Live Coding in the CS Classroom: An Interview Study of Instructor Practices

Daniel Manesh
danielmanesh@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Vee Pettit
vpettit@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Yuhang Zheng
yuhangzheng@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Yan Chen
ych@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

David H. Smith IV
dhsmith4@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Sang Won Lee
sangwonlee@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Abstract

Background and Context. Live coding is a pedagogical technique where an instructor writes code in front of their class. The literature on live coding consists primarily of experiments and case studies, and few researchers have sought to understand how instructors actually employ live coding in their classrooms.

Objectives. In this work, we seek to gain an in-depth understanding of how instructors use live coding, including what they are teaching with live coding, how live coding fits in with other lecture activities, and how instructors expect students to engage.

Method. We conducted semi-structured interviews with 22 instructors who have experience with live coding, and we analyzed the results using thematic analysis.

Findings. Our findings suggest that live coding is a flexible tool that is used in a variety of ways across the CS curriculum. Live coding can fit into many different lecture styles, and is commonly paired with active learning activities such as peer instruction or coding exercises. Instructors valued live coding for its many opportunities for student interaction, but they often felt these opportunities were not fully taken advantage of in practice. Instructors had mixed feelings about whether students should type along with them during live coding, and providing or restricting access to the completed code or lecture recordings was sometimes seen as a way to influence student behaviors during lectures.

Implications. Our work contextualizes prior studies on the efficacy of live coding and informs future directions for exploration, which can include examining the affective impacts of live coding and exploring how we might augment live coding to improve how we impart knowledge of processes and procedures. In addition, our work highlights possibilities for designing tools for the CS classroom that can encourage student participation in live coding and facilitate lecture delivery.

CCS Concepts

• **Social and professional topics → Computing education**.

ICER 2026 Vol. 1, Uppsala, Sweden

ACM ISBN 979-8-4007-2203-5/2026/08
https://doi.org/10.1145/3765964.3811669

Keywords

live coding, semi-structured interviews, lectures

ACM Reference Format:
Daniel Manesh, Vee Pettit, Yuhang Zheng, Yan Chen, David H. Smith IV, and Sang Won Lee. 2026. Live Coding in the CS Classroom: An Interview Study of Instructor Practices. In *Proceedings of the ACM Conference on International Computing Education Research Vol.1 (ICER 2026 Vol. 1), August 11–14, 2026, Uppsala, Sweden.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3765964.3811669

1 Introduction

In the context of computing education, live coding refers to a pedagogical technique in which instructors write code live in front of learners, typically projecting their screens and thinking aloud to explain their thoughts [33, 48–50]. Live coding is valued by instructors as a way to demonstrate tacit programming knowledge like debugging, testing, and incremental development [3, 49]. Students also often perceive live coding as engaging and beneficial [16, 19, 61]. Live coding appears to be widespread, and many have advocated for its use in the CS classroom [7, 10, 16, 18, 50, 70].

Despite its apparent popularity, few have explored how instructors operationalize live coding in general practice. Much prior work on live coding consists of experience reports [8, 18, 40] and experiments [16, 48, 53, 55, 61, 69], which only offer a glimpse into how each individual author used live coding. While some researchers have reported on interviews with instructors [3, 33, 63], two of these studies rely primarily on interviews with TAs [3, 63], and the studies have a comparatively narrow scope—for example, focusing primarily on expected student behaviors during live coding [33]. There are countless choices instructors might make when integrating live coding into their lectures: How much of the lecture will be live coding? What material is appropriate for live coding? What other lecture methods will be used? What materials will be provided to students after the lecture? The detailed choices instructors make when using live coding, and the reasoning behind them, remain underexplored.

Addressing this research gap can be useful to practitioners who might want to reflect on or modify their own instructional practice. It can also be useful for education researchers, as it contextualizes prior research on live coding and points to future areas of exploration. For example, while the perceived costs and benefits of live

Table 1: A summary of the themes we constructed through thematic analysis and their relationship to our research questions.

Theme	Section		Summary
Live coding is versatile	§ 4.1	(**RQ1A**)	Live coding is used across the CS curriculum (§ 4.1.1) to introduce and apply concepts (§ 4.1.2) and to demonstrate a variety of processes and procedures (§ 4.1.4). It may encourage a positive attitude regarding making mistakes (§ 4.1.3) and around creative exploration with code (§ 4.1.5).
	§ 4.2	(**RQ1B**)	In live coding lectures, instructors also present information with slides, whiteboards, and computational notebooks (§ 4.2.1). Live coding can also be used to transition into and out of active learning activities like peer instruction or coding exercises (§ 4.2.2).
(Missed) Opportunities for Participation	§ 4.3	(**RQ2**)	While live coding provides several opportunities for student participation (§ 4.3.1), instructors often wish students took better advantage of these opportunities (§ 4.3.2).
Nudging Student Behavior through Selective Access to Course Materials	§ 4.4	(**RQ2**)	Instructors have mixed feelings about whether students should type along with them during live coding (§ 4.4.1). Instructors may try to influence student behavior by providing or withholding materials like lecture notes and video recordings (§ 4.4.2).

coding are well-documented [52], it is possible these costs and benefits are mediated by the specific ways that live coding is employed. Especially given recent work that suggests live coding does not lead to learning gains [16, 53, 55, 69], it can be important to understand if there are variations in live coding practices that are not captured by these studies.

