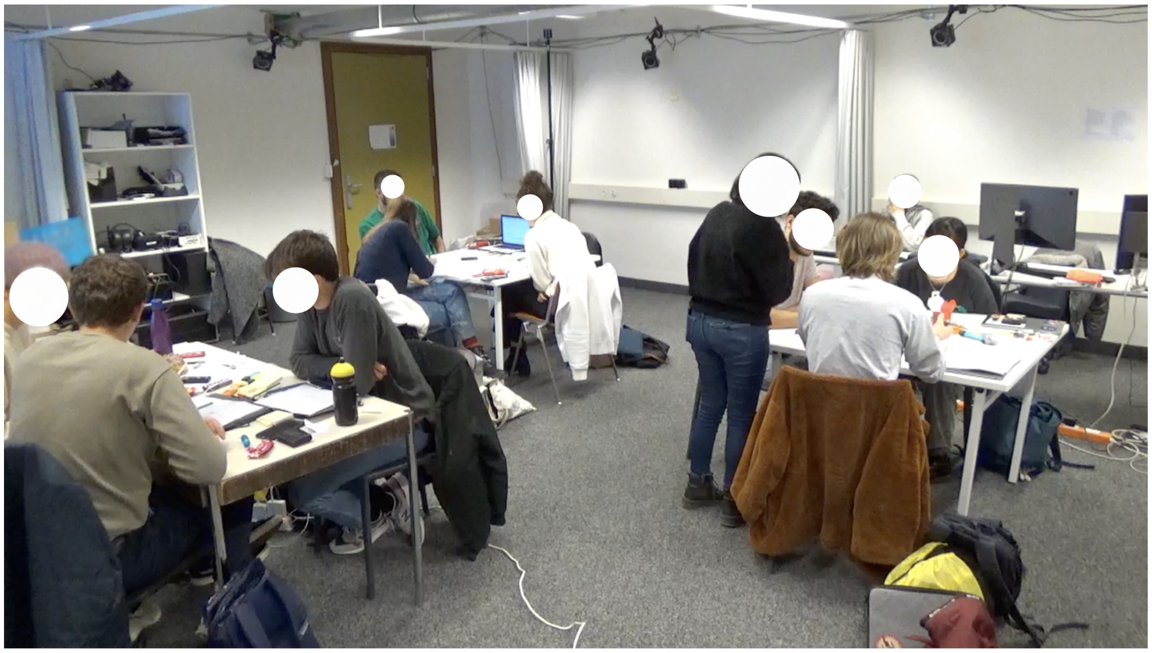

Figure 1: Participatory design workshop session. All participants and researchers faces in has been obfuscated to protect their privacy.

were asked to consent to the recording of their data by signing a consent form[11]. The session articulated five main steps and associated activities, as summarised in Fig. 2. We explain each of the steps below.

(1) *Introduction and Setup.* This phase set the stage for the participatory design workshop. The participants were greeted and first shown a presentation to introduce them to the session's goal, explaining the main tasks and what was expected of them. To provide a collaborative setup for the workshop, we intended to group the participants in smaller teams. Participants were randomly assigned into three separate groups, each with three members. The participants were asked to draw a random chit denoting their designated group number from a mixture of chits with all the group numbers. An overview of the participant demographics of each group are present in Table. 1. We then simulated a hybrid setup (see *Introduction and Setup* section in Fig. 2) for each group's participants to work with their respective members on an activity that followed. To do so, one of the randomly selected participants in each group was allocated to a remote location setup in a isolated room consisting of a table and chair, along with their laptops before beginning of the task. The two other members of the group were physically co-located in the same space.

(2) *Find Issues, Uncover Ideas.* After the introduction and set-up, we presented the participants with the first activity of the session. This activity was designed to elicit insights from the participants about the frustrations they face when trying to program together in the hybrid setup. In this activity, each group was asked to work together with their respective members to try to solve a programming problem. The programming problem was the same for all groups and was made available to them on paper. Due to the limited duration of the workshop, we opted to provide a programming problem comprising smaller steps that could be solved collaboratively in groups instead of assigning a larger group project. The programming problem required them to create a text input field with a validation check that the number of characters entered did not exceed a certain limit. The problem was in French and is available in the OSF repository. Participants were free to use any platform of their choice to code, collaborate and work on the programming problem. While solving the problem, each participant, whether co-located or remote, was instructed to individually take simultaneous notes of the challenges they faced in order to communicate and collaborate effectively. The aim of this activity was not to test participants' problem-solving skills, so we did not obligate them to solve the given problem. The activity lasted 20 minutes and was followed by a break of 10 minutes.

(3) *Reflection on Problem.* Following the break, the members of each group were gathered in the same location to reflect together on the identified issues during hybrid collaboration. The duration of this activity was 15 minutes. The participants were asked to discuss and reflect together in their respective groups on each of the issues they noted down and finally collectively narrow them down to a maximum of five challenges they deemed to be most important. While there was not a hard limit for identifying the number of challenges, we were also mindful that the next activity would expect participants to design solutions for each issue identified. Therefore, we wanted to prevent participants

[11]Consent form available in Supp. 3

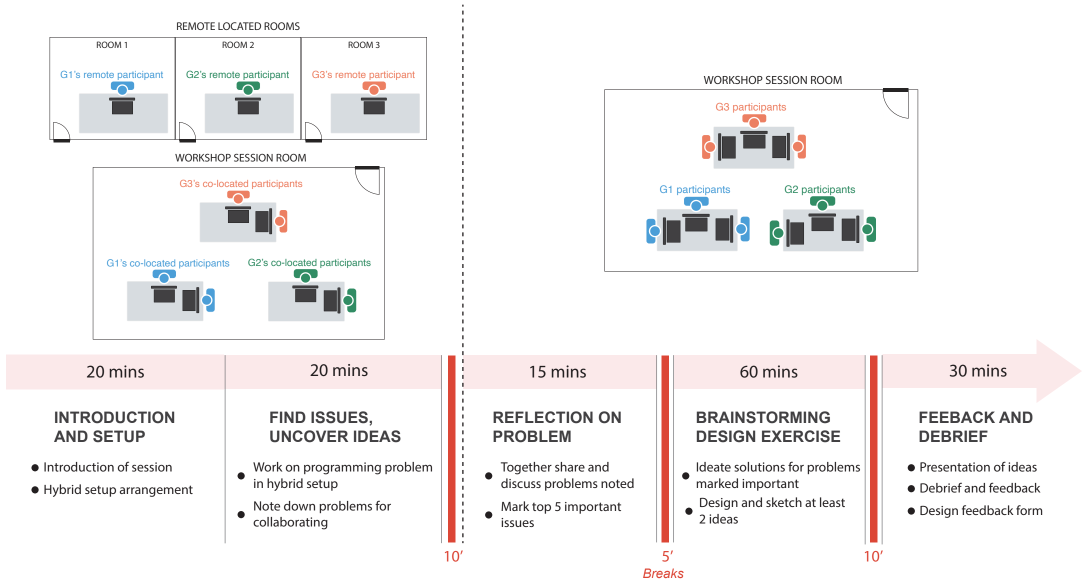

Figure 2: Summary and setup of the different activities and their duration in the participatory design workshop session.

from becoming mentally fatigued and asked them to select the top five issues. The main objective of this activity was to make participants find commonalities in the challenges they faced individually during the previous activity and identify the specific ones they most resonate with. Another break of 5 minutes was given to the participants at the end of the activity.

