ABSTRACT

Title of Thesis: "I FEEL LIKE I'M TEACHING IN A GLADIATOR RING": BARRIERS AND BENEFITS OF LIVE CODING

Caroline Palma Berger
Master of Science, 2023

Thesis Directed By: Professor Niklas Elmqvist
College of Information Studies

Live coding—synchronously writing software in front of students for the purpose of teaching—can be an effective method for engaging students and instilling practical programming skills. However, not all live coding sessions are effective and not all instructors are successful in this challenging task. We present results from an interview study involving university instructors, teaching assistants, and students identifying both barriers and benefits of live coding. We also designed and collected participant feedback on a prototype live coding tool to better facilitate learner engagement with the live coding pedagogical practice. Finally, we use this feedback to propose guidelines for how to design tools to support effective live coding in the classroom. This work advances our understanding of the benefits and challenges of live coding in university computer science instruction and highlights potential future work on the design of tools to better support this productive instructional practice.

“I FEEL LIKE I’M TEACHING IN A GLADIATOR RING”: BARRIERS AND BENEFITS OF LIVE CODING

by

Caroline Palma Berger

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2023

Advisory Committee:
Professor Niklas Elmqvist, Chair
Professor David Weintrop, Committee member
Professor Joel Chan, Committee member

Dedication

To my two roommates, thanks for listening to me digest my ideas and doing the dishes when I was too consumed with work. This is to you my dear sister and DC partner in crime, Christina and to the love of my life, Ben.

Acknowledgments

Thanks, Niklas, for guiding and motivating me through this process. You taught me to approach academic work from a cool and collected perspective, to trust my gut, and to enjoy academic writing. Beyond the work that went into this thesis, I appreciate your career guidance as I begin down the path to becoming a professor and researcher. Thanks for the hands-on help with tools and processes, and for creating a warm, collaborative research group atmosphere.

David, Joel, thanks for being open-minded and for encouraging academic risk-taking. Professor Eun Kyoung Choe, thank you for your expertly designed course on approaching academic writing and the master's thesis. The notes I took during this class, the discussions, and deadline to get the first two chapters drafted in the fall were key to the timely delivery of this document and my learning through this process. Much appreciation to Saransh Grover for his input in the data analysis and Clemens Klokmose for his inspiration regarding interview methods.

Thanks to all the participants that shared compelling stories about their experiences learning and teaching.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

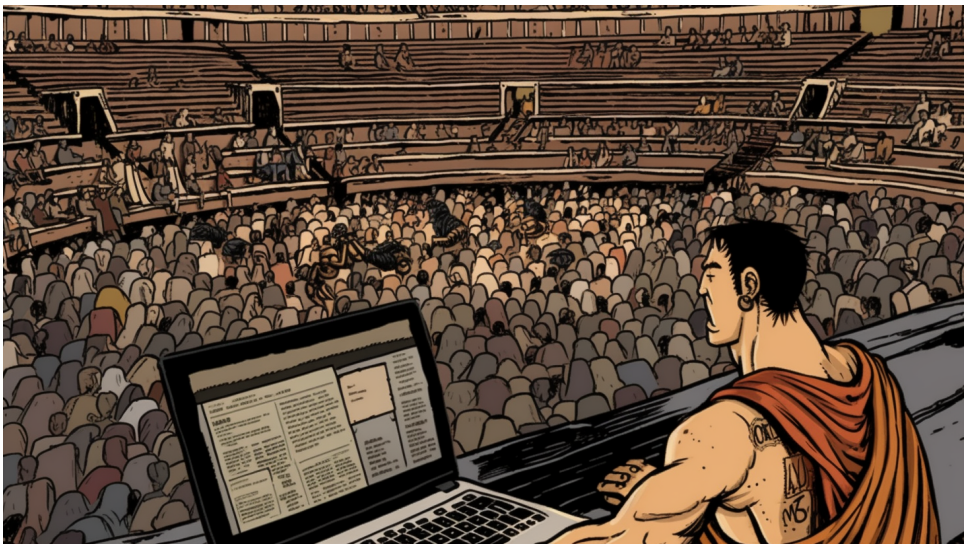

Figure 1.1: **Live coding as a performance.** Our interview study found that many instructors feel live coding to be a high-stakes and cognitively taxing activity more akin to a live performance. (Illustration by MidJourney version 5.)

> *"And then I'm teaching [...] in a classroom that feels like a gladiatorial ring. 200 seats in a wall up in front of me. And I have to lean back to see the top. And really the only constraint in that classroom is that it's terrifying. It is the most terrifying experience I've ever had."* – Participant 08 (Computer Science instructor)

The gladiator descends into his pit engulfed by shrill screams from the spectators. Each level of the arena stares down at him, anticipating the long-awaited battle. But we aren't in Ancient Rome; we are in a giant tiered lecture hall on a present-day university campus. The

spectators are students learning programming. And the gladiator? It's the instructor, of course, armed with only their laptop and some notes, and the feat they are about to attempt is to write running source code in front of an auditorium full of students. While an effective means of engaging the audience and conveying practical programming skills, instructors often liken this form of *live coding* to a performance or—as P08 puts it above—a gladiatorial battle because of its high stakes, captive audience, and technical challenge (Figure 1.1).

Live coding is defined as *"the process of writing code live on a computer in front of students during class"* [1, p. 164]. Live coding models the process of programming [2] and gives students insight into practical programming skills and practices. However, research shows that active engagement is an important component of effective learning [3]. Fostering engagement requires teacher preparation and time [4], and risks increasing workload and burnout [5]. Additional work is needed to gather the perspectives of teachers on active learning during live coding activities [1].

Why are some instructors more effective than others at live coding? What are the barriers and what are the best practices and benefits? And how can tooling support this practice in live and online classrooms? In this thesis, we endeavor to answer these questions and many others by conducting an interview study involving a range of live coding stakeholders: the instructors and teaching assistants who perform these sessions, and the students who participate in them. We conducted semi-structured interviews with single participants as well as groups. During each session, we also solicited feedback on a live coding prototype tool designed to scaffold the practice by enabling students to follow along on their own devices (Figure 3.2), as well as give instructors a high-level overview of student programming activity (Figure 3.1 and Figure 5.1). Our thematic analysis of these sessions enabled us to identify factors that serve as **barriers** against effective live coding, as well as those that **benefit** the practice.

In summary, we found that constraints in the teaching environment, negative feelings, and communication breakdown make live coding difficult. In particular, teachers and students reported intentional and unintentional mistakes, ability to involve the class, and flexibility of scaffolding as benefits of live coding. Based on our findings, we recommend considering how personal computers might detract from live coding lectures and how to direct student attention to important parts of the instructor's code. To support student-teacher collaborative live coding, we recommend tools to support a unified digital space accessed via multiple devices. When a large classroom of students are live coding, errors that students encounter and peeking into student editors may help teachers monitor student progress.

The remainder of this thesis is structured as follows:

Chapter 2 synthesizes related literature.

Chapter 3 describes the prototypes and the methodological approach of interviews and prototype feedback.

Chapter 4 reports on the barriers and benefits of live coding identified by our study.

Chapter 5 proposes design guidance for live coding tools, and discusses limitations and presents areas for future work.

Chapter 6 concludes the work.

We have submitted a manuscript of this work to a computing education conference.

Chapter 2: Related Work

This work is situated at the intersection of education, computer science, and human-computer interaction. Here we examine the current state of live coding tools and practices, and audience engagement tools. After, we introduce cognitive apprenticeship, the theoretical framework that grounds the work.

2.1 Active Learning in Programming Instruction

A significant body of research has investigated the challenges associated with teaching novices how to program (for reviews, see [6] and [7]). This research has attended to motivational and sociocultural aspects [8], pedagogy [9], the cognitive complexity of the topic [10], and the design of programming languages and environments [11]. Across the insights from this literature, active learning has emerged as a productive instructional practice with a variety of instantiations in contemporary computer science classrooms [12]. As opposed to *passive learning* where the teacher conveys information with little student involvement, *active learning* stipulates continuous interaction between students and instructor, supporting the student's engagement in the process of learning [13]. Despite active learning having shown to be effective for STEM (Science, Technology, Engineering, and Mathematics) subjects [14], passive instruction is still common in the teaching of computer programming [15, 16].

Learning to program requires practice with feedback to help the student guide their own learning. This kind of deliberate practice—practice that is coupled with continuous feedback—is vital for acquiring the necessary cognitive, perceptual, and motor skills [17]. Such feedback might come from programming tools, in the form of error messages or syntax highlighting, or in classroom settings from teaching assistants, instructors, or peers. In particular, this necessary feedback can be delivered at a scale to a classroom of students through *live coding*: writing code live on a computer in front of a class [1].

Active learning in computer science includes techniques of flipping the classroom, project-based learning, and peer instruction [12]. Students appreciate active learning in computer science, with reports of increased satisfaction, motivation, and confidence along with improved learning outcomes over traditional techniques [12]. Centering the student in their learning, active learning techniques help students to learn at their own pace. Despite positive reactions and improved educational outcomes of active learning, such techniques require additional effort for instructors when compared to traditional techniques [12]. In this paper, we aim to examine possibilities for active learning in the context of live coding without increasing instructor workload.