In this work, we address this gap by conducting interviews with 22 CS instructors who regularly employ live coding in their lectures. Through our interviews, we set out to gain a deeper understanding of how instructors use live coding by answering the following research questions.

- **RQ1:** How do instructors employ live coding in the CS classroom?
  - **RQ1A:** What type of content is being taught with live coding and why?
  - **RQ1B:** How does live coding fit in with the rest of the lecture?
- **RQ2:** What do instructors envision as the role of the student in live coding?
  - **RQ2A:** What do instructors want students to do during live coding and why?
  - **RQ2B:** How do instructors encourage such student behaviors?

Table 1 summarizes our results. Regarding RQ1, we found that live coding is a highly versatile technique that instructors use across the CS curriculum. Our instructors use live coding to introduce and apply concepts, demonstrate procedures, and model effective programming and problem-solving processes. Often, our instructors used live coding for just one part of a larger lecture, for example, to introduce and contextualize active learning activities like coding exercises and peer instruction.

Regarding RQ2, we found that instructors had widely divergent opinions on whether students should type along with them while live coding, with some thinking it would help them learn and others feeling like it was a distraction. Instructors valued that live coding could provide several touchpoints for student interaction, but many wished that students participated more. While instructors usually did not force students to adopt specific behaviors while they live coded, many considered the selective release of lecture materials, such as the completed code, to be a way to influence student behavior—for example, withholding the completed code so that typing along during the lecture would be the only way students could obtain it.

Our results can help practitioners decide if and how to use live coding as part of their practice. While we cannot provide a single correct answer, we hope our results provide a richer and more nuanced understanding of live coding practices that can help instructors make decisions about their own particular teaching contexts. Our results also point to future directions for computing education research, such as examining live coding beyond CS1, investigating the effects of live coding on student self-efficacy and confidence, considering ways to augment and improve the efficacy of live coding, and exploring whether typing along with the instructor helps or hinders students.

2 Related Works

2.1 Characterizing Live Coding

2.1.1 Case Studies, Experiments, and Observations. The literature on live coding includes several case studies and experiments that offer a description of how the authors employ live coding in their classrooms. For example, some studies describe live coding as one component of a lecture or lab that is used for demonstrating example problems [16, 55] and some describe live coding as a lead-in activity for a coding exercise [54, 69]. A 2021 literature review by Selvaraj et al. examined 20 papers related to live coding and characterized different live coding practices based on three dimensions: the level of student interaction; whether the live coding was synchronous or asynchronous; and how much of the code was written

before the lecture, if any [52]. While informative, many aspects of live coding lectures remain unclear or under-reported. How much of each lecture is dedicated to live coding? For which courses and topics are instructors using live coding and why? What tools are used to facilitate live coding? In this work, we seek to characterize the variety of ways live coding can be used in the classroom and gain a deeper understanding of the decisions instructors make incorporating live coding in their lectures.

A smaller body of work has tried to understand live coding pedagogy through observation. In one work, the authors analyzed 20 programming videos (not necessarily lecture recordings) to derive design recommendations for a live coding system [10]. In another work, the authors analyzed recordings of live coding lectures for a single class and characterized the live coding portion as having two separate steps: the lead up, where the instructors present concepts, possibly using diagrams or written notes; and an iterative development cycle, where live coding proceeds with a mixture of writing code, explaining the thought process, and debugging [23]. While these studies are valuable, they may not generalize to live coding lectures, as one uses only videos and the other uses observations from a single instructor's lectures. Furthermore, observational studies are limited in that they lack insights from the instructor as to why they made certain decisions.

2.1.2 Interviews with Instructors. A small number of researchers have reported on insights from interviews with instructors who use live coding [3, 33, 63]. The present work differs from these past works in both scale and scope.

In terms of scale, previous interview studies have a relatively limited sample size, for example including five instructors and four TAs [63] or two instructors and seven TAs [3]. Through a larger sample size of instructors, our work seeks to capture more variation in live coding teaching practices.

In terms of scope, this work addresses different research questions than previous interview studies on live coding. In terms of RQ1, while two prior works touch on instructor practices [3, 63], they focus on barriers and motivations rather than on the type of content covered (RQ1A) or how live coding interacts with other lecture activities (RQ1B). In terms of RQ2, one prior work focuses on expected student behaviors [33] (RQ2A), but none dive into how instructors try to influence these behaviors (RQ2B).