(4) *Brainstorming Design Exercises.* This activity was the next to follow. It required each group to come up with solutions to the most compelling issues in hybrid collaboration that they had outlined in their previous task. As their solutions, each group was expected to sketch and provide at least two final design diagrams or ideas to tackle the problems. We wanted the subsequent designs from the participants to reflect a diversity of concepts; therefore, we urged them to create as many designs as possible, the minimum being two. We anticipated that some participants might be unfamiliar with the sketching activity. Therefore, to guide them, we offered practical tips on how to conduct rapid sketching with examples. To stimulate their brainstorming and engage them in the designing process, we provided each group with design kits consisting of markers, pens, pencils, post-it notes and A4 paper. Participants were also encouraged to discuss, note down any new problem they discovered at this stage and design potential solutions to mitigate those problems. To facilitate this activity, the researchers would visit each group's table and ask about the challenges that participants identified individually and as a team, as well as ensure that all members were expressing their points of view and actively involved. The participants would then prompted to explain the corresponding solutions they were trying to design for the identified process. During this design process, if participants got stuck, the researchers would guide them to describe their ideas in text as bullet points and then try to sketch based on that. During these discussions, the researchers would also ask questions to further stimulate their ideas and design process while being careful not to bias their thought process. Through the sketches from this activity, we wanted to capture participants' perspectives on the supports they envisioned within coding platforms to optimise their collaborative interactions during the programming learning process. This activity lasted 60 minutes, which followed a 10 minute break.

(5) *Feedback and Debriefing.* This was the final activity which was essentially a plenary review session of the designs produced by the participants. Each group had to briefly present and describe their identified problems and solution ideas to the other groups and researchers. After each presentation, other participants were invited to give feedback and share their thoughts in order to gain perspective on the designs produced. The researchers also participated in these discussions to stimulate the audience and understand the participants' point of view. At the end of all presentations, each group was asked to complete a short design evaluation questionnaire[12] to provide a subjective assessment of the advantages and disadvantages of the designs produced by the other groups.

4.3 Data Collection

The workshop session was audio and video recorded and the design materials created by the participants were collected for analysis. The video data was only used as a reference to visualise or listen to parts of the session that were not properly captured by the audio recording or notes. During the first session activity, the researchers observed and took written notes of how the different groups attempted to work together in the hybrid setup in order to observe their choice of collaborative medium, understand collaborative behaviours and problems they encountered. The researchers then collected all the materials the participants produced during the

[12]The design evaluation questionnaire is available in Supp. 3.

session's different activities (available in Supp. 4). This included the following: (i) list of problems that each group had identified during the programming problem solving process in the hybrid setup, (ii) all the design diagrams that the groups had produced as an output of the session, and (iii) the design evaluation form the participants had filled in at the end of the session. The audio of the plenary feedback and discussions on the presented designs at the end of the session were also recorded, and transcribed by the first author to analyse later.

4.4 Data Analysis Method

All data collected from the workshop sessions were qualitatively analysed, along with the session observation notes, by the first author and a research assistant with a background in Human Computer Interaction (HCI). First, the session's observational data from the notes taken during the *"Find Issues, Uncover Ideas"* activity were analysed by the first author to identify how the different groups collaborated, what coding platforms they used, how they communicated with their team members, and what patterns of collaborative behaviour emerged.

Second, in order to comprehend the problems faced by the participants while trying to collaboratively program in the hybrid mode, the first author along with the research assistant jointly analysed the problems identified by all the groups. All problems expressed by participants in post-it notes during the activities or mentioned during the final presentation, regardless of the perceived importance of the problems to the participants — were collated in an affinity diagram [4] to identify the main patterns of challenges generated. The researchers also referred to the transcripts of the problems described by the groups at the end of the session presentation to use as anecdotal evidence to guide this process.

Next, the researchers employed thematic analysis [7] to analyse the designs produced by the participants. Both the first author and the research assistant identified all the distinctive features for each of the design diagram produced by the three groups. Having two coders to identify the design elements in these diagrams helped to eliminate bias in interpretation of the elements identified. An agreement rate of 94% was obtained between the two coders after this phase of identification. The agreement level was computed following the equation proposed by Miles et al. [39]: *agreement level = no. of agreements / (total no. of agreements + disagreement).* A total of 49 design elements were identified from all the diagrams, out of which 8 design elements did not reflect any collaborative or learning related aspect. Therefore, the two coders together coded the remaining 41 design elements. We employed inductive coding [8] approach to comprehend and highlight the interaction paradigms that learners sought in their workspace to navigate their collaborative learning processes in a hybrid collaborative setting. The resulting codes were jointly discussed by the codes and arranged into relevant themes.

Finally, qualitative feedback from the design evaluation forms completed by the participants at the end of the design workshop were used as anecdotal evidence to identify design features suggested by others as useful or not.

5 FINDINGS

Below we present the outcome of the participatory design workshop[13]. We first describe the observations of collaborative problem solving behaviour in the workshop. Next, we present the analysis of the challenges expressed by the participants. Finally, we present the designs produced by the participants and its analysis. To report our findings we refer to each participant by their group number and participant number as specified in Table. 1, e.g. G1P1 represents Group G1's participant P1.

5.1 Programming Collaboratively In A Hybrid Setting

Our observations revealed the different means by which participants attempted to collaborate on programming in the hybrid setup, and how their work dynamics unfolded. The observations revealed while all groups used Jupyter Notebooks for executing code, they also used other non-programming platforms to collaboratively write code and communicate with their respective remote team members. Participants in each group used `WhatsApp` with remote members to communicate about the task at hand. While group G2 did this only using text chat, and the other groups communicated via audio calls. Two groups, G1 and G3, used `Google Docs` alongside Jupyter Notebooks to support their collaborative code-writing process. While discussing over their `WhatsApp` calls the remote and co-located members of these two groups would simultaneously try to write the code in `Google Docs` and then test it in their respective Jupyter Notebooks. In the other group G2, the team members did not try to code synchronously. Instead, the remote member tried to do the code individually at their respective locations, while the co-located members engaged in spontaneous conversations and tried to solve the problem together. The remote and co-located members would share their codes and discuss them via a chat on `WhatsApp`.

These observations reflect that learners seek to co-construct ideas and solutions with their team members in a hybrid collaborative context when programming in group. However their collective learning processes often find little support within their chosen programming platform. As a result, they tend to distribute their transactive learning processes across different resources, tools and media situated outside their coding environment. While this allows learners to temporarily navigate their collaborative interactions with other members, they however face a number of obstacles in their working process.