2.2 Live Coding Practices

A common technical setup for live coding is for the instructor to simply project their integrated development environment (or share their screen) while they are live coding [18]. Students can then follow along in their local editors. The teacher often talks aloud to provide rationale for the actions taken [1, 18] and encourages the students to ask questions during the process [18]. Some students prefer to code along with the instructor [19], while others do not [20].

A flipped classroom approach where students watch a prerecorded video of an instructor writing code is another flavor of live coding that has shown to be effective for learning programming [21]. Associated with positive learning outcomes, the live-coding videos for learning programming (LV4LP) platform supports self-regulation by providing the ability for students to pause and rewatch portions of the video, and opportunities for students to reflect by annotating timestamps of the video [21]. However, such approaches are not strictly speaking "live" in that they do not allow the instructor to interact directly with the students and respond to their questions.

While there remain open questions as to the specifics of its benefits and limitations relative to other teaching methods, live coding is a promising direction due to positive reactions from teachers and students [1, 22]. It is also a scalable delivery method because of its one-to-many relationship between instructor and students. Importantly, live coding models the incremental process of programming [2] rather than instantly displaying a finished product that, once run, will produce an error-free output. Live coding presents a more realistic picture of the practice of programming which can be important for easing novice students into the craft and complexities of programming.

2.3 Live Coding Tools

Simultaneously writing correct code, describing the code you are writing, and managing the technology used for sharing the source code with the students can be a significant challenge for instructors. To lower the cognitive load of instructors, Improv helps instructors switch between slides and live coding [23]. Other tools exist to monitor students as they are programming

in a lesson. For example, VizProg helps instructors to monitor live learning by displaying student progress towards a solution [24], and Overcode analyzes students' submissions [25]. Codeopticon gives a gallery view of students' code for tutoring purposes [26]. Our prototype is informed by this work and builds on it by including a gallery view along with other features to encourage student participation and facilitate feedback.

A core strength of live coding is the ability to react to the classroom and adjust materials in real time. However, many existing tools trend towards encouraging progress towards a correct solution, which implies the requirement of exercise test cases to be prepared and loaded into the system before the lesson, thus increasing instructional workload. In addition to adding preparation requirements on the instructional team, by predefining the exercises, the instructor is not able to create activities that respond to the in-class environment, the progress of the lesson, student's questions, and difficulties.

An alternative approach is to provide feedback to the instructor (and students) about trial-and-error, but not necessarily correctness. This gives instructors greater liberty to create exercises during class time and allows for student expression in the results that might not be possible by predefining output, for example by connecting the answer to an important part of their own lives. For this reason, our emphasis in this paper is to encourage mindful experimentation and tinkering rather than arriving at a correct solution.

2.4 Audience Engagement Tools

While instructor-student interaction is trivial in smaller classrooms, it can be near-impossible in a large auditorium. In such settings, tools can extend an in-person performance by supporting

performer-audience interaction at scale; essentially a form of *audience user interfaces* or *audience engagement interfaces* [27]. Clickers are the canonical such audience tools in educational settings, inviting even large classrooms to engage in lectures [28]. By motivating peer-to-peer and the instructor-student interaction, clickers support active collaborative learning and engagement that are associated with favorable learning outcomes [29]. In large classrooms, clickers help to create a positive and active atmosphere, and help the instructor monitor peer-to-peer learning [28].

Audience engagement tools have the potential to invite student participation in live coding sessions. For example, Presemo supports polls, chats, and voting in performer-audience settings [27]. Zoom has a whiteboard feature to annotate code with free-drawing and a screen sharing feature for students and instructors to share their code. Kahoot, common in primary school settings, supports slides, multiple-choice quizzes, polls, puzzles, and open-ended questions. Different features of audience engagement tools offer opportunities to increase engagement in live coding.

2.5 Theoretical Framework: Cognitive Apprenticeship

This research draws on the theoretical framework of *cognitive apprenticeship* [30, 31]. Cognitive apprenticeship—*"a model of instruction that works to make thinking visible... to teach a fairly complex task to students"* [30]—can help understanding and interpreting the benefits of live coding [1] because of its practical, hands-on, and example-based approach. Vihavainen et al. [32] report on how teaching programming with cognitive apprenticeship helped to increase retention rates in an introduction to programming course. Drawing from Collins et al. [30], the key pedagogical aspects of cognitive apprenticeship are

- Modeling: *"teacher performs a task so students can observe"*;
- Coaching: *"teacher observes and facilitates while students perform a task"*;
- Scaffolding: *"teacher provides supports to help the student perform a task"*;
- Articulation: *"teacher encourages students to verbalize their knowledge and thinking"*;
- Reflection: *"teacher enables students to compare their performance with others"*; and
- Exploration: *"teacher invites students to pose and solve their own problems"*.

Of the key pedagogical dimensions of cognitive apprenticeship, live coding is particularly well-aligned with *modeling* [1]. However, further investigation is needed to explore ways in which live coding tools could support the stages of cognitive apprenticeship beyond modeling: coaching, scaffolding, articulation, reflection, and exploration. These stages are particularly consistent with active learning [30, 33]. For this reason, our prototype focuses on supporting the the later phases of cognitive apprenticeship during live coding.

Chapter 3: Methodology

We performed an interview study to understand the barriers and benefits of live-coding. In general, our study is based on a research-through-design methodological approach, *"an approach to conducting scholarly research that employs the methods, practices, and processes of design practice with the intention of generating new knowledge... [such as]... novel perspectives that advance understanding of problematic situations; insights and implications with respect to how specific theory can best be operationalized in a thing and artifacts that both sensitize the community and broaden the space for design action"* [34, pp. 167-168]. The approach relies heavily on prototypes that ground communication and serve to externalize ideas and hypothesis, and invite critique and continued ideation. Prototypes, in their physicality, serve as a sounding board for ideas to spring. However, by presenting designs, there is an inherent priming effect, as the participant is shown one possibility, so brainstorming is not from scratch and ideation might be constrained and altered by familiarity with a particular design solution.

To encourage divergent brainstorming, the aesthetics and the interactions of the prototype are low fidelity: black and white colors and primal buttons over features that invite complex interaction. We introduced the prototype as a work-in-progress tool and encouraged honest critique by reassuring the participants that the draft could be thrown out and recreated without much effort. We asked the participants to perform tasks using the prototype and to think-aloud.

For each design artifact, we asked for feedback on what the participant used and saw. Before and after the prototype review, we interviewed participants. The demographics questionnaire and the facilitator guide that includes the interview questions and test tasks are available at `https://osf.io/rq8bd/?view_only=420c9f015e804297bf7b3d3838e3d317`.

3.1 Prototype

Our prototype live-coding tool (available at `https://gc9vfa.axshare.com/`) was designed to encourage class participation in live coding. It asks students to articulate the strategies the instructor uses and then employ them themselves as they work through the programming activities. We build on prior work that explored reflection of programming strategies with professional software engineers [35]. As discussed above, the prototype is informed by cognitive apprenticeship, encouraging lessons to go beyond modeling to reflect on programming strategies and to engage the student in experimenting with code alongside the instructor. Prior work recommends coding along during live coding as a future avenue for exploration [36].

Here we list key design features of the prototype and link them to documented best practices for supporting apprenticeship in introductory programming contexts:

- *"Learning by doing. The craft will only be mastered by actually practicing it."* [32, p. 94]
  - Students can code along with the instructor.
  - Students can code independently.
- *"Continuous feedback. Continuous feedback must be implemented in both directions. The student receives multi-level feedback from his progress and instructors, and the instructor*

receives feedback by monitoring the students progress and challenges." [32, p. 94].

  - Students can ask questions.
  - Instructor can peek into students code.
  - Instructors can see gallery of students coding.
  - Instructors can create polls.
  - Instructors can give hints.

- *"Teacher encourages students to verbalize their knowledge and thinking."* [30, p. 15],

  - Students can explain the strategies they use.
  - Students can explain the strategies instructors use.

3.1.1 Instructor and Teaching Assistant Prototype Walkthrough

Here we describe the sequence of screens the instructor and teaching assistant reviewed. We asked the instructor and teaching assistant to perform the following tasks:

T1 Imagine that you have just arrived in the classroom and are beginning to set up. Start a live coding session.

T2 Students have arrived in your lecture, and you begin live coding.

T3 Give students a problem to work on independently.

The following describes the sequence of screens based on the tasks.