2.2 Lectures and Active Learning in CS Education

Lectures are ubiquitous in higher education [41]. As with any other field, CS students can benefit from the structured guidance provided by an expert lecturer [21, 22, 65]. At the same time, the lecture format may allow students to sit and watch passively, which may not be as beneficial to learning as more active methods [12]. Thus, educators commonly try to incorporate more active learning into their lectures [47]. For the remainder of this section, we discuss some of the active learning strategies that are common in CS lectures and that came up in our interviews.

Peer Instruction (PI) is an activity where an instructor presents the students with a multiple-choice question, and then has students (1) think through it individually; (2) discuss with a partner; and then (3) submit their answer [15]. In a CS classroom, these multiple-choice questions might, for example, involve tracing code or solving Parsons problems [59, 73]. PI has been widely studied in the context of computing education, where the evidence suggests that it can be an effective intervention to boost learning and increase student retention and that it can be applied in a variety of contexts, including across the CS curriculum and in both small and large classrooms [42–46, 60]. PI might be seen as a specific formulation of a similar technique called think-pair-share, which can support open-ended questions and may not require students to formally submit their answers [1]. Think-pair-share activities can also be effective in the CS classroom [26, 27].

Instructors may also give students coding exercises during lecture, wherein the student is required to write code by themselves or in small groups [20, 61]. Coding exercise can pair with live coding—for example, the instructor may use live coding to walk through a solution to the coding exercise [56]. Alternatively, instructors may call on individual students to solve a coding exercise in front of the class [18].

In order to make more time for active learning activities, some instructors may use a flipped classroom approach, where students self-study before the lecture and class time is reserved for active learning [30]. Some instructors who use the flipped classroom approach use live coding as their active learning activity during lectures [37, 57]. Instructors may disagree whether or not live coding constitutes an active learning strategy, and as pointed out by Selvaraj et al., it might depend on one's definition of live coding [52].

2.3 Theoretical Frameworks

2.3.1 Cognitive Apprenticeship. The Cognitive Apprenticeship (CA) framework draws upon traditional apprenticeship methods to outline how experts can transfer their knowledge for reasoning-based tasks [14]. The CA framework consists of four dimensions of teaching—content, methods, sequence, and sociology—along with characteristics and guidance within each dimension [14]. In terms of methods, CA outlines a progression of six techniques, the first of which is *modeling*, wherein an expert demonstrates a task while externalizing their thought process through thinking aloud [14].

CA is one of the most commonly cited theoretical frameworks in the live coding literature, where studies often refer to live coding as an example of the modeling phase of CA [52]. Few other works have explored how live coding intersects with other teaching methods of CA or the other dimensions of CA [52]. The present work may shed light on how current live coding practices intersect with other parts of the CA framework, namely the *content* dimension (through RQ1A) and the other teaching techniques in the *methods* dimension (through RQ1B).

2.3.2 Cognitive Load Theory. Cognitive load theory (CLT) posits that instruction places two[1] types of demands on working memory: *intrinsic load*, imposed by the unavoidable complexity of the information to be learned; and *extraneous load*, imposed by unnecessary information such as irrelevant content, distractions, etc. [17]. According to CLT, both types of cognitive load are additive and learning is impeded when the total cognitive load exceeds available mental resources [17].

[1] An older but still used version of the theory includes *germane load* as a third type [17]

Compared to presenting static code in lectures, one study found that live coding reduced extraneous cognitive load [48]. The researchers suggested that by writing code line by line through live coding, the cognitive load on students might decrease as they know exactly where to focus [48]. Another study found no difference in cognitive load when using live coding [68], and another found that the different ways students follow live coding lectures can lead to different demands on cognitive resources [33]. This work can help us understand the variety of teaching practices (RQ1) and student behaviors (RQ2A) in live coding lectures, which may have variable effects on cognitive load and learning.

3 Methods

Table 2: The background information of our participants, including years of teaching experience (YoE) and recent courses taught using live coding and which were discussed in our interviews. We use DSA for "Data Structures and Algorithms" and SE for "Software Engineering."