5.2 Challenges Faced In Hybrid Collaboration For Programming

(1) **Challenges to collaborate on code.** From the *Reflection on Problem* activity, the most common complaint among participants was the inability to share their code and work together on it with their team members. All the groups expressed it was particularly frustrating not being able to synchronously work on the code and have no real-time knowledge of what the others were changing in the code. G3P8, *"We could not see 'what*

[13]Details of analysis and findings are available in Supp. 5.

happens on Jupyter' when somebody tries the code." Group G1 expressed that not having the provision to work synchronously on their code, led them to use `Google Docs`. In this regard, group G2 participants also stated that they found it difficult to comprehend, coordinate and collaborate as a collective team. G2P6, *"we had difficulty interacting as a team together."* While group G2 tried to find ways to share code (e. g., through `WhatsApp` chat), they also conveyed that having to figure out means to coordinate on the code effectively distracted them from focusing on solving the programming problem. G2P5, *"We wanted to share code using* `WhatsApp` *and it was difficult because it was distracting. We had to figure out how to share code and not be able to concentrate on the real problem we had to solve."* All the groups resorted to sharing code via `Google Docs` or `WhatsApp`; however, they did not find it to be the most efficient method as these platforms did not preserve the code format. G1P1, *"...so we use* `Google Docs` *to code, but we did not have the correct formatting like Jupyter, and even we can't execute the code."*

(2) **Challenges to communicate with team-members.** Participants in two groups expressed there was a lack of flexibility in a communication medium for engaging into discourse about solving the given problem as a team. As observed earlier, all groups tried to use different communication strategies either via call or text messages on `WhatsApp` to converse with their members. However groups G1 and G3 expressed this to be a limiting experience, as they were not able to share screen with their teammates. G1P2, *"And we also used* `WhatsApp`*, but we discovered bugs. We don't have enough flexibility to communicate, for example we can't share screens with others."*

These challenges expressed by participants highlight that when attempting to solve a programming problem with peers in a hybrid mode, the lack of a shared workspace inhibits learners' awareness of other members' actions on the shared task. This prevents them from coordinating effectively on the programming problem. In addition, figuring out for themselves how to orchestrate these collaborative interactions distributed across different platforms poses further difficulties for their learning process and collaboration.

5.3 Participatory Designs

A total of nine design diagrams were produced by the three groups at the end of participatory design session. Out of these, group G1 created two, group G2 created four and group G3 created three diagrams respectively. The coding process in our thematic analysis yielded 19 unique codes which were arranged into four main themes which we describe below. To report the various design elements from the different group

(1) **Mediating collaborative interactions with peers.** The design elements created by the participants reflected how learners seek to mediate collaborative interactions in hybrid modes for doing programming together with their peers. The most prevalent feature that all the groups depicted in their designs was that of ***synchronous collaboration in shared coding workspace***. In order to coordinate with co-located and remote members, participants wanted a shared view of the code editor with the ability to perform real-time synchronous actions on it together. This is depicted in the design elements of G1-A, G2-A, G3-A in Fig. 3. In their design presentations, the groups also drew parallels between their design elements and `Google Docs` and expressed a preference for their coding platform to have real-time synchronised editing options. This was particularly deemed useful for working on group projects with other members. G1P3, *"For projects, you can activate the real-time synchronisation option for a file and work together on it."* Pertaining to this feature of shared coding workspace, designs from each group also reflected the necessity for fostering ***awareness of activities of collaborators***. For example, element G2-A in Fig. 3 shows the cursor position and the associated coder's name. It demonstrates *visibility* (i.e. witness that an action is occurring) and *transparency* (i.e. know exactly what action is occurring) [53] of other collaborators' action on the shared code. Furthermore, another component of the same design deemed useful by other participants indicated supporting group awareness through a shared history feature. G2P4, *"(in the platform) you also could see the history of the previous versions of the code and who wrote what."* Other design elements also reflected this requirement for a collective notion of code but did not necessarily act synchronously on it. Like in G1-C, the design depicted sharing code excerpts with connected peers through a chat medium, and as explained by G1P1, *"... we can share the code and allow people to come over and copy, modify or execute the shared code."* Design elements from G1 also involved sharing screens to share code.

(2) **Communication with peers.** Collaborative interactions with peers entail learners being able to exchange and negotiate ideas through mutual discussions, enable awareness of each other's actions and provide context and meaning to their individual and collective learning goals. Therefore, to collaborate productively and co-construct a shared notion of code, another critical design element frequented in the participatory designs was fostering flexible communication strategies between collaborators. Each group's design included a conversational component in their programming workspace (e.g., G1-B, G2-B and G3-B in Fig. 3). A consistent idea reflected in all groups' design was ***integrating communication channel within the programming environment*** so that learners do not need to distribute their attention across different platforms elsewhere to communicate. G1P1 specified, *"If there is a system to call each other on the platform, then we do not need to use* `WhatsApp` *or* `Discord`*."* Although the modes of conversation across the designs varied, the text-based chat feature repeated across each group's designs. Interestingly, these designs revealed that the text chat-based platforms should not only serve as a conversation platform but also allow users to share editable code snippets with their peers and classroom teaching assistants (element G1-C in Fig. 3). Groups G1 and G3, along with text chats, also incorporated other conversational features of audio or video options. Additionally, group G1 also included a screen-share element for communicating their work. Furthermore, from the designs, it is also evident that participants want these communication facilities to ***enable both private and group conversations***. One of the designs from Group 3 focused on facilitating this communication across