➔ **Task 1:** The instructor generates a class identifier and shares it with the class to log in. The instructor then starts a live coding session. The instructor can live code in an editor and run

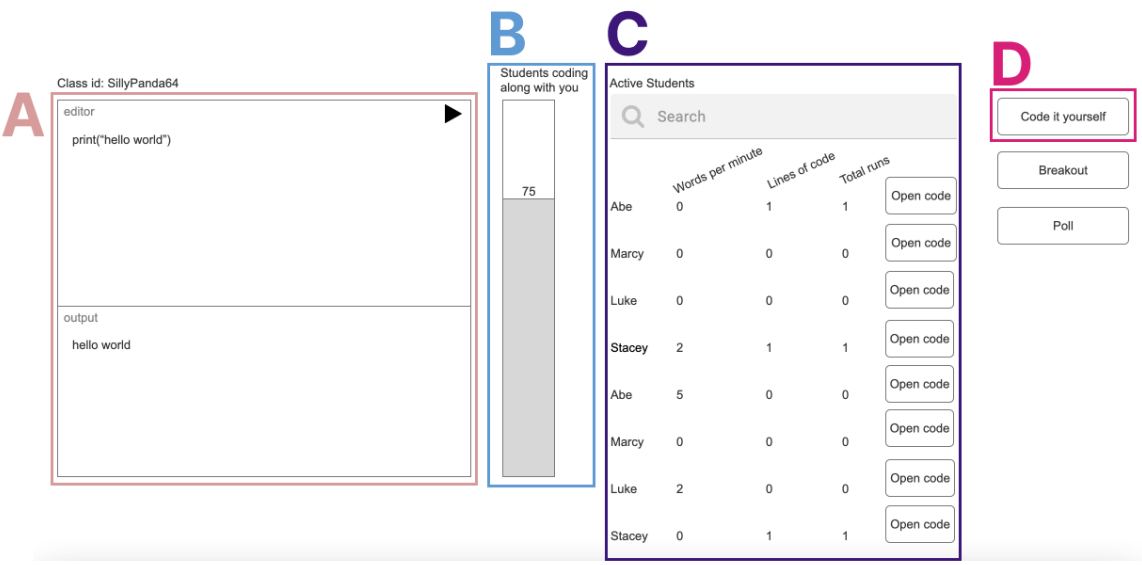

Figure 3.1: **Teacher live coding view.** The teacher can code in their editor (A). The number of students coding along with the instructor is displayed in the meter (B) and information about students is in the active student pane (C). The teacher can start an independent activity by selecting code it yourself (D).

their code (Figure 3.1, A). In addition, the display has a meter to track the number of students who are actively coding along with the demonstration (Figure 3.1, B). There is also a search bar so the instructor can also look up specific students (Figure 3.1, C). At first, the meter is empty when there are no active students populated on the list simulating that an instructor has just arrived, and students are not yet in class, nor in the live coding session.

➲ **Task 2:** The instructor will see a screen (Figure 3.1) that simulates the instructor mid-way through the session. The editor has some code that the instructor has written and resulting output from a recent run (Figure 3.1, A), and the bar is filled up part way with a 75 annotation to communicate that 75 students are coding along with the professor (Figure 3.1, B). Active students are displayed in a dashboard along with metrics on their activity: words per minute, lines of code, and total runs (Figure 3.1, C). The instructor can peek into a student's editor by selecting an open code button for an individual student. The instructor can also begin an independent programming

activity through selecting the 'Code it yourself' button (Figure 3.1, D), start a group activity by selecting 'Breakout', and ask the class multiple choice questions by polling students.

➔ **Task 3:** To start an independent in-class coding activity, the instructor can select "Code it yourself." The instructor can set a timer in minutes and seconds for the activity and type in instructions. To avoid increasing workload and to embody the spontaneity and flexibility of live coding, the instructions are designed to be filled in 'on-the-fly' during the lesson time perhaps inspired by a student's comment.

After the instructor launches the independent coding activity, the instructor can view student activity through a dashboard. Along with peeking into individual students' code, the instructor can view a gallery of students (Figure 5.1) as well as provide hints to the students and message students. In the gallery, the instructor can see each student's editor and output. The instructor can project an ambient visualization which serves as crowdsourced feedback of students' activity in their editors. If students are not active, that is no students are typing or running their code, the projector is a blank, black screen. As students begin the coding activity, the screen comes to life and moves in relation to the typing and running of the code in the class. The underlying motivation is to encourage students in their experimentation and to reward effort instead of correctness.

3.1.2 Student Prototype Walkthrough

Here we describe the sequence of screens the student reviewed. We asked the student to perform the following tasks:

T4 Imagine that you have just arrived in class and are opening your computer. Join the session.

The teacher projects the code 123 on the board.

T5 Program with the instructor.

T6 The instructor has just started a session and asked you to code independently.

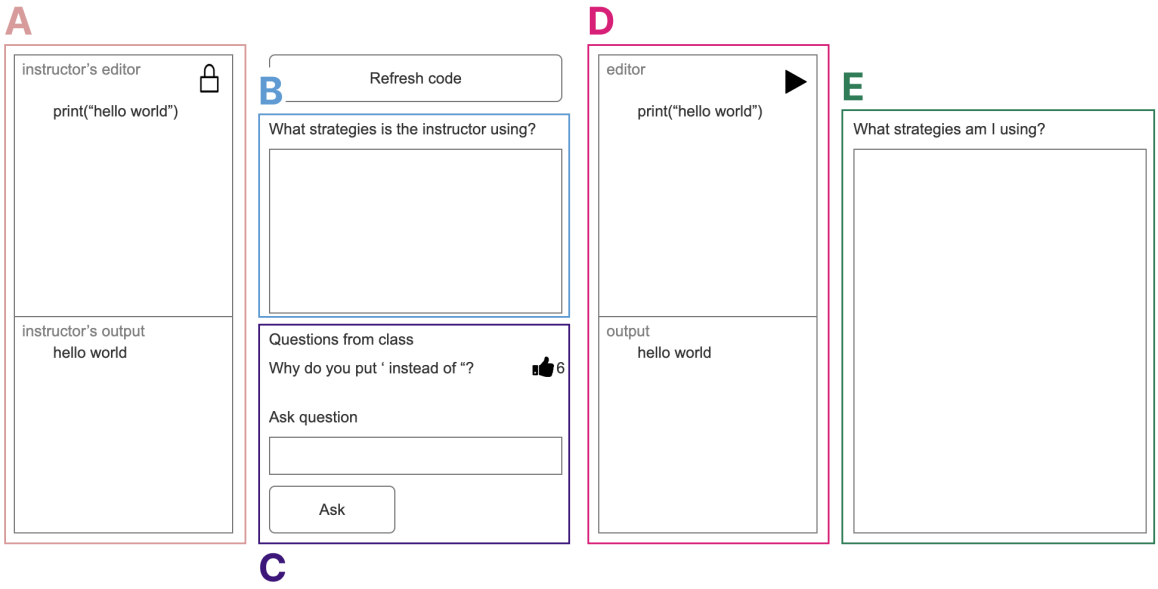

Figure 3.2: **Student coding along with the instructor view.** Student view has the instructor's editor (A), their own editor (D), and notepads (B, E) where the student articulates the strategies they use and the instructor uses while coding. Refresh code updates the student's code to match the instructor's code. The student can ask questions (C).

The following describes the sequence of screens based on the tasks.

➲ **Task 4:** The student prototype begins with a login where the student is asked their name and a class id. After logging in, the student sees a live copy of the instructor's editor and output that is updated on the fly as the instructor types and runs their code (Figure 3.2, A). Inspired by the articulation phase of cognitive apprenticeship [30], there is a note taking box for students to record the strategies the instructor is using (Figure 3.2, B). There is a question and answering system that students can use to pose questions and vote on other student's questions (Figure 3.2, C).

➔ **Task 5:** Here students can code along with the instructor (Figure 3.2, D) and if they fall behind, refresh their code so that an up-to-date version of the code is populated into their window (the same as the current code of the instructor). Also aligned with cognitive apprenticeship, the student can articulate what strategies they themselves are using (Figure 3.2, E).

➔ **Task 6:** When the instructor asks students to code independently, instructions are shown on the page, along with a timer displaying the remaining time in the activity. Students can share their code with the class which would display the code in front of the room with the projector, ask a classmate for help, and check their code with a classmate. While they are working independently, the students can see a visualization in the front of the classroom that moves based on the classes programming activity.

3.2 Positionality

As the methods and analysis are shaped by the researchers and their prior experiences, we discuss the lens the research team approaches the work. Demographics-wise, the author who performed the interviews and thematic analysis is a white, American, neurotypical cis woman in her twenties. She started programming during high school in electives and at an afterschool programming club and studied computer science in her undergraduate degree at a large public university in North America. She learned to program largely through attending lectures where professors would live code and enjoyed this experience.

During the summers, she taught programming and robotics where she live coded often and asked students to live code in front of their peers. She observed moments of pride when students demonstrated mastery in front of their classmates. She holds the belief that anyone can learn to

program given resources, time, and motivation. She is skeptical if technology and new tools are the answers to the problems faced in computer science education, or if technology is the source and contributing force to the problems. Unimpressed with the gender and racial diversity in her undergraduate classes, she strives to make computer science a more welcoming space.

3.3 Participants

We recruited participants through social media and university mailing list servers. Two instructors, seven teaching assistants, and six students participated in the study. All participants were 18 years or older, spoke English, and had either taught or attended a post-secondary course that included programming in the past year. Demographic information about participants is displayed in Table 3.1. With the exception of two participants (P02 and P03), the study was conducted with one participant and the first author. P02 and P03 expressed a preference to participate in the study together, so the first author conducted the study with the two participants at the same time. The study duration was between 30 minutes and 1 hour.

Thirteen participants identified as men, one participant identified as a woman, and one participant identified as non-binary. Although we did not ask, P09 disclosed that he has ADHD and P14 disclosed that she has ADHD, dyslexia, and autism. Including the perspectives of neurodiverse individuals added a layer of richness to the findings.