	YoE	Course(s)
P01	34	Intro Python; Intro Java
P02	7	Intro Python
P03	18	Databases; DSA
P04	16	Intro Data Science
P05	2	Java; Computer Organization
P06	24	Intro Data Science
P07	4	Intro Java
P08	25	Intro Java
P09	0.5	Intro Python
P10	8	Intro MATLAB
P11	10	Data Science
P12	13	Intro Python; C++; DSA
P13	40	DSA
P14	24	SE; Game Development
P15	30	Intro Java
P16	2	SE; Web Development
P17	13	Intro Python; DSA
P18	20	Intro Python; Databases
P19	15	Intro Java; SE; Compilers
P20	36	Data Science; DSA
P21	13	Intro Python; Intro Java; DSA
P22	18	Intro Python; DSA; C++

3.1 Participants

We interviewed 22 instructors who had experience with live coding in their CS lectures. Participants included research-oriented professors ($n = 4$), graduate students with instructor experience ($n = 2$), a high school teacher ($n = 1$), and several other university-level teaching faculty, lecturers, and adjuncts ($n = 15$). Four participants identified as women and the rest ($n = 18$) identified as men. Participants had an average of 16.9 years of teaching experience, ranging from a single semester to 40 years. Our participants' affiliations span 14 institutions, with one institution located in Europe, one in Canada, and the rest in the USA. Information about individual participants can be found in Table 2.

An initial round of 10 participants[2] was recruited through our university's mailing lists and through an open call on social media, and a subsequent round of participants was recruited through mailing lists pertaining to computing education. All participants filled out a screening survey which defined live coding and asked them several questions about their teaching experience, including how often they used live coding, for which courses did they use live coding, and additional optional questions about their live coding practice. The survey allowed us to ensure that we only reached out to eligible participants with live coding experience. While we primarily chose participants at random from our eligible pool, the survey results also allowed us to use purposive sampling [4] to target participants with a wide variety of experiences (e.g., small versus large class sizes).

3.2 Interview Protocol

Interviews were conducted through Zoom and lasted between 45 and 60 minutes. The first 10 interviews were conducted by the first author, and the subsequent 12 interviews were conducted jointly by the first and third authors.

The goal of the interviews was to get a holistic understanding of how the instructor used live coding in their lectures along with the reasoning behind their decisions. The interviews were semi-structured: we roughly followed an interview guide, but asked follow-up questions freely and did not always go through questions in the same order. We generally began the interviews by asking about the instructor's teaching background and about which courses they use live coding in and why. We then asked several questions, including how live coding fit into a typical lecture, what their expectations were for students during live coding, what materials they provided to students, and what they perceived as the benefits and drawbacks of live coding.

The study procedure was approved by our organization's IRB. After the interview, participants were compensated with a $40 gift card for their time.

3.3 Data Analysis

Due to the exploratory nature of our research questions, we opted to analyze our data using Thematic Analysis (TA) [13]. In particular, we used an approach to TA called *template analysis*, wherein a structured, hierarchical codebook is created based on a subset of the data and the codebook is iteratively refined and revised throughout the analysis [6, 25]. As a "codebook TA" approach, template analysis is a pragmatic interpretivist approach that balances structure with flexibility [5, 39].

Our approach follows the steps of template analysis outlined by Brooks et al. [6]. We began by transcribing our interviews using an automated service and then manually correcting the transcripts. Then, the first and second authors independently read through four interviews, taking notes and doing an initial pass at coding. We then came together to discuss and collaboratively define an initial codebook. From there, the first two authors went through multiple iterations of independently coding one or two interviews at a time and then coming together to discuss discrepancies. We note that

[2]These interviews were conducted as part of an earlier study, and were analyzed as part of that work with more narrowly-scoped research questions [33].

Table 3: The finalized codebook used in our thematic analysis, which evolved through several rounds of analysis and discussion. For brevity, we only include three levels of the hierarchy and provide only a selection of the level-3 codes in the right column.

Level-1 Codes	Level-2 Codes	Selected Level-3 Codes
What is being taught?	Courses	Intro Java, intro Python, data science, software engineering
	Context	New concepts, practice problems, examples, misconceptions
	Programming process	Incremental development, debugging, problem-solving
	Programming tools	Debugger, shell, Git, IDEs, browser dev tools
	Positive attitude	Exploration, mistakes
Lecture Fit	Information delivery	Slides, IDE, notebook, whiteboard, code comments
	Class activities	Coding exercises, peer instruction, multiple-choice questions
Ad-hoc Participation	Prompts and questions	What comes next, predict the outcome, intentional mistakes
	Participation trouble	No response, same students respond
	Prompting participation	Cold calling, long pause, raising hands
Expected Student Behaviors	Typing along	Helps learning, exploration, staying on task, cognitive load, errors
	Taking notes	Hand-written, screenshots, code comments
	Course materials	Lecture recordings, finished code, starter code

disagreements typically had less to do with the interpretation of the data and more to do with the definition and boundaries of our codebook. Thus, rather than calculating inter-rate reliability—which is not typically done for Codebook TA methods [5]—we resolved disagreement through discussion and codebook updates. After processing 23% (5/22) of the interviews this way, our codebook reached a level of stability where we were no longer adding new codes to the higher levels of our hierarchy.