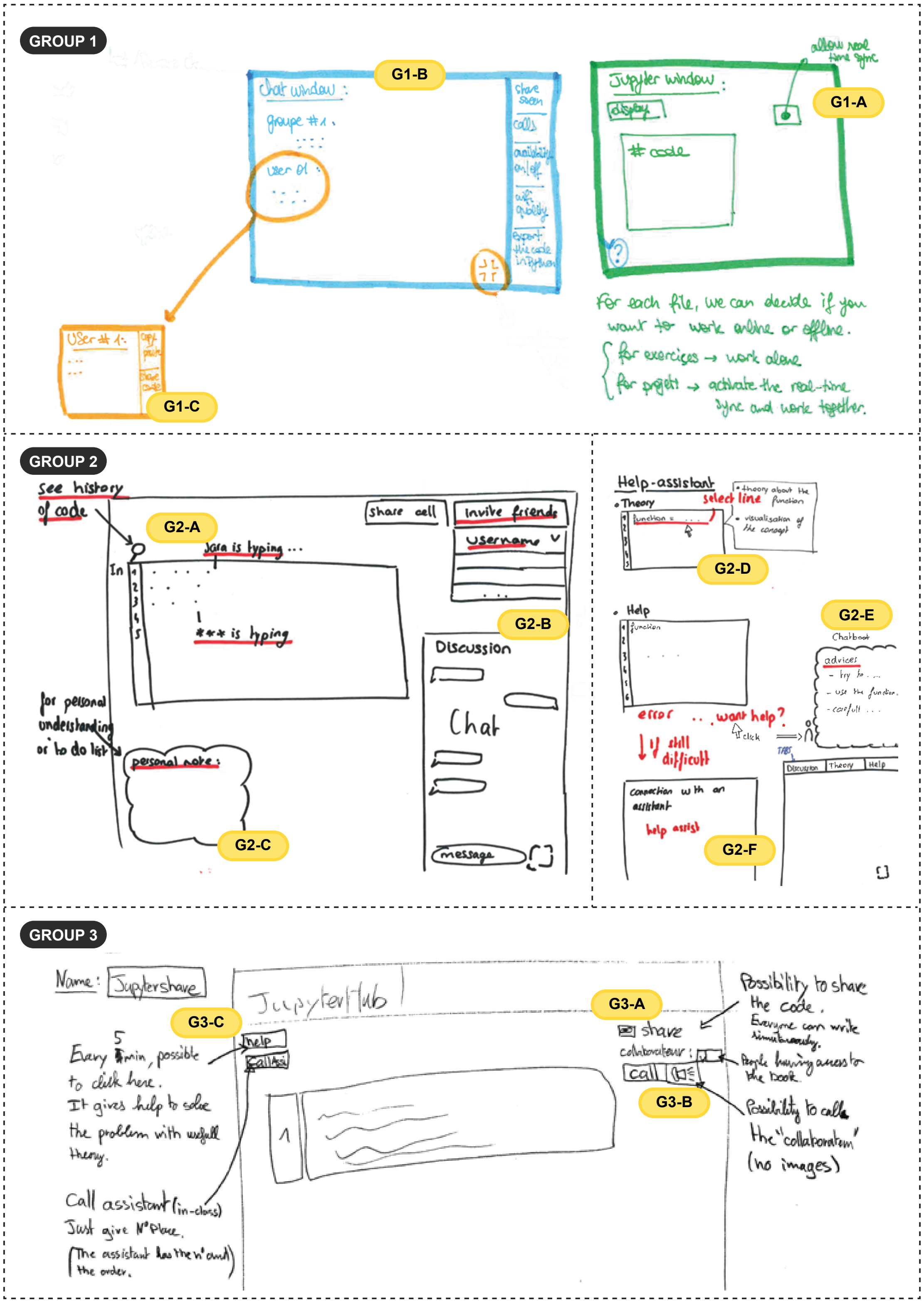

Figure 3: This figure presents examples of participatory design elements produced by the three groups from four of their diagrams. The full design data from all the nine designs are available in Supp. 4.

several groups—-having the option to form or be a part of different groups in addition to the one with which they were assigned to work.

(3) **Control levels of access in the workspace.** The design elements for collaborating in hybrid modes highlighted that along with the co-constructing shared notion of their task and work, participants need ***autonomous control over shared workspace access***. This is reflected in the design by G1, where they devised the real-time synchronous sharing option to be managed based on the type of programming activity they are engaged in. As mentioned during their presentation, G2P6, *"About real-time collaboration, we want to be able to choose whether we want to do our exercise personally, or share work or share screen together in a project."* In addition, the designs revealed a preference among the participants for the ***ability to control access to the shared workspace***. This degree of access varied across the designs produced by the three groups. As seen, group G1 bound their sharing and editing option to code excerpts, i.e. code-cells of Jupyter Notebooks. In contrast, group G3 wanted to share the whole document with their collaborators to edit concurrently. Group G2 wanted to be able to share the whole file as well as code excerpts. The control over sharing workspace is not only applicable to its level of access, but also applies to ***choosing collaborators*** with whom it will be shared. All the designs included component to share their codes or documents with their friends, specifically the group 3 emphasised that group-forming features in programming environments should not be restrict users to one group, but have the provision to be part of multiple groups during. G3P9, *"it will be nice to create a group of everybody...have several groups, one for a project, maybe one or two from the class."* Another aspect of the participatory designs also explicitly highlighted that participants wanted to define a ***personal boundary in the workspace***. This was observed in the designs of both groups 2 and 3. Both the designs included a feature for personal note-taking or learning supportive space to support their individual learning processes. G2P4, *"We could also have little personal bubbles, where you could put notes for your own personal understanding."*

(4) **Supporting learning regulation.** In the context of programming problem solving, along with fostering learners collaborative dynamics, participants sought features to regulate their learning and problem solving process embedded in their programming environment. Two of the design diagrams included features to provide ***progress enabling hints*** to help learners solve the programming problems. For example, in Fig. 3, for the element G2-E, participants designed a chat-bot feature to provide them with hints when they are stuck on an impasse in the code. G2P4, *"We can have a chat-bot, like a programmed assistant who can give us advice when we have an error in the code."* In a similar design G3-C Fig. 3 also wanted progress-enabling hints every five minutes with relevant theories related to the problem for them to progress in their problem-solving process. This feature of dietetic references to theory of the problem being solved is also seen in the design element G2-D (Fig. 3). Furthermore, along with these automated hint features, design elements also described ***features to solicit help from experts***, like teaching assistants (TA). The design element of G2-D, depicted a feature to chat and share code with a TA when they are unable to progress in a problem despite using hints. *G2P5, "If it is really difficult, we can connect to a real TA, and share our codes so that they can help."* In addition to these help-seeking features, participants also described features for eliciting feedback on whether a problem or the overall exercise had been solved, and a personal workspace in the environment to support their individual learning and understanding (e.g., design element G2-C in Fig. 3). This feature of personal note taking was specifically deemed as very useful during the design evaluation. As one participant mentioned, *"Personal note is a great idea, because I often use comments on code and on the long term it is very difficult to read the code."* It indicates that in order to self-regulate their programming learning process learners need affordance to better visualise their learning.

6 DISCUSSION

Our findings offer insights into novice learners' behaviours who collaborate in hybrid modes, and the supports they envision in their programming workspace to coordinate, communicate and work together. In the following, we discuss these findings, and provide design considerations for future design of collaborative programming workspaces in hybrid modes.

6.1 Regulating hybrid collaboration in programming environments

In the context of a classroom, programming learning and problem-solving is not isolated activity but is influenced by the dynamic interplay between peers, tasks, and contexts. Given a hybrid learning environment, our findings suggest that learners engaged in programming classroom activities and group projects situate their problem-solving regulation within this hybrid social context. As learners collaboratively try to solve a programming problem in this setting, we see that they seek to engage in transactive interactions with their co-located or remote peers through their programming workspace. The transactive nature of the interactions is reflected when individual reasoning in a group task builds on and refers to the reasoning shared by the group [57]. Through these interactions learners build collective understanding and coordination of the programming task they are solving. In addition, learners' preferred designs of in-situ resources within programming workspaces reflect a desire to be supported in their transactive interactions of the group problem-solving process in hybrid modes. For example, they want to share codes with their co-located or remote peers in varying degrees, undertake collaborative action on code, and maintain discourse about the problem-solving process. From the lens of contemporary educational psychology, this suggests that learners want to engage in socially shared regulation of programming learning in order to solve the problem they are working on together. In the theory of social modes of learning regulation during collaboration, Hadwin et al. [26] defines *socially shared regulation of learning* as interdependent processes by which group members working towards a co-constructed or shared outcome regulate their collective activity. Informed by individual goals, this regulation involves collective goal setting, monitoring, and adaptation of learning processes subject to the social context. As such, programming workspaces for novices should facilitate this socially shared regulation to unfold in a hybrid mode.