3.4 Procedures

Participants were asked to complete a demographics questionnaire (age, gender, and educational background) and then participate in a semi-structured interview, a prototype review, and a closing

#	Role	Setting	Age	Gender	Education
P01	Teaching Assistant	Online	26-35	Man	B.Sc. CS; M.Sc. CS; Ph.D. Info. Sci. student
P02	Teaching Assistant	In-Person	18-25	Non-Binary	B.Sc. CS student
P03	Student	In-Person	18-25	Man	B.Sc. CS, Math student
P04	Student	In-Person	18-25	Man	B.A. Psychology; M.Sc. HCI student
P05	Student	In-Person	26-35	Man	B.Eng. CS, Eng.; M.Sc. HCI student
P06	Teaching Assistant	In-Person	18-25	Man	B.Sc. CS student
P07	Teaching Assistant	In-Person	26-35	Man	BSc. CS; MSc. CS; Ph.D. HCI student
P08	Instructor	In-Person	36-45	Man	M.Sc. Info. Sys.
P09	Student	In-Person	18-25	Man	B.Sc. CS student
P10	Student	In-Person	18-25	Man	B.Sc. CS; M.Sc. HCI student
P11	Instructor	In-Person	45+	Man	B.Sc. Chemistry; Ph.D Info. Sci. student
P12	Teaching Assistant	In-Person	18-25	Man	B.Sc. CS student
P13	Student	In-Person	18-25	Man	B.Sc. CS, Robotics student
P14	Teaching Assistant	Online	18-25	Woman	B.Sc. Math, CS; Ph.D. CS, CS Ed. student
P15	Teaching Assistant	In-Person	26-35	Man	B.A. Design; Ph.D. Info. Sci. student

Table 3.1: **Participant demographics.** Participant roles, study setting, age, gender, and education background.

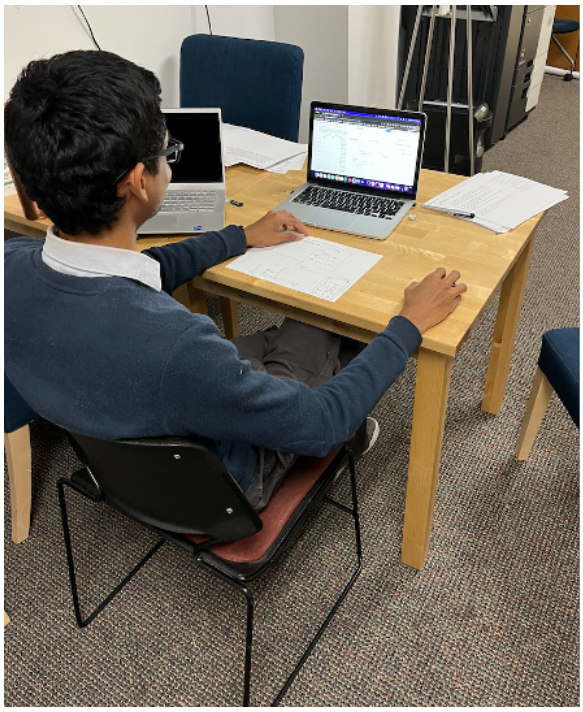

Figure 3.3: **Study setup.** A participant interacting with the prototype during our study. In front of the participant is a sketch where he redrew the interface.

semi-structured interview. Inspired by contextual inquiry techniques [37], the interviews aimed to capture accounts of participants' lived experience.

The sequencing is loosely inspired by the future workshops method [38] as participants critique current live coding practices in the first interview and fantasize about live coding in a perfect world without constraints in the closing interview. By reviewing the prototype between critique and fantasy, the participants interact with a possible design intervention to ignite ideation of other possible solutions (Figure 3.3).

The studies occurred in a U.S. public university setting: two on Zoom and thirteen in-person. We recorded participant's voices and interactions with the prototype through screen recording. All participants gave their informed consent before participating in any of the study activities, and the study was approved by our institution's ethical review board.

With the exception of the first participant, participants interacted with a medium-fidelity prototype of the envisioned live coding platform. After the first participant, we made changes to the prototype for clarity and to reduce the number of presented features. The changes aimed to direct the conversation away from common audience engagement tools (polls and question-asking systems) to provoke conversations specific to live coding.

3.5 Data Analysis

We analyzed the data through thematic analysis [39] guided by practices in analyzing interview data [40] with attention to types and richness of information over counts and frequency [41]. The process of engaging with the interview data took several forms (Figure 3.4 from scribbles on the back of the facilitator guide during in-person interviews that synthesized common topics of discussion from participants to numerous Excel documents with excerpts on interesting quotes from listening to the recordings then onto a visual display of clustered data.

We familiarized ourselves with the entire corpus of interviews and prototype reviews by listening to recordings. In a reflexive manner, codes emerged from the data with a focus on the ways in which students and teachers reflect on their live coding experiences. Descriptions, barriers, benefits, and design opportunities of live coding were codes that emerged from the data. Table 3.2 is an excerpt representing the coding process.

After identifying codes, we went through the interviews to identify excerpts that described the codes. To integrate data we created clusters as depicted in Figure 3.5. We analyzed significant statements to generate themes [41]. After identifying themes, we went through interviews to find additional evidence to support the themes. A colleague reviewed our preliminary groupings of

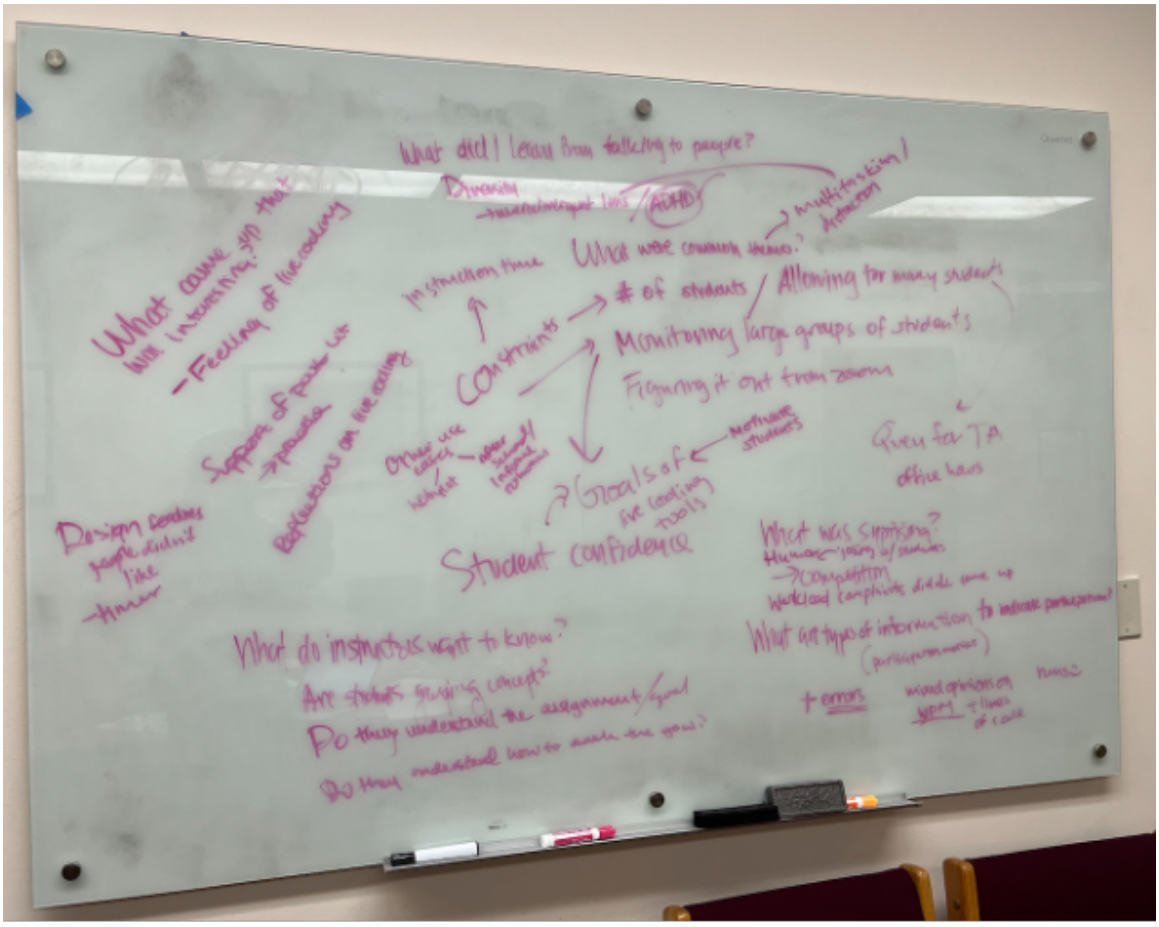

Figure 3.4: **Early data analysis.** Whiteboard with common topics throughout the interviews.