The first author then coded the remainder of the interviews and proceeded with additional rounds of analysis that involved going back and forth between the codebook and transcripts to update codes and develop interpretive themes. At this point, updates to codes primarily involved changing their scope or higher-order classification [25]. For example, "*participation*" was originally a subcode of "*expected student behaviors*", but then "*participation*" became its own top-level code (change in higher-order classification) and "*Expected Student Behaviors*" was reinterpreted to refer to non-interactive behaviors like note-taking (change in scope). The themes were developed throughout the process, evolving from descriptive top-level codes (Table 3) into interpretive "storybook" themes [39] (Table 1). Throughout this process, the first author periodically checked in with the second author to discuss updates to the codebook and candidate themes, as is suggested practice for solo coding [51]. The final codebook can be seen in Table 3.

4 Results

We constructed three themes based on our analysis. The first is that live coding is a highly versatile pedagogical tool that can be used to present a wide variety of material and can be used in conjunction with many other lecture techniques. The second is that live coding does not automatically make students active participants, although it does provide several opportunities for participation. The third is that instructors may consider the selective release of lecture materials as a way to influence student behavior (e.g., not releasing their final code to encourage students to type along with them). The three themes are summarized in Table 1, which also lists which sections correspond to each theme.

4.1 What Are We Teaching with Live Coding?

In this section, we explore what instructors teach with live coding, looking not only at the specific subjects, but also examining the context of *what* is being live-coded and what specific skills or concepts instructors hope to teach using live coding.

4.1.1 Live Coding Across the CS Curriculum. Our participants used live coding in a wide variety of courses, including introductory CS courses, data science, data structures and algorithms, databases, software engineering, and other specialized courses, such as mobile app development and game programming. Table 2 shows more information about the courses each participant taught using live coding.

When we asked the instructors if there were courses in which they did not use live coding, the most common response was, unsurprisingly, for courses that do not require very much coding. Notably, some instructors did use live coding in their software engineering (3/22) and algorithms courses (7/22), while others felt those courses were less suited to live coding as they were more about concepts than code (3/22). Live coding was widely seen as appropriate for introductory courses, and over half of the instructors had experience using live coding in that context.

4.1.2 New Concepts vs. Integrative Practice. Our interviews reveal a dichotomy in how one might use live coding. On one end, one might use live coding to introduce new material and concepts, for example, showing many different examples of a for loop and explaining how

it works. On the other end, one might use live coding to work through practice problems, integrating already-learned concepts.

P01 exemplifies the approach of using live coding to introduce new concepts.

> *(P01) Lectures are me live coding, talking them through examples, usually focused on an individual type of programming statement or concept. [...] I'm trying to make a distinction between that and live coding, where you say, let's solve this problem, and then start coding it together. My live coding has mostly been we're going to learn about the for loop today. So let's look at all the possible ways you might tackle it.*

In contrast, P17 felt exactly the opposite, and preferred using live coding only for practice problems.

> *(P17) So I don't think that live coding works very well to introduce a new concept for the first time. [...] Normally, we're not going to do the live coding portion unless students have seen or have already been introduced to the concept somewhere else, because just typing out a construct doesn't do a lot to help other students get a deeper understanding of the concept.*

When P17 wanted to explain a new concept in class, they typically did so using slides, followed by a live coding example that uses the concept.

In practice, instructors might use both approaches, even within the same lecture.

> *(P06) Sometimes it's just, "okay, I just got to show you how to do this." But other times it's more like, "Okay, we've learned the basic concepts here. And so now like here's a problem, how would we attack it?" And so it kind of just depends on the topic and what we happen to be doing at that time. So I would say it varies a lot.*

Even though P01 primarily used live coding to introduce new concepts through live-coded examples, as their students gained more experience towards the end of the course, they would incorporate more practice problems into their lectures. On the other hand, P05 used a flipped classroom model where they used lecture time to walk through practice problems using live coding, but they felt the need to re-teach some of the material while live coding because students did not always complete the readings.

One benefit of introducing new material with live coding is that you can start with a simple example and then make incremental changes to compare variations or gradually introduce more information.

> *(P01) For instance, today I was teaching dictionaries in my Python course, so we just defined one with just a few entries and printed it out. And then we said, "Okay, now what can I do with a dictionary? What is its purpose?" And I, you know, I looked something up and blah blah blah, but it's only a couple lines at a time.*

A similar approach is to start out with an already-written piece of code and then run and modify it in front of the class. We might call this a *live examples* approach—it may not involve as many code changes as doing live coding from scratch, but it is more dynamic than static code on a lecture slide, which cannot be executed or modified. Four instructors described sometimes using this live examples approach, which might be particularly appropriate when discussing a larger piece of code.

4.1.3 Learning from Mistakes. Nearly all our participants (21/22) expressed the idea that mistakes and errors during live coding provided a valuable learning experience for the students. While mistakes could come from student suggestions or accidental errors on the instructor's part, the majority of instructors (13/22) we interviewed would regularly introduce intentional mistakes for pedagogical purposes.