Furthermore, to work on a task as a group demonstrating socially shared regulation of learning requires individuals to self-regulate their processes while also guiding, supporting, and regulating as a collective social entity [26, 29]. Interestingly, our participatory design themes also suggest that learners in the hybrid setting sought to self-regulate and progress in their collaborative programming problem-solving process. Self-regulation in programming problem-solving involves planning and evaluating progress towards solving a computational problem in an iterative way [36]. As such, to solve programming problems together as a team, individuals in groups must contribute to collaborative coding by regulating their individual comprehension of the problem and writing codes. This requires learners to grasp programming concepts, articulate programming skills, and be able to debug and solve errors in their coding process, which can often be challenging for novices. Supporting learners' programming understanding and facilitating their problem-solving process becomes critical to their programming regulation. In this context, integration of coding assistants like `OpenAI Codex`[14] that use generative Large Language Models (LLM), as demonstrated in a recent study by Kazemitabaar et al. [30] shows potential to support novices' programming learning processes. While these LLM assisted coding tools in programming environments may play a role in assisting learners' self-regulation, its role in facilitating programming collaboration between peers still needs to be explored. Particularly, it is not clear whether how group interactions with a mix of human and intelligent coding assistant might socially regulate. In summary, to facilitate students' collaborative programming-solving processes in hybrid classrooms, it is imperative to provide learners with affordances that enable and sustain their socially shared and self-regulation of programming learning.

6.2 Design considerations for collaboration in programming environments

Based on our observations and participatory design workshop findings, we see learners prefer features integrated in their programming workspaces to collaboratively program in hybrid setting. Hence, we present a set of design considerations and implications for the design of programming environments for novices in hybrid-taught programming classrooms. Although collaboration in hybrid classrooms may include other types of media, such as online forums and video-conferencing systems, our design reflections specifically focus on the programming environment.

(1) **Enabling virtual mode of collaborative task interactions.** In programming problem solving, writing code to solve the task is mechanised within the framework of a programming environment. Furthermore, in order for students to regulate collective coding, they also need to be able to code individually, so learners prefer to have their individual instance of programming environment. As the learners' behaviour in our observations of collaborative problem-solving in a hybrid setup shows, the co-located members, along with the remote members, also prefer to collaborate on the code via an online programming environment so that the code is accessible to all, regardless of their location. This suggests that collaborative interactions on programming tasks in a hybrid learning environment tend to unfold and be driven by interactions occurring in a virtual space, which is embedded in the programming workspace. In contrast to hybrid collaborations in other workspaces [41] unrelated to programming education, the collaborative enactment of the main task for programming is not distributed across physical and virtual spaces and is rather solely virtual. Programming tasks are collaboratively solved by remote interactions on the task, even when individuals are in the same physical location. It implies that in order to facilitate hybrid collaboration in programming, the design of the workspace should include a virtual component for the programming task and have features created explicitly for collaboration in this virtual space. Bidirectional code synchronisation, awareness of different collaborators' contributions and providing a virtual space are supported by versioning tools such as `GitHub` in programming environments. However, as the study by Ying et al. [62] shows, for novice programmers, the learning curve for using `GitHub` is steep, and working with it is a rather complex process, as students are anxious to ruin the whole project. As such, future programming environments may design simpler version control systems for projects or specific coding activities — taking into account the goals and needs of novice programmers. It should be noted, however, that this might not guarantee that inter-personal interactions between co-located members will entirely be virtual; for example, co-located users may still use gestural references or verbal communication in person, while their task enactment could be applied to the shared workspace.

(2) **Tools for shared workspace awareness**. For collaborative programming problem-solving in hybrid modes, it necessitates programming environments to act not only as a cognitive tool but also as an agent of inter-personal social interactions between collaborators to build knowledge together. Therefore, when designing programming environments, collaborators need to be equipped with means and mechanisms to engage in transactive interactions. Social tools or features in programming environments for hybrid collaboration identified in the participatory designs, include sharing editable snippets of code, allowing collaborators to access a programming file or snippet of code for asynchronous editing, and enabling real-time collaborative editing of shared code and files. In addition, to provide context and awareness for coordinated actions, participants also emphasised distinguishing and recording the contributions of different collaborators through shared history, name tags, and positional references in the code. In particular, this last feature fosters *workspace awareness* by supporting deictic references in the oral/written exchanges of the collaborators[15]. According to previous research, these features for workspace awareness are

[14]https://openai.com/blog/openai-codex, last accessed May 2023.

[15]A deictic reference is the use of gestures or expressions in language (e.g., *this, that, these*) that points to the time, place, or situation in which a speaker is speaking. For example, two remote collaborators working synchronously on a shared document can use cursor movements to direct attention to specific elements in the editor. Here the cursor (or *telepointer*) acts as a deictic reference [12].

critical in the design of distributed educational groupware [25]. Furthermore, enabling discourse development through conversational channels is crucial to a hybrid collaboration, as they provide a way to contextualise this deictic awareness. Especially for remote members, both these deictic references and discussions provide the most thorough presentation of co-located or other remote members' contexts.
These social supports in coding environments can be provided to students leveraging some of the existing solutions in programming integrated development environments (IDEs) for remote collaboration. For example, extensions for live sharing of coding sessions, real-time collaboration (RTC) on code are supported in `Studio Live Share` for Microsoft's Visual Studio, `Code With Me` in JetBrains IDE. Commercial solutions for Notebook-based software allowing RTC functionality include `CoCalc` and Google's `Colab`. Recently open-source solution `JuptyterLab` also incorporated RTC capabilities in their Notebooks. In addition some of these platforms also integrate options for audio, video and text chat through the same or different extensions. Adopting the use of these programming platforms with only the necessary social functionalities enabled based on a particular course's context, can provide a starting point for facilitating collaborative problem solving in hybrid taught classrooms.

(3) **Persistence of collaborative programming workspace**. While enabling social learning interactions through programming platforms for collaborative problem solving, it is also critical that the results of these coordinated actions remain persistent, especially in the context of work-in-progress projects and assignments. As expressed by participants in our design workshop, they envision a persistent, online, mutually accessible collaborative editing and coding space for working on group projects. Students want continuous collaborative access to their project's coding file, where they can code synchronously and have the ability to code asynchronously, with an awareness of the contributions of different collaborators. Previous research has also emphasised the importance of persistent information and tools in designing collaborative workspaces. Dillenbourg et al.[17] suggest that designing persistent collaborative environments helps learners to externalise their group working processes and promotes better understanding of their tasks.
While the available programming IDEs and tools mentioned in the previous point allow simultaneous code editing, synchronous tracking and real-time collaboration, their shared access is dependent on the lifetime of the session. When the sharer terminates a shared session, the results of collaborative changes are stored locally, but are not propagated to or accessible by other collaborators' sessions. Currently, there is very little support in available programming platforms for simultaneous real-time and non-real-time collaboration. The design of this aspect in programming has only been explored in a handful of studies [22, 38], using real-time collaboration and version control systems. However, these systems incorporating version control are often too complex for novice programmers to use. This necessitates the design of persistent collaborative supports in programming environments for novices to help them regulate their programming activities collaboratively.