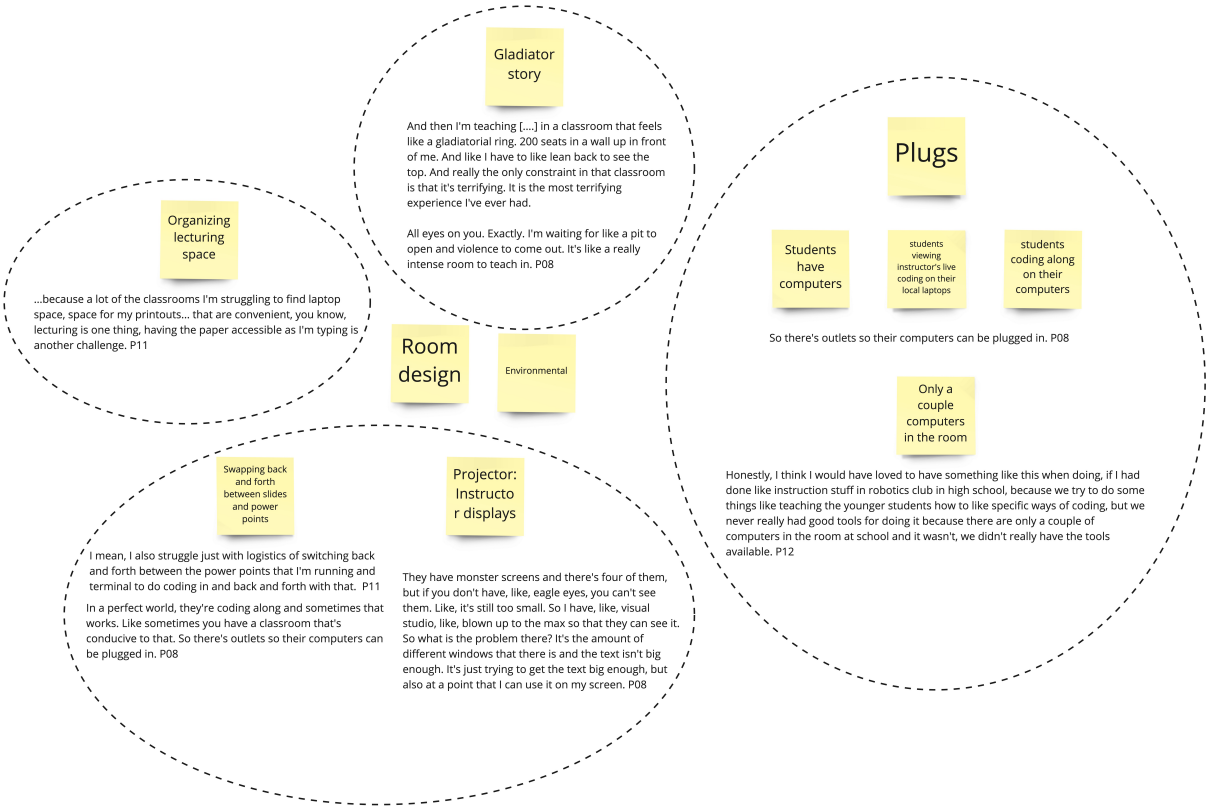

Figure 3.5: **Excerpt of integration via clustering.** Significant statements and notes are clustered on a visual display.

data and gave feedback to improve the clarity of our resulting themes.

CODE	NOTE	QUOTE
Description	Live coding as a performance	*"It's like live performance. It's really hard to practice it enough that you know that it's going to work, but also have that kind of ability to take student suggestions and potentially go in a direction that you haven't tested and might not work out."* (P08)
Barrier	Fear of messing up	*"Part of it is the pressure of just being in front of an audience. And you sort of, I mean, naturally you don't want to mess up. And so thinking of that gives you some sort of, I guess, anxiety, but I guess for me over time at first I was definitely like nervous since it was my first time doing anything like that. But I think in my experience, I got less nervous and much more comfortable. But yeah, I think the main thing is definitely just the anxiety of messing up so badly for students."* (P06)
Benefit	On-the-fly nature of live coding	*"I'm doing some example, then it's easier to change stuff on the fly and then surprise students."* (P02)
Design opportunity	Gallery camera view	*"Okay, these students got it these students didn't. I would love to have a second screen that had, you know, the small kind of security camera view where I had every student desktop and be able to see that they're all on their own."* (P11)

Table 3.2: **Excerpts of coding process.** Interviews were coded into descriptions, barriers, benefits, and design opportunities with accompanying notes.

Chapter 4: Findings

By interviewing instructors, teaching assistants, and students and reviewing a prototype we uncovered barriers and benefits of live coding. Themes arose from the participant responses within the categories of barriers, benefits, and design guidelines. Below we present each theme, including participants's own words.

4.1 Barriers of Live Coding

Barriers of live coding were related to the teaching environment, emotions, and issues with student-teacher communication.

4.1.1 Teaching Environment: Inadequate Resources and Setup

The first barrier our analysis revealed is related to the larger environment in which live coding takes place. Attributes of the classroom and the lack of necessary resources make live coding difficult.

According to P08, the **absence of electrical outlets** in a lecture hall or **inaccessible positioning of electrical outlets** makes it difficult for students to use their laptops during class and code along with the instructor or perform other in-class activities that require a computer. If the class does not occur in a computer lab, participation via personal computer required students have **access to**

a laptop to bring to class. The need for access to electrical outlets is particularly important given the prolonged engagement with often resource-intensive applications.

> *... sometimes you have a classroom that's conducive to [students coding along]. So there's outlets so their computers can be plugged in.* (P08, instructor, pre-interview)

Unlike slide-only lectures, during live coding there are several pieces of information that the instructor must juggle at once. For example, the instructor manages content slides and their live coding environment on a computer. One instructor identified **managing lecture slides and the editor** as challenging, which corroborates prior work [23]:

> *I mean, I also struggle just with logistics of switching back and forth between the PowerPoint [slides] that I'm running and terminal to do coding in...* (P11, instructor, pre-interview)

In addition to digital space management, **organizing the physical lecture space** is difficult during live coding lectures. Instructors who keep a correct working copy of the code have to organize their space to be able to see their code and their cheat sheet.

> *... because a lot of the classrooms I'm struggling to find laptop space, space for my printouts... that are convenient, you know, lecturing is one thing, having the paper accessible as I'm typing is another challenge.* (P11, instructor, prototype review)

The projector is essential during live coding as it displays the code for students to see. Text must be large enough that students sitting at the back in hundred person classrooms can see the code, yet not too large that the code is unmanageable in the instructor's editor. Said P10, a student who regularly attended live-coding lectures:

I think it would have been nice to have some way to follow along better with live coding, [...] especially because my lectures were like 150, 200 people and [it's] such a big class and sometimes you sit in the back and... [...] I realized quite soon that I have to sit all the way in the front for me to pay attention in class, especially in those big classes, because if you sit in the back you get distracted, but also you can't really see what [the instructor is] doing. (P10, student, pre-interview)

An instructor reported struggling to get correct **projector to screen ratio** so that both student and instructor can see the code.

[The room has] monster screens and there's four of them, but if you don't have eagle eyes, you can't see them. It's still too small. So I have Visual Studio blown up to the max so that they can see it. So what is the problem there? It's the amount of different windows that there is and the text isn't big enough. It's just trying to get the text big enough, but also at a point that I can use it on my screen. (P08, instructor, prototype review)

4.1.2 Live Coding Can Be Scary

In addition to problems in the environment, live coding can an emotionally fraught experience. Participants reported feelings of **nervousness** and having **stage fright**:

You're on the spot, and you're typing and you're nervous, you're just going to freeze up. And then everyone is staring at you, and it just makes it worse. (P02, teaching assistant, pre-interview)

Since a common live coding setup is one person in front of many, the attention is focused onto the performer. Furthermore, the classroom layout can create feelings of anxiety while the instructor live codes.

> *And then I'm teaching [...] in a classroom that feels like a gladiatorial ring. 200 seats in a wall up in front of me. And I have to lean back to see the top. And really the only constraint in that classroom is that it's terrifying. It is the most terrifying experience I've ever had.*
>
> ***First author:*** *All eyes on you.*
>
> *Exactly. I'm waiting for a pit to open and violence to come out. It's a really intense room to teach in.* (P08, instructor, prototype review)

The instructor teaches in a tiered lecture hall with hundreds of students. The **performative nature** associated with live-coding can create anxiety. With experience, the stress of live coding is reduced according to teaching assistant P06 and instructors P08 and P11. Says P06,

> *Part of it is the pressure of just being in front of an audience. And you sort of, I mean, naturally you don't want to mess up. And so thinking of that gives you some sort of, I guess, anxiety, but I guess for me over time at first I was definitely nervous since it was my first time doing anything like that. But I think in my experience, I got less nervous and much more comfortable.* (P06, teaching assistant, pre-interview)

Even if the instructor is not performing, there are emotional barriers for both the instructor and student to participate in live coding. If students are coding during class, the thought of monitoring students' editors in an online gallery view can be **overwhelming**.

I imagine I'll struggle to look at everyone's code. I won't be able to help it... I can't imagine 40 [students]. I imagine I'll struggle at like three or four. I'll struggle. Even at two, I can't imagine... Watching two people code... They might be taking different approaches. In that case my brain would just collapse... (P15, teaching assistant, post-interview)

In reacting to asking student to code in front of the class, a teaching assistant noted the punitive nature of editors over classical whiteboard coding. Unlike the free form acceptance of a whiteboard, the editor punishes:

If there's a spelling mistake [when white boarding], or if [a student coding on a whiteboard in front of the class] miss[es] a comma or something, no one cares... You do that on a computer, then ***it'll scream at you****, and then there will be the red squiggly.* (P02, teaching assistant, pre-interview)

Annotations from the editor call out mistakes, which can embarrass a student who is coding in front of the class. The intimidating nature of live coding accentuated with a large audience prevents P08 from asking his students to code in front of their peers:

I worry about [asking students to code in front of the class] in my 100 person classroom, it's intimidating for me sometimes and so I could see that would be really ***intimidating for students****.* (P08, instructor, pre-interview)

The negative emotions associated with live coding make it difficult to teach and participate.