One reason instructors found mistakes valuable was that they offered an opportunity to discuss common issues and misconceptions with students. As P13 expressed, "*often mistakes are more instructive than just students seeing the correct solution at first.*" Instructors could draw on student suggestions as potential sources of authentic mistakes.

> *(P10) I understand 90% of the weird ways students might think about problems. There's always that 10% where a student is like, "Could we do it this way?" I'm like, "you know, I have never thought of that in my entire life, but yeah, let's go down that route and see why it works or why it doesn't work."*
>
> *(P19) I try to follow the student feedback and ideas. Even if I know that they're not going in the best direction, I think that's also important to illustrate.*

Other instructors (5/22) would draw on their knowledge of common misconceptions to show them to students and correct them on the fly.

Instructors also valued that mistakes gave the opportunity to illustrate and teach debugging skills. Live coding can provide the opportunity to debug in an authentic context.

> *(P11) A nice thing about live coding is that you can show students some of the process that you otherwise would not be able to easily show, like debugging or running into natural bugs. It's just less natural without it because it's like, oh, I have to show you, here's a place where you would run into this bug, but it's like from nowhere, right?*
>
> *(P21) Spending a whole lecture on debugging strategies, students don't get it. [...] So instead, you give them enough that when you model it, you go: okay, well I'm going to try to do this style of debugging here.*

Live coding gave instructors the opportunity to model specific debugging skills, including how to interpret error messages and how to use the debugger.

Finally, over half of the instructors we interviewed (13/22) expressed that making mistakes might help students to feel more confident when they make mistakes in their own programming. This confidence could come from both having learned debugging strategies from live coding and from seeing instructors treat making mistakes as a normal and expected part of programming.

> *(P01) I convey that it [making mistakes] happens all the time. It's no big deal.*
>
> *(P15) I try to point out to everybody, I don't write perfect code. We all make mistakes when we're writing code,*

> *and it's more about how you find and remove those mistakes than writing the code perfectly the first time.*
>
> *(P21) We can't only model success. We also need to model failure sometimes.*

Not all mistakes during live coding were seen as productive. For example, P20 felt that when using live coding to show solutions to coding exercises, introducing mistakes could lead to unnecessary confusion. In addition, some instructors felt that if they made frequent errors or could not debug their own mistakes that this could lead to wasted class time or a loss of confidence from the students. Strategies for overcoming those situations include having "solution code" as a backup (5/22) or debugging the error offline and discussing it during the next class (1/22).

4.1.4 Process and Procedures. Many instructors felt live coding was useful for demonstrating the programming process. Through live coding, instructors could demonstrate things like refactoring (2/22), incremental development (3/22), working with existing code bases (1/22), test-driven development (4/22), and debugging (§ 4.1.3). Some instructors felt that it would be difficult to demonstrate these processes another way.

> *(P19) The advantage of doing it live is that you make the process, the way that you're thinking, a bit more explicit, which is difficult to put on static paper.*
>
> *(P21) Quite often, what they get is the final solution code. And so if they're not live in person, they're not seeing that development of the code.*

P05 and P22 both described the process of writing code in C as non-linear, involving jumping back and forth between multiple files and refactoring logic into functions as they went along. P05 found this progression difficult to capture in their own lecture notes, and described their notes as "*more of a mind map*" which would be essentially inscrutable if given to another instructor or the students.

Instructors also used live coding as an opportunity to demonstrate how to use a variety of programming tools in an authentic context. For example, instructors' live coding practices included demonstrations on how to use the debugger (2/22), shell commands (3/22), Git (1/22), the browser's developer tools (1/22), and various features of the IDE (3/22). As P08 said, "*for things that are more environment based, that's a process, and that's something I want to model with you and practice with you so you can do that on your own.*" P14 also provides their own video recordings for these types of demonstrations.

> *(P14) Here's how you make a project. Here's how you connect to GitHub. Here's how you write your first failing test, or here's how you set up Flutter. So those kinds of things, I give them on YouTube. This is something I've shown in class, and there's no way your notes can capture this.*

Finally, instructors felt that live coding gave them the opportunity to model the problem-solving process by thinking aloud while working through a coding problem.

> *(P17) The point of the live coding is... there is the live coding itself, but most of the value in the exercise comes from the commentary that I'm giving to sort of externalize my internal process of how I would work through these problems so that students can see that.*

While P17 felt practice problems were the main point of live coding, other instructors saw it as just one use case (see § 4.1.2).

4.1.5 Fostering Exploration. Some instructors felt that by tinkering and experimenting with code during live coding, they might empower students to adopt a similar mindset and set of strategies.