(4) **Flexibility of workspace territory.** Our findings shed light on the fact that a collaborative programming workspace for novices in hybrid mode also needs to define a personal entity to enable their self-regulation of programming learning along with socially shared regulation. This finding mirrors the aspect of territoriality in hybrid collaboration which unfurls in team projects including spatial aspects [41]. Writing code and debugging programming problems for novices is often a messy and iterative process. This process fundamentally administers students' regulation of programming learning, and a personal space for learners to externalise this regulation further enables this process. Furthermore, in a collaborative environment, these phases of trial and error can be an overwhelming amount of information for others, as well as entail privacy demands for individuals. The commercially available programming tools are individual-centric systems that allow owners to share an instance of their environment, giving them the choice to autonomously control their workspace. However these tools do not incorporate a common programming task enactment space, while also maintaining an individual workspace. In this respect, programming environments should design these personal and group territories within a programming workspace in such a way that there is the possibility of osmosis between the two — learning from each workspace territory can foster the other and vice versa.

7 LIMITATIONS

Our study has some limitations. As the study was conducted after the completion of the course from which we recruited participants, the number of participants who responded and participated was limited. Furthermore, the participation responses to the study invitation were received from students who had access to Jupyter Notebook plug-in during the course. Finally, the selection of our sample may ignore those who encounter challenges in hybrid learning and end up dropping the course.

8 CONCLUSION

Our work aims to inform the design of programming workspaces for novices to facilitate effective collaboration on programming problems in hybrid modes of learning. We employed a user-centred approach and conducted a participatory design study with nine students from a hybrid-taught introductory programming course. Our study intended to identify the challenges faced by novices and gain an understanding of their perspectives on the designs of collaborative programming workspaces to support their collaborative work processes. Our findings reflect that to effectively collaborate when programming in hybrid environments, learners seek to mediate their interactions with peers to build a shared understanding of their task, provide context, and maintain this shared understanding through communication. Furthermore, they prefer to be able to autonomously control the sharing of their workspace and regulate their own learning within it. Based on these findings, we present a set of design principles for pedagogical tools in hybrid programming learning settings. We hope that this study paves the road to design of more effective tools for teaching programming.

ACKNOWLEDGMENTS

We sincerely thank Sruti Bhattacharjee for her assistance in designing and facilitating the participatory design workshop, as well as helping in analysing the data. We also thank James Tyler and Kavous Salehzadeh Niksirat for taking part in the dry run of the participatory design workshop and providing helpful feedback. Finally, we are grateful to the anonymous reviewers for their constructive and insightful comments.

REFERENCES

[1] Ali Alammary. 2019. Blended learning models for introductory programming courses: A systematic review. *PLOS ONE* 14, 9 (Sept. 2019), 1–26. https://doi.org/10.1371/journal.pone.0221765

[2] Nikolaos Avouris, Stefanos Kaxiras, Odysseas Koufopavlou, Kyriakos Sgarbas, and Polyxeni Stathopoulou. 2010. Teaching introduction to computing through a project-based collaborative learning approach. In *2010 14th Panhellenic Conference on Informatics*. IEEE, Tripoli, Greece, 237–241. https://doi.org/10.1109/PCI.2010.13

[3] Susan Bergin, Ronan Reilly, and Desmond Traynor. 2005. Examining the Role of Self-Regulated Learning on Introductory Programming Performance. In *Proceedings of the First International Workshop on Computing Education Research* (Seattle, WA, USA) *(ICER '05)*. Association for Computing Machinery, New York, NY, USA, 81–86. https://doi.org/10.1145/1089786.1089794

[4] Hugh Beyer and Karen Holtzblatt. 1999. Contextual design. *interactions* 6, 1 (1999), 32–42.

[5] Marcel Borowski, Bjarke V. Fog, Carla F. Griggio, James R. Eagan, and Clemens N. Klokmose. 2022. Between Principle and Pragmatism: Reflections on Prototyping Computational Media with Webstrates. *ACM Trans. Comput.-Hum. Interact.* 30, 2 (oct 2022). https://doi.org/10.1145/3569895

[6] Kristy Elizabeth Boyer, August A. Dwight, R. Taylor Fondren, Mladen A. Vouk, and James C. Lester. 2008. A Development Environment for Distributed Synchronous Collaborative Programming. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain) *(ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 158–162. https://doi.org/10.1145/1384271.1384315

[7] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[8] Virginia Braun and Victoria Clarke. 2020. One size fits all? What counts as quality practice in (reflexive) thematic analysis? *Qualitative Research in Psychology* 18, 3 (Aug. 2020), 1–25. https://doi.org/10.1080/14780887.2020.1769238

[9] Crescencio Bravo, Rafael Duque, and Jesús Gallardo. 2013. A groupware system to support collaborative programming: Design and experiences. *Journal of Systems and Software* 86, 7 (2013), 1759–1771. https://doi.org/10.1016/j.jss.2012.08.039

[10] Kelly Burke, Kregg Aytes, Laku Chidambaram, and Jeffrey Johnson. 1999. A Study of Partially Distributed Work Groups: The Impact of Media, Location, and Time on Perceptions and Performance. *Small Group Research - SMALL GROUP RES* 30 (08 1999), 453–490. https://doi.org/10.1177/104649649903000404

[11] Jennifer Campbell, Diane Horton, Michelle Craig, and Paul Gries. 2014. Evaluating an Inverted CS1. In *SIGCSE '14: Proceedings of the 45th ACM technical symposium on Computer science education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 307–312. https://doi.org/10.1145/2538862.2538943

[12] Mauro Cherubini, Marc-Antoine Nüssli, and Pierre Dillenbourg. 2008. Deixis and Gaze in Collaborative Work at a Distance (over a Shared Map): A Computational Model to Detect Misunderstandings. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications* (Savannah, Georgia) *(ETRA '08)*. Association for Computing Machinery, New York, NY, USA, 173–180. https://doi.org/10.1145/1344471.1344515

[13] Arik Cheshin, Yongsuk Kim, D. Bos Nathan, Nan Ning, and Judith S. Olson. 2013. Emergence of differing electronic communication norms within partially distributed teams. *Journal of Personnel Psychology* 12 (2013), 7–21. https://doi.org/10.1027/1866-5888/a000076 Place: Germany Publisher: Hogrefe Publishing.