4.1.3 Signal Broken: Student and Teacher No Longer Communicating

Negative emotions may be coupled with the absence of feedback and the pedagogical conversation between instructor and student. When reflecting on the pacing of the lecture, a teaching assistant noted difficulty having a back-and-forth with students.

> *... there's no way for us [instructor and teaching assistants] and for them [students] to communicate on the code. So the only time we sort of ping the class, do you get it, is there any questions? That's the only time [students] can have an opportunity to get feedback. And that's the only time for [the professor] to have an opportunity to give it.* (P15, teaching assistant, prototype review)

The teaching assistant explained that during live coding lectures, there are few opportunities for the instructor to **check for understanding** and students to ask questions. Another teaching assistant noted the unidirectional nature of communication where students are **passively looking on as he explains**.

> *So I feel [live coding] is a one-way street. So it's just me talking about what I'm writing down like that. And them seeing the screen... I found it very easy for students to kind of switch off. So they do, they'll just keep looking at it and then, oh, that works.* (P07, teaching assistant, pre-interview)

The instructors and teaching assistants in our study often perceived that such lack of active learning led to **student disengagement**.

Monitoring what students are doing during live coding lectures is hard. Teaching is an embodied experience where teachers use body language cues (P14, teaching assistant), the flurry

of students typing as they begin an exercise (P11, instructor), and noise of students collaborating (P08, instructor) to gauge student participation. By walking around small classrooms, teachers can observe what students are doing, even if they quickly alt-tab away from another assignment (P07, teaching assistant). The number of students in in-person lectures and the absence of body language and classroom sounds in virtual environments can make it hard for teachers to gauge student understanding and participation.

4.2 Benefits of Live Coding

Alongside the barriers our analysis identified, we also found positive aspects of the practice. We found themes consistent with cognitive apprenticeship in the benefits of live coding, especially in modeling mistakes and scaffolding. We also found that live coding can facilitate student-centered learning. Findings corroborate prior work on student's perceptions on the benefits of live coding [19].

4.2.1 The Power of Mistakes

Unlike presenting complete, static code, making mistakes while live coding helps students to learn. The teacher modeling the mistakes helps to identify promising paths to take by demonstrating faulty courses of action. This was commented on by a student who said:

> *... when he makes mistakes, it's like that's a good example of what not to do. And if he makes the code beforehand then you're not really gonna see that.* (P09, student, pre-interview)

Students can learn **programming strategies** from the teacher's mistakes. At the same time,

these mistakes can serve as teaching opportunities for instructors to talk about the at-times messy practice of programming. As one teacher said:

> *Part of it is because I know I'm going to make mistakes and then I can use those mistakes to teach students certain things.* (P06, teaching assistant, pre-interview)

In brainstorming strategies for a new instructor, P11 advises *"Don't don't stress about mistakes. They are learning opportunities"*. Mistakes, both intentional and accidental, help to **foster student engagement**.

> *Oh, the students love to find the problems as I code them. I kind of built this open atmosphere with a lot of debugging being a big focus of the course and encourage the students as I'm typing and we're seeing what's going on if they see something that's wrong, a lot of times they'll say aloud. Otherwise, it's the process... I built in errors in the code to see a lot of the problems that they may stumble across as they code.* (P11, instructor, pre-interview)

By creating a treasure-hunt for mistakes while live coding, some instructors harness students attention in the learning activity.

4.2.2 Supporting Student-Centered Learning

Live coding affords teachers to **react** to students and to **change on the fly**, leading to reports of student engagement.

> *I'm doing some example, then it's easier to change stuff on the fly and then surprise students.* (P02, teaching assistant, pre-interview)

Asking students to make **predictions** and incorporating their suggestions is a pedagogical practice that can accompany live coding and was mentioned be several teachers (P08, P12, P14).

> *So when it goes really well is when you can get students to participate. So it's like, you know, you're doing some code and it's like, okay, what's the next step and you get feedback and you can be like, oh, well, that's a great suggestion. Let's try that and maybe it doesn't work out and then you can say, okay, so how do we adjust?* (P08, instructor, pre-interview)

On a related note, live coding allows for flexibility to introduce areas that the instructor has not yet explored.

> *Well, what if we did blah? Yeah. And I go, that's a good question. I have never thought about approaching this that way. Let's see what it does.* (P11, instructor, pre-interview)

Unlike slide-based lectures and static code examples where all material is fixed, our participants felt that live coding supports student-centered learning by incorporating suggestions known by the student and that might be unknown to the professor.

4.2.3 Changing Scaffolding on the Fly

During live coding, the teacher can modify the difficulty by changing the content of examples and the amount of skeleton code provided. If the exercise is too easy, live coding affords changes in real time. By **adjusting the difficulty**, the instructor can increase engagement. This can be seen in one instructors reflection of their own live coding practice:

I was like losing them. So then like, whatever, I just made the example more complicated, like nesting arrays, or storing different kinds of things in arrays, or making the example more applicable, for example. And yeah, that worked. I was able to live respond, I guess. (P02, teaching assistant, pre-interview)

The instructor has control over the **amount of starter code** provided to students, and can start from nothing or begin with a skeleton.

I give them some skeleton code at the beginning point to start with and then they will start to develop the pieces that I'm doing at the same time. (P11, instructor, pre-interview)

However, a student reported difficulty if the instructor begins with starter code without explanation:

It's usually better when they start from zero with some example. I've seen some professors or where if they're finishing some project and half of it is already done. And then they start from there, the professor already has the half of it in their mind, but this might be the first time ever a student is seeing it. And yeah, I've had that experience and being completely lost. And then you have to go back afterwards and see the half that they started with, understand that first, and then mentally replay the lecture and then it makes sense but it's a little bit of work. (P02, teaching assistant, pre-interview)

A principle of cognitive apprenticeship is *making thinking visible* [30] and a tenant of live coding is the demonstration of the incremental process of programming [2]. By starting mid-way through the code without explanation, the thinking process behind the code is hidden from

students. Scaffolds, while they provide the instructor freedom, should be used wisely as to not create confusion.

Chapter 5: Discussion

This analysis of teacher and student experiences with live coding in large introductory computer science classrooms revealed insights into barriers associated with live coding as well as benefits of the pedagogical approach. We interpreted our findings, delving into possible considerations to overcome problems in live coding and ways to support effective live coding. Here we propose design guidelines for consideration when building live coding tools. Despite presenting compelling dimensions of live coding, our study is limited in terms of practical outcomes and actual tooling support. More work is required to gain further insight into live coding as a pedagogical practice.

5.1 Interpretation of Findings

The major barriers of live coding identified in our analysis include

1. Constraints of the teaching environment;
2. Negative feelings; and
3. Communication-related issues.

The major benefits of live coding are

1. Intentional and unintentional mistakes;

2. Student involvement; and

3. Flexibility of scaffolding.

In other words, our findings suggest that while live coding has pedagogical benefits consistent with cognitive apprenticeship, there are barriers to the practice. Select barriers hinder participatory flavors of live coding: student-instructor collaborative live coding and student-led live coding. Namely, the student can not participate via laptop if their learning environment lacks necessary computing supports. Even if the environment is equipped, the feelings of stage fright and embarrassment may prevent students from coding in front of their peers, and the instructor often will not ask due to fear of putting a student on the spot.

In the design and selection of classrooms for introductory computer science courses, our findings suggest identifying room layouts that elicit comfort and confidence amongst teachers as opposed to anxiety. The podium should have ample space for instructors to organize their materials and the necessary equipment so that the teacher can live code comfortably. Finally, the projection screens should be large enough (or there should be multiple ones in different locations) so that all students can see code on the projector, regardless of their position in the room. Additional teaching support in the classroom could help offset the instructor feeling overwhelmed, as additional teaching staff could help monitor student engagement and progress for in-class exercises.

Our findings indicate that although teachers and students reported educational benefits, live coding is a daunting task. New instructors and teaching assistants might find support and advice from senior faculty to gain confidence in live coding. In addition to reducing discomfort, experienced teachers could share practices and techniques that support the reported educational

benefits of live coding.

Finally, live coding might require new instructors (or seasoned instructors teaching an unfamiliar topic) to abandon the notion that they are experts. Effective live coding involves making mistakes in front of the classroom and incorporating student suggestions. Debugging faulty code under time pressure and the watchful eyes of a hundred students adds significant difficulty to an already cognitively taxing task. This often requires a mindset change for instructors accustomed to smooth and structured slides-only class delivery methods. In other words, the instructor must get used to not being an infallible expert, but a guide in the collaborative learning process for the entire classroom.

5.2 Design Guidelines

In the design of tools to support student engagement in live coding, new technology should lessen or remove barriers of live coding all together and bolster the positive attributes of live coding. We propose the following design considerations. Table 5.1 situates the design guidelines into our theoretical framework and the type of live coding facilitated by the guideline. The creation of design guidelines is grounded in our findings as an effort to reduce barriers and bolster benefits.

5.2.1 Personal Computers Optional

During lessons where the instructor live codes, students should be able to opt in or out of using their own personal laptops. Some students and teachers expressed that computers acted as a distractor. Some preferred to listen only or take notes via pen and paper. Paired with the

constraints of access to laptops and outlets, personal computer use should be optional. Students should be able to follow the lecture in a way that best serves their personal learning style. Not requiring personal computers addresses constraints in the teaching environment and helps to facilitate student attention to the pedagogically beneficial parts of instructor-led live coding (e.g., modelling mistakes).