> *(P02) And so we want them to come away with it with like, "Okay, I can change this so that this happens, I can change that."*
>
> *(P11) I really wanted to foster a sense of creativity around the code. Students should be encouraged to tinker and play around a little bit to help them understand things, and I think showing it frequently throughout class helps with giving them a sense of like, "oh yeah, this is how I would explore this question" rather than just like, turn to ChatGPT or ask a question directly on the discussion board.*

Even without the explicit goal of encouraging tinkering with code, instructors generally had a flexible approach with live coding and were willing to try out different things based on student suggestions (see § 4.3.1).

4.2 How Live Coding Fits within a Lecture

Although some instructors would spend their entire lectures live coding, we found that more often live coding was combined with active learning activities and other methods of information transmission. In this section, we discuss how instructors balance live coding with these other lecture activities, as well as the synergies and frictions between them.

4.2.1 Modes of Information Delivery: Slides, IDEs, Notebooks, Whiteboards, and Comments. Many instructors (14/22) used slides in conjunction with live coding. Some instructors used slides only sparingly, for example, to share course logistics (3/22), visualize select concepts (2/22), or briefly review content at the start of the lecture (2/22). Other instructors spent a more significant portion of their lectures going through slides. One strategy was to switch back and forth from slides to live coding, essentially using live coding to demonstrate the examples from the slides (6/22). This context switching could place an additional cognitive burden on instructors.

> *(P09) I just want to have an easier time balancing the live-coded segments with the slides.*

An alternative strategy reduces context switching by introducing a topic using slides and then using live coding to demonstrate the concept in the context of a longer practice problem or demonstration (6/22). For example, while P22 used slides only sparingly when teaching object-oriented programming, in their DSA course, they would commonly use slides to introduce a data structure and then transition to implementing it with live coding. Although less common, two instructors felt that starting out with live coding and then transitioning to slides could be effective: P12 described using live coding to motivate the need to learn a concept and then switching

to slides; and P16 sometimes used slides as a recap of what students were meant to learn during live coding.

Some instructors primarily used computational notebooks to deliver their lectures. Notebooks were used in every data science or visualization course (4/22) and also for an introductory programming course in MATLAB. Notebooks essentially replace both the lecture slides and code editor: the content that would have gone on slides can go in markdown cells, and code cells can be used to display examples and do live coding. Having both code and markdown in the same interface allowed instructors to present the notebooks linearly, explaining concepts and filling in or modifying code cells without having to switch between different applications.

> *(P11) The nice thing with Jupyter notebooks is I can very much easily sequence the discussion of the syntax with like here's an example in a code cell. And let's try modifying that.*

> *(P20) I feel a Jupyter notebook is just extremely helpful to chunk out what I'm teaching in little pieces.*

Notebooks also facilitated student note-taking, as the instructors would provide students with the template version before the lecture so that they could fill it out with the instructor.

Some instructors (5/22) also used whiteboards and document cameras to convey information during the lecture, though usually separate from live coding. One exception was P14, who taught test-driven development (TDD) using a whiteboard in conjunction with projecting their screen and live coding. P14 would use the whiteboard to write out "task lists" and the three steps of TDD: red, green, and refactor (i.e., writing a test, making the test pass, and refactoring the code [2]). In their words, the task list and TDD steps acted as subgoal labels to describe the TDD process. Notably, this was the only example shared with us of an instructor writing the explicit steps in a process somewhere outside of code comments while live coding.

Finally, while live coding, some instructors would also write down some of the information spoken aloud in code comments. This was often done to record the process, for example, writing down the steps of an algorithm before implementing it (6/22). P14 and P22 were specifically against writing these types of comments, as they felt it did not model standard industry practices like self-documenting code and selective commenting. P13, P16, and P21 all felt there was not enough time during live coding to add comments to their code. Finally, P11, who used notebooks, felt that information about process and concepts was better captured in markdown cells.

4.2.2 Live Coding and Active Learning Activities. Our participants often used live coding in conjunction with other active learning techniques, including coding exercises, think-pair-share activities, and peer instruction or similar collaborative multiple-choice questions.

Sixteen of our participants regularly gave students coding exercises as part of their lectures. Most commonly, instructors would give students short coding exercises similar to what was just demonstrated with live coding. After coding exercises, instructors often used live coding to present solutions to the class, whether presenting their own solutions or asking students to guide them. This was a common point where instructors could lean on student participation.

> *(P22) When I go through my solution, there's probably a slightly higher percentage of questions. I'll try to engage: what did you do? Did you do it differently?*

For coding exercises, instructors sometimes needed to distribute starter code. To do this on the fly, P09 used a system called EdStem while P18 opted for Pastebin. Some instructors had students download template computational notebook files before class (5/22), some had students type along with them as they built up the starter code (2/22), and others did not need to distribute starter code at all. P05 cited the lack of efficient code-sharing options as a barrier to conducting in-class exercises.

Six of our instructors had incorporated peer instruction or similar multiple-choice questions as part of their live coding lectures. P15 felt that live coding helped provide context for peer instruction.