[14] Merijke Coenraad, Jen Palmer, Donna Eatinger, David Weintrop, and Diana Franklin. 2022. Using participatory design to integrate stakeholder voices in the creation of a culturally relevant computing curriculum. *International Journal of Child-Computer Interaction* 31 (2022), 100353. https://doi.org/10.1016/j.ijcci.2021.100353

[15] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. CodeWrite: Supporting Student-Driven Practice of Java. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 471–476. https://doi.org/10.1145/1953163.1953299

[16] Pierre Dillenbourg, Sanna Järvelä, and Frank Fischer. 2009. The Evolution of Research on Computer-Supported Collaborative Learning. In *Technology-Enhanced Learning*, Nicolas Balacheff, Sten Ludvigsen, Ton de Jong, Ard Lazonder, and Sally Barnes (Eds.). Springer Netherlands, Dordrecht, 3–19. https://doi.org/10.1007/978-1-4020-9827-7_1

[17] Pierre Dillenbourg and David Traum. 2006. Sharing Solutions: Persistence and Grounding in Multimodal Collaborative Problem Solving. *Journal of the Learning Sciences* 15, 1 (2006), 121–151. https://doi.org/10.1207/s15327809jls1501_9

[18] Benedict Du Boulay. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73.

[19] Leovy Echeverría, Ruth Cobos, Liliana Machuca, and Ivan Claros. 2017. Using collaborative learning scenarios to teach programming to non-CS majors. *Computer applications in engineering education* 25, 5 (2017), 719–731.

[20] K. Anders Ericsson, Ralf T. Krampe, and Clemens Tesch-Römer. 1993. The role of deliberate practice in the acquisition of expert performance. *Psychological Review* 100 (1993), 363–406. https://doi.org/10.1037/0033-295X.100.3.363 Place: US Publisher: American Psychological Association.

[21] Katrina Falkner, Rebecca Vivian, and Nickolas J.G. Falkner. 2014. Identifying Computer Science Self-Regulated Learning Strategies. In *Proceedings of the 2014 Conference on Innovation &; Technology in Computer Science Education* (Uppsala, Sweden) *(ITiCSE '14)*. Association for Computing Machinery, New York, NY, USA, 291–296. https://doi.org/10.1145/2591708.2591715

[22] Hongfei Fan, Chengzheng Sun, and Haifeng Shen. 2012. ATCoPE: Any-Time Collaborative Programming Environment for Seamless Integration of Real-Time and Non-Real-Time Teamwork in Software Development. In *Proceedings of the 2012 ACM International Conference on Supporting Group Work* (Sanibel Island, Florida, USA) *(GROUP '12)*. Association for Computing Machinery, New York, NY, USA, 107–116. https://doi.org/10.1145/2389176.2389194

[23] Dilrukshi Gamage, Indika Perera, and Shantha Fernando. 2020. MOOCs Lack Interactivity and Collaborativeness: Evaluating MOOC Platforms. *International Journal of Engineering Pedagogy (iJEP)* 10, 2 (March 2020), 94–111. https://doi.org/10.3991/ijep.v10i2.11886 Number: 2.

[24] Thomas D. Griffin, Jennifer Wiley, and Carlos R. Salas. 2013. *Supporting Effective Self-Regulated Learning: The Critical Role of Monitoring.* Springer New York, New York, NY, 19–34. https://doi.org/10.1007/978-1-4419-5546-3_2

[25] Carl Gutwin, Gwen Stark, and Saul Greenberg. 1995. Support for Workspace Awareness in Educational Groupware. In *The First International Conference on Computer Support for Collaborative Learning* (Indiana Univ., Bloomington, Indiana, USA) *(CSCL '95)*. L. Erlbaum Associates Inc., USA, 147–156. https://doi.org/10.3115/222020.222126

[26] Allyson Hadwin, Sanna Järvelä, and Mariel Miller. 2017. Self-Regulation, Co-Regulation, and Shared Regulation in Collaborative Learning Environments. In *Handbook of Self-Regulation of Learning and Performance* (2 ed.), Dale H. Schunk and Jeffrey A. Greene (Eds.). Routledge, New York, 83–106. https://doi.org/10.4324/9781315697048-6

[27] Rashina Hoda, Annette Henderson, Shiree Lee, Bridget Beh, and Jason Greenwood. 2014. Aligning technological and pedagogical considerations: Harnessing touch-technology to enhance opportunities for collaborative gameplay and reciprocal teaching in NZ early education. *International Journal of Child-Computer Interaction* 2, 1 (2014), 48–59. https://doi.org/10.1016/j.ijcci.2014.06.001

[28] Tony Jenkins. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, Vol. 4. LTSN-ICS, Loughborough, 53–58.

[29] Sanna Järvelä, Paul Kirschner, Ernesto Panadero, Jonna Malmberg, C. Phielix, Jos Jaspers, Marika Koivuniemi, and Hanna Järvenoja. 2015. Enhancing socially shared regulation in collaborative learning groups: Designing for CSCL regulation tools. *Educational Technology Research and Development* 63 (02 2015), 125–142. https://doi.org/10.1007/s11423-014-9358-1

[30] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. https://doi.org/10.1145/3544548.3580919

[31] Finn Kensing and Andreas Munk-Madsen. 1993. PD: Structure in the Toolbox. *Commun. ACM* 36, 6 (jun 1993), 78–85. https://doi.org/10.1145/153571.163278

[32] Patricia Lasserre and Carolyn Szostak. 2011. Effects of Team-Based Learning on a CS1 Course. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (Darmstadt, Germany) *(ITiCSE '11)*. Association for Computing Machinery, New York, NY, USA, 133–137. https://doi.org/10.1145/1999747.1999787

[33] Celine Latulipe, N. Bruce Long, and Carlos E. Seminario. 2015. Structuring Flipped Classes with Lightweight Teams and Gamification. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 392–397. https://doi.org/10.1145/2676723.2677240

[34] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. Ethnography. In *Research Methods in Human Computer Interaction (Second Edition)* (second edition ed.), Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser (Eds.).

Morgan Kaufmann, Boston, 229–261. https://doi.org/10.1016/B978-0-12-805390-4.00009-1

[35] Ally Limke, Nicholas Lytle, Maggie Lin, Sana Mahmoud, Marnie Hill, Veronica Cateté, and Tiffany Barnes. 2023. Empowering Students as Leaders of Co-Design for Block-Based Programming. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, Article 98, 7 pages. https://doi.org/10.1145/3544549.3585775

[36] Dastyni Loksa and Amy J. Ko. 2016. The Role of Self-Regulation in Programming Problem Solving Process and Success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. Association for Computing Machinery, New York, NY, USA, 83–91. https://doi.org/10.1145/2960310.2960334

[37] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion)*. Association for Computing Machinery, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[38] Yifan Ma, Batu Qi, Wenhua Xu, Mingjie Wang, Bowen Du, and Hongfei Fan. 2022. Integrating Real-Time and Non-Real-Time Collaborative Programming: Workflow, Techniques, and Prototypes. *Proc. ACM Hum.-Comput. Interact.* 7, GROUP, Article 13 (dec 2022), 19 pages. https://doi.org/10.1145/3567563