Future work should explore the reintroduction of analog audience engagement tools (like clickers) back into the classroom to encourage student involvement without introducing digital distractions.

5.2.2 Directing Attention

Tools should support directing student attention to areas of the code of importance. A teaching assistant reported that her students struggle to identify what they should be looking at and paying attention to, a problem linked to the teaching environment and teacher-student communication. Live coding tools should direct student attention to which part of the code is being discussed so that students are synchronizing the instructors words with the intended piece of code. If students are looking at the right portion of the code, they can better observe the process of creating programs and debugging, and have the requisite context to be able to participate. Tools could support giving teacher's control of which areas of their screen are projected so that only a portion of a window is projected. Increased control over screen sharing will also help the projector-to-screen ratio be manageable from the perspective of the instructor and readable from the perspective of the student in a large lecture hall. Within the projected code, highlighting or zooming are possible mechanisms to concentrate attention.

5.2.3 Many Keyboards, One Digital Space

In student-teacher collaborative live coding, supporting collaborative editing in a shared space where the instructor and student can code on their own laptops in the same digital space helps to respect personal space. In classrooms where moving around is difficult, having a shared digital space helps to facilitate the sharing of code. In addition to making students and teachers feel comfortable, a shared digital space has the opportunity to make coding in the class less daunting as the student is not required to be physically in front of the classroom. Since both teacher and student see the same code, more directed and specific communication is possible. A shared space invites student involvement.

5.2.4 Errors as Signals of Student Progress

When students live code during a lecture, instructors expressed interest in seeing the errors students produce when they compile and/or run their code as a way of monitoring progress. The existence of errors students encounter informs the teacher that students are participating. The types of errors helps to communicate the road blocks students encounter. Error information helps the instructor to know when to provide feedback and what type of feedback to provide bridging communication between student and instructor.

5.2.5 Peeking into Student Editors

Instructors noted that it was helpful to be able to see the students' editors to monitor progress, identify students to ask to share their work, and troubleshoot. Additional instructional support might be needed, such as an assistant in the room to monitor the gallery depending on the

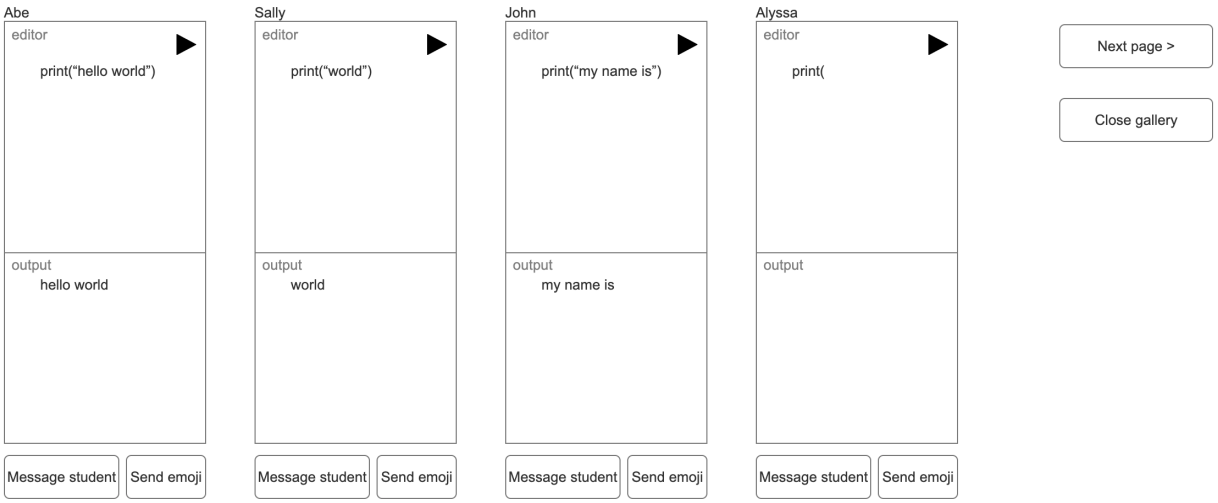

Figure 5.1: **Gallery view.** Instructors and teaching assistants can monitor students coding, send them a message, or an emoji.

size of the class. Supporting the teacher's ability to see student's code facilitates communication.

One design possibility is depicted in Figure 5.1.

Design Guideline	Cognitive Apprenticeship Stage	Type of Live Coding
Personal computers optional	Modeling	Instructor-led
Directing attention	Modeling	Instructor-led
Many keyboards, one digital space	Coaching, Scaffolding	Student-instructor collaborative
Errors as signals of student progress	Coaching, Scaffolding	Student-led
Peeking into student editors	Coaching, Scaffolding	Student-led

Table 5.1: **Overview of design guidelines.** A mapping of the design guidelines to cognitive apprenticeship aspects and the type of live coding the design guideline facilitates.

5.3 Limitations

Half of P04's data was lost due to technical problems reducing the corpus of data that was analyzed. Our work includes a small number of participants, therefore findings are not generalizable. Furthermore, since the work was conducted in a U.S. public university, findings

might not translate to other educational settings or geographic contexts. The set of barriers, benefits and proposed design guidelines are not exhaustive. Finally, we have proposed the above the design guidelines, but not yet validated them in a real live-coding tool.

5.4 Future Work

The list of barriers, facilitators and design guidelines is not exhaustive. Future work could expand these findings by running future studies with groups of participants with differing backgrounds from the participants in our study. Furthermore, we plan to validate the proposed design guidelines.

Participants noted that there might be a connection between our work and tools for teaching writing. Additional research could investigate the relationship between live writing and live coding tools. Outside of the introductory to programming classroom context examined in this paper, participants highlighted the applicability of live coding tools to support knowledge transfer in hobbyist communities and informal learning environments, like robotics clubs. Continued work should explore live coding in other settings. Finally, we recommend exploring the reintroduction of analog audience engagement tools (like clickers) to facilitate classroom participation in live coding exercises.

Chapter 6: Conclusion

By interviewing instructors, teaching assistants, and students and getting their feedback on a prototype we created, we uncovered barriers and benefits of live coding. We then use these findings to propose design guidelines for live-coding support tools. Constraints in the teaching environment, negative feelings, and communication-related issues make live coding hard. Yet, pedagogical benefits exist to live coding such as modeling intentional and unintentional mistakes, encouraging student involvement, and flexibility in scaffolding. We recommend tools remove or reduce barriers and support positive aspects of live coding, and present additional design guidance for attributes, tasks, and features that live coding tools could support.

6.1 Reflection

As the thesis represents a learning experience of practicing research and apprenticeship of senior academic and junior academic, I reflect on the process of planning, performing, and reporting on the research for this thesis.

6.1.1 Research is Living

During the literature review, deliberate practice came to life during tennis lessons. The minute corrections to my forehand and backhand—a form of continuous feedback—immersed

me in the literature. I interacted with the literature in an unexpected and new dimension by connecting concepts from education research to my own life. Although on a plane distinct from coding and university classrooms, I translated mechanisms of feedback for tennis instruction into the prototypes. I learned that research is different from other tasks; it's hard to put down. After the day is done, unanswered questions linger, and answers and inspiration come from unexpected sources. I noticed that when I was far away from screens, I had breakthroughs.

6.1.2 Writing as a Process; Writing is Practice

Academic writing scared me. The thesis process taught me that writing is a process where the drafts will be iterated upon, so they need not be perfect. In fact, I learned to drop kick the perfectionist out the window as she was the enemy of completion. Inspired by Stephen King, I aimed to write courageously. That meant getting words out fearlessly and unabashedly despite of my own self-critic. I attended library workshops which helped append my toolkit of strategies and tips for writing. Yet, I found the best way to improve at writing was to actually sit down and write.

I learned through experience that writing, like tennis, required practice. But, I wanted practice to be fun, so blogging and journaling were my two practice arenas. Keeping the muscles warm helped me to produce many words in few hours that were of decent quality. Writing came in bursts, and was motivated by deadlines. The immersion and immediacy drove the production of prose. I was reminded of a course I took by Professor Laurance Ouellet Tremblay at McGill on the variety of approaches to writing, and kept these in mind for inspiration, although sadly I didn't have the opportunity to try Hemingway's approach to writing, and editing during my

thesis. Perhaps writing with a drink in hand is better for fiction.

6.1.3 Keys: Confidence and Joy

Being confident in what I was doing, even though it was the first time doing it, was cornerstone to conducting the work involved in this thesis. Recognizing that what I was doing whether it be prototyping or writing was not rocket science, and I, in fact, was prepared to do the task at hand. Dropping my internal imposter and replacing her with an academic powerhouse persona made each day easier and more enjoyable. As soon as I did something once, for example, writing a paper, I knew the following times would be less intimidating. Each time I proved I could do it, I gained confidence in myself.

Despite my affinity for timelines and planning, there were occasions where a clear path forward was blurry. When I came to a crossroad, I took my interest into consideration; what would be the most fun to do next? The benefit to the approach is I found my days enjoyable and fulfilling. Confidence and joy facilitated intrinsic motivation and ownership over the project.

Appendix A: Institutional Review Board Exemption Letter

UNIVERSITY OF MARYLAND
INSTITUTIONAL REVIEW BOARD

1204 Marie Mount Hall
College Park, MD 20742-5125
TEL 301.405.4212
FAX 301.314.1475
irb@umd.edu
www.umresearch.umd.edu/IRB

DATE:	January 31, 2023
TO:	Niklas Elmqvist, PhD
FROM:	University of Maryland College Park (UMCP) IRB
PROJECT TITLE:	[1971012-1] Learning Environments for Computer Programming
SUBMISSION TYPE:	New Project
ACTION:	DETERMINATION OF EXEMPT STATUS
DECISION DATE:	January 31, 2023
REVIEW CATEGORY:	Exemption category #45CFR46.104(d)(2)(ii)

Thank you for your submission of New Project materials for this project. The University of Maryland College Park (UMCP) IRB has determined this project is EXEMPT FROM IRB REVIEW according to federal regulations.

We will retain a copy of this correspondence within our records.

If you have any questions, please contact the IRB Office at 301-405-4212 or irb@umd.edu. Please include your project title and reference number in all correspondence with this committee.

This letter has been electronically signed in accordance with all applicable regulations, and a copy is retained within University of Maryland College Park (UMCP) IRB's records.

Bibliography

[1] Ana Selvaraj, Eda Zhang, Leo Porter, and Adalbert Gerald Soosai Raj. Live coding: A review of the literature. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education*, pages 164–170, New York, NY, USA, 2021. ACM.

[2] Sanwar Ali. Effective teaching pedagogies for undergraduate computer science. *Mathematics & Computer Education*, 39(3):243–257, 2005.

[3] Joel Michael. Where's the evidence that active learning works? *Advances in Physiology Education*, 30(4):159–167, 2006.

[4] Ricardo Caceffo, Guilherme Gama, and Rodolfo Azevedo. Exploring active learning approaches to computer science classes. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, pages 922–927, New York, NY, USA, 2018. ACM.

[5] Maya El Helou, Mona Nabhani, and Rima Bahous. Teachers' views on causes leading to their burnout. *School Leadership & Management*, 36(5):551–567, 2016.

[6] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail N. Giannakos, Amruth N. Kumar, Linda M. Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. Introductory programming: a systematic literature review. In *Proceedings Companion of the ACM Conference on Innovation and Technology in Computer Science Education*, pages 55–106, New York, NY, USA, 2018. ACM.

[7] Anthony V. Robins. *Novice Programmers and Introductory Programming*, page 327–376. Cambridge Handbooks in Psychology. Cambridge University Press, Cambridge, UK, 2019.

[8] Lauri Malmi, Judy Sheard, Päivi Kinnunen, Simon, and Jane E. Sinclair. Theories and models of emotions, attitudes, and self-efficacy in the context of programming education. In *Proceedings of the ACM International Computing Education Research Conference*, pages 36–47, New York, NY, USA, 2020. ACM.

[9] Katrina Falkner and Judy Sheard. *Pedagogic Approaches*, page 445–480. Cambridge Handbooks in Psychology. Cambridge University Press, Cambridge, UK, 2019.

[10] Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education*, 18(1):1:1–1:24, 2017.

[11] Monica M. McGill and Adrienne Decker. Construction of a taxonomy for tools, languages, and environments across computing education. In *Proceedings of the ACM International Computing Education Research Conference*, pages 124–135, New York, NY, USA, 2020. ACM.

[12] João Henrique Berssanette and Antonio Carlos de Francisco. Active learning in the context of the teaching/learning of computer programming: A systematic review. *Journal of Information Technology Education: Research*, 20:201–220, 2021.

[13] Charles C. Bonwell and James A. Eison. *Active learning: Creating excitement in the classroom*. The George Washington University, School of Education and Human Development, Washington, DC, 1991.

[14] Scott Freeman, Sarah L. Eddy, Miles McDonough, Michelle K. Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23):8410–8415, 2014.

[15] Jeffrey E. Froyd, Maura Borrego, Stephanie Cutler, Charles Henderson, and Michael J. Prince. Estimates of use of research-based instructional strategies in core electrical or computer engineering courses. *IEEE Transactions on Education*, 56(4):393–399, 2013.

[16] Scott Grissom, Renée A. McCauley, and Laurie Murphy. How student centered is the computer science classroom? A survey of college faculty. *ACM Transactions on Computing Education*, 18(1):5:1–5:27, 2017.

[17] K. Anders Ericsson, Ralf Th. Krampe, and Clemens Tesch-Römer. The role of deliberate practice in the acquisition of expert performance. *Psychological Review*, 100(3):363–406, 1993.

[18] Alexander Johan Nederbragt, Rayna Michelle Harris, Alison Presmanes Hill, and Greg Wilson. Ten quick tips for teaching with participatory live coding. *PLoS Computational Biology*, 16(9):1–7, 2020.

[19] Adalbert Gerald Soosai Raj, Jignesh M. Patel, Richard Halverson, and Erica Rosenfeld Halverson. Role of live-coding in learning introductory programming. In *Proceedings of the Koli Calling International Conference on Computing Education Research*, pages 13:1–13:8, New York, NY, USA, 2018. ACM.

[20] Per Lauvås and Rolando Gonzalez. An experience report using scrimba: An interactive and cooperative web development tool in a blended learning setting. In *Norsk IKT-konferanse for forskning og utdanning*, Oslo, 2017. Open Journal Systems.

[21] Yu-Tzu Lin, Martin K-C Yeh, and Sheng-Rong Tan. Teaching programming by revealing thinking process: Watching experts' live coding videos with reflection annotations. *IEEE Transactions on Education*, 65(4):617–627, 2022.

[22] Marc J. Rubin. The effectiveness of live-coding to teach introductory programming. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, pages 651–656, New York, NY, USA, 2013. ACM.

[23] Charles H. Chen and Philip J. Guo. Improv: Teaching programming at scale via live coding. In *Proceedings of the ACM Conference on Learning @ Scale*, pages 9:1–9:10, New York, NY, USA, 2019. ACM.

[24] Ashley Zhang, Yan Chen, and Steve Oney. Vizprog: Identifying misunderstandings by visualizing students' coding progress. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, page forthcoming, New York, NY, USA, 2023. ACM.

[25] Elena L. Glassman, Jeremy Scott, Rishabh Singh, Philip J. Guo, and Robert C. Miller. Overcode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction*, 22(2):7:1–7:35, 2015.

[26] Philip J. Guo. Codeopticon: Real-time, one-to-many human tutoring for computer programming. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 599–608, New York, NY, USA, 2015. ACM.

[27] Matti Nelimarkka, Kai Kuikkaniemi, Antti Salovaara, and Giulio Jacucci. Live participation: Augmenting events with audience-performer interaction systems. In *Proceedings of the ACM Conference on Designing Interactive Systems*, pages 509–520, New York, NY, USA, 2016. ACM.

[28] Jane E. Caldwell. Clickers in the large classroom: Current research and best-practice tips. *CBE—Life Sciences Education*, 6(1):9–20, 2007.

[29] Lorena Blasco-Arcas, Isabel Buil, Blanca Hernández-Ortega, and F Javier Sese. Using clickers in class. the role of interactivity, active collaborative learning and engagement in learning performance. *Computers & Education*, 62:102–110, 2013.

[30] Allan Collins, John Seely Brown, and Ann Holum. Cognitive apprenticeship: Making thinking visible. *American Educator*, 15(3):6–11, 1991.

[31] Allan Collins and Manu Kapur. *Cognitive Apprenticeship*, page 109–127. Cambridge Handbooks in Psychology. Cambridge University Press, Cambridge, UK, 2 edition, 2014.

[32] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, pages 93–98, New York, NY, USA, 2011. ACM.

[33] Richard M. Felder and Rebecca Brent. Active learning: An introduction. *ASQ Higher Education Brief*, 2(4):1–5, 2009.

[34] John Zimmerman and Jodi Forlizzi. Research through design in HCI. In Judith S. Olson and Wendy A. Kellogg, editors, *Ways of Knowing in HCI*, pages 167–189. Springer, New York, NY, USA, 2014.

[35] Maryam Arab, Thomas D. LaToza, Jenny Liang, and Amy J. Ko. An exploratory study of sharing strategic programming knowledge. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 66:1–66:15, New York, NY, USA, 2022. ACM.

[36] Derek Hwang, Vardhan Agarwal, Yuzi Lyu, Divyam Rana, Satya Ganesh Susarla, and Adalbert Gerald Soosai Raj. A qualitative analysis of lecture videos and student feedback on static code examples and live coding: A case study. In *Proceedings of the ACM Australasian Computing Education Conference*, pages 147–157, New York, NY, USA, 2021. ACM.

[37] Karen Holtzblatt and Sandra Jones. Contextual inquiry: A participatory technique for system design. In *Participatory Design*, pages 177–210. CRC Press, Boca Raton, 2017.

[38] Rene Victor Valqui Vidal et al. The future workshop: Democratic problem solving. *Economic Analysis Working Papers*, 5(4):21, 2006.

[39] Victoria Clarke, Virginia Braun, and Nikki Hayfield. Thematic analysis. *Qualitative psychology: A practical guide to research methods*, 3:222–248, 2015.

[40] Robert S. Weiss. *Learning from Strangers: The Art and Method of Qualitative Interview Studies*. Simon and Schuster, New York, NY, USA, 1995.

[41] John W. Creswell and Cheryl N. Poth. *Qualitative Inquiry and Research Design: Choosing Among Five Approaches*. Sage publications, Los Angeles, CA, USA, fourth edition, 2018.