[39] Matthew B Miles, A Michael Huberman, and Johnny Saldaña. 2019. Qualitative data analysis: A methods sourcebook. https://us.sagepub.com/en-us/nam/qualitative-data-analysis/book246128

[40] Tanya Nazaretsky, Carmel Bar, Michal Walter, and Giora Alexandron. 2022. Empowering Teachers with AI: Co-Designing a Learning Analytics Tool for Personalized Instruction in the Science Classroom. In *LAK22: 12th International Learning Analytics and Knowledge Conference* (Online, USA) *(LAK22)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3506860.3506861

[41] Thomas Neumayr, Mirjam Augstein, and Bettina Kubicek. 2022. Territoriality in Hybrid Collaboration. *Proc. ACM Hum.-Comput. Interact.* 6, CSCW2, Article 332 (nov 2022), 37 pages. https://doi.org/10.1145/3555224

[42] Thomas Neumayr, Hans-Christian Jetter, Mirjam Augstein, Judith Friedl, and Thomas Luger. 2018. Domino: A Descriptive Framework for Hybrid Collaboration and Coupling Styles in Partially Distributed Teams. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 128 (nov 2018), 24 pages. https://doi.org/10.1145/3274397

[43] Meike Osinski and Nikol Rummel. 2019. Towards Successful Knowledge Integration in Online Collaboration: An Experiment on the Role of Meta-Knowledge. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 31 (nov 2019), 17 pages. https://doi.org/10.1145/3359133

[44] David Preston. 2005. PAIR Programming as a Model of Collaborative Learning: A Review of the Research. *J. Comput. Sci. Coll.* 20, 4 (apr 2005), 39–45.

[45] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (oct 2017), 24 pages. https://doi.org/10.1145/3077618

[46] Shanon Reckinger and Bryce Hughes. 2020. Strategies for Implementing In-Class, Active, Programming Assessments: A Multi-Level Model. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 454–460. https://doi.org/10.1145/3328778.3366850

[47] Teodoro F. Revano and Manuel B. Garcia. 2021. Designing Human-Centered Learning Analytics Dashboard for Higher Education Using a Participatory Design Approach. In *2021 IEEE 13th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*. IEEE, Manila, Philippines, 1–5. https://doi.org/10.1109/HNICEM54116.2021.9731917

[48] María Rodríguez-Triana, Alejandra Martínez-Monés, Juan Asensio-Pérez, and Yannis Dimitriadis. 2012. Towards a Monitoring-Aware Design Process for CSCL Scripts. In *Collaboration and Technology*, Vol. 7493. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33284-5_21

[49] Jeremy Roschelle and Stephanie D. Teasley. 1995. The Construction of Shared Knowledge in Collaborative Problem Solving. In *Computer Supported Collaborative Learning (NATO ASI Series)*, Claire O'Malley (Ed.). Springer, Berlin, Heidelberg, 69–97. https://doi.org/10.1007/978-3-642-85098-1_5

[50] Stephan Salinger, Christopher Oezbek, Karl Beecher, and Julia Schenk. 2010. Saros: An Eclipse Plug-in for Distributed Party Programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering* (Cape Town, South Africa) *(CHASE '10)*. Association for Computing Machinery, New York, NY, USA, 48–55. https://doi.org/10.1145/1833310.1833319

[51] Gavriel Salomon. 1992. New Challenges for Educational Research: studying the individual within learning environments. *Scandinavian Journal of Educational Research* 36, 3 (1992), 167–182. https://doi.org/10.1080/0031383920360301

[52] Elizabeth B.-N. Sanders and Pieter Jan Stappers. 2008. Co-creation and the new landscapes of design. *CoDesign* 4, 1 (2008), 5–18. https://doi.org/10.1080/15710880701875068

[53] Stacey D. Scott, M. Sheelagh T. Carpendale, and Kori Inkpen. 2004. Territoriality in Collaborative Tabletop Workspaces. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work* (Chicago, Illinois, USA) *(CSCW '04)*. Association for Computing Machinery, New York, NY, USA, 294–303. https://doi.org/10.1145/1031607.1031655

[54] Judy Sheard, S. Simon, Margaret Hamilton, and Jan Lönnberg. 2009. Analysis of Research into the Teaching and Learning of Programming. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop* (Berkeley, CA, USA) *(ICER '09)*. Association for Computing Machinery, New York, NY, USA, 93–104. https://doi.org/10.1145/1584322.1584334

[55] Joanna Smith, Joe Tessler, Elliot Kramer, and Calvin Lin. 2012. Using Peer Review to Teach Software Testing. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (Auckland, New Zealand) *(ICER '12)*. Association for Computing Machinery, New York, NY, USA, 93–98. https://doi.org/10.1145/2361276.2361295

[56] Angela E.B. Stewart, Hana Vrzakova, Chen Sun, Jade Yonehiro, Cathlyn Adele Stone, Nicholas D. Duran, Valerie Shute, and Sidney K. D'Mello. 2019. I Say, You Say, We Say: Using Spoken Language to Model Socio-Cognitive Processes during Computer-Supported Collaborative Problem Solving. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 194 (nov 2019), 19 pages. https://doi.org/10.1145/3359296

[57] Stephanie D. Teasley. 1997. Talking About Reasoning: How Important Is the Peer in Peer Collaboration? In *Discourse, Tools and Reasoning: Essays on Situated Cognition*, Lauren B. Resnick, Roger Säljö, Clotilde Pontecorvo, and Barbara Burge (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 361–384. https://doi.org/10.1007/978-3-662-03362-3_16

[58] April Yi Wang, Yan Chen, John Joon Young Chung, Christopher Brooks, and Steve Oney. 2021. PuzzleMe: Leveraging Peer Assessment for In-Class Programming Exercises. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 415 (oct 2021), 24 pages. https://doi.org/10.1145/3479559

[59] Jeremy Warner and Philip J. Guo. 2017. CodePilot: Scaffolding End-to-End Collaborative Software Development for Novice Programmers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1136–1141. https://doi.org/10.1145/3025453.3025876

[60] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J. Ko. 2019. A theory of instruction for introductory programming skills. *Computer Science Education* 29, 2-3 (2019), 205–253. https://doi.org/10.1080/08993408.2019.1565235

[61] Soonja Yeom, Nicole Herbert, and Riseul Ryu. 2022. Project-Based Collaborative Learning Enhances Students' Programming Performance. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1* (Dublin, Ireland) *(ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 248–254. https://doi.org/10.1145/3502718.3524779

[62] Kimberly Michelle Ying and Kristy Elizabeth Boyer. 2020. Understanding Students' Needs for Better Collaborative Coding Tools. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3334480.3383068

[63] Saijing Zheng, Mary Beth Rosson, Patrick C. Shih, and John M. Carroll. 2015. Designing MOOCs as Interactive Places for Collaborative Learning. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (Vancouver, BC, Canada) *(L@S '15)*. Association for Computing Machinery, New York, NY, USA, 343–346. https://doi.org/10.1145/2724660.2728689