> *(P15) It's not like I put up a random Boolean expression with, you know, x, y, and z variables that they don't even have any idea what they represent, right? I just pulled something live out of what we just wrote, and they already know what it means because they heard me narrate it. They saw me write it, they saw the code run. And now I'm asking them which expression is equivalent, which one would do the same thing?*

Similar to the coding exercises, instructors could go over solutions using live coding to demonstrate the right and wrong answers.

> *(P02) if most people got it wrong, then I can stop in the middle of that and explore that concept and show them how it works.*

In this way, live coding can be used to transition both into and out of other active learning activities.

4.3 (Missed) Opportunities for Participation in Live Coding

Instructors often perceived live coding to be an interactive activity rather than an instructor-led monologue.

> *(P03) It's a conversation: I do something and ask them questions and hear from them and so on. So it's like all the time we're talking together.*

Live coding seems to offer several touch-points for student participation, though instructors often wish students were more actively engaged.

4.3.1 Opportunities for Participation. Almost all instructors we interviewed (21/22) would commonly solicit student input while live coding. One way to do so was to ask students what step should be taken next (13/22). While this could sometimes just serve to keep students cognitively engaged, it could also be used to steer the lecture, as an instructor can adjust their code based on student suggestions. P01 felt that having students shape the lecture "*makes the lecture so much more meaningful.*"

Instructors described taking students' suggestions, even when they were not the ideal way of doing things.

> *(P15) Even if there are competing suggestions and I know one of them is better, often I will take the naive*

> *suggestion, and rewrite it so that we can talk about what doesn't work about it, or how can we improve it.*

Thus, taking suggestions may solicit authentic student misconceptions or mistakes and offer a learning opportunity to correct them. P19 described taking a similar approach with student suggestions, but felt the improvisational approach was more appropriate for advanced courses, whereas in introductory courses you risk "*losing the audience*." In a similar vein, P10 mentioned having to take care in their introductory course to not overwhelm students when showing alternative solutions.

Ten of the instructors felt that making mistakes (intentional or not) during live coding was an effective way to increase student engagement.

> *(P06) I'll purposely throw a wrench in the works there to force some interaction.*

> *(P22) Every now and then, I'll drop something, and that's when they're more likely to be engaged. If I can't find a syntax error, we'll sort of collaboratively work on it.*

Thus, collaborative debugging provides another opportunity for student participation.

Finally, another strategy used by some instructors (4/22) was to ask students to predict what would happen when the code is executed.

> *(P10) We'll talk about it even before we run it. Like, who thinks this is right? Who thinks this is not right? And then we'll have debates or discussions before we even run anything.*

Asking students what line of code comes next can be another type of prediction, as students can compare what they thought should come next with what the instructor ends up writing.

4.3.2 Improving Participation. Despite the many opportunities for student interaction afforded by live coding, most instructors (14/22) struggled with student participation. Ten instructors explicitly mentioned that student participation during live coding was low and they wanted to improve it, and five instructors felt participation was limited to a small, self-selecting group of students.

The level of participation may be affected by variables like the class size and course level.

> *(P05) I think a lot of it is the size of the classes I teach. A lot of my lectures are 200 to 250 people, so people do not want to speak up and be wrong. [...] It's better in the higher-level class I'm teaching, I think, because I have more people who care more.*

These participation dynamics may be largely unrelated to live coding. As P01 said when discussing unequal participation, *"there's nothing magic about live coding that changes this, but there are some students who are more engaged than others."*

Instructors shared a variety of approaches to soliciting student participation, such as taking long pauses after asking questions (2/22), having students raise their hands to vote instead of speaking (1/22), or directly asking questions to a single student or a small group of students (4/22). P12 described an activity where they divided the class into rows and rotated through each row of students to ask them to provide the next line of code. P10 experimented with a "random student picker" to cold call on individuals, but ultimately abandoned that approach because of negative feedback from students.

Even though many struggled with student participation during live coding, some still made a point to mention that live coding was more engaging than other alternatives. For example, P04 felt they received more student questions when using live coding than when using slides. A similar sentiment was shared by P01, who said "*the slide approach is just deadly.*"

4.4 What Should Students do during Live Coding?

During the interviews, we asked instructors what they thought students should be doing when the instructor was live coding. While few instructors required that students engage in one specific way or another, most still held views on whether students ought to be typing along, taking notes, or just paying attention.

4.4.1 Typing Along and Taking Notes. Instructors had widely varying opinions on whether or not students should type along with them to copy down the instructor's code. Half of the instructors liked or preferred that students typed along with them while they live-coded; five instructors explicitly did not want their students to type along; and the remaining six expressed no preference. Instructors felt that the benefits of typing along were: that the physical act of typing along might help students learn better (8/22), that it could encourage experimentation with the code (3/22), that it could help students train their typing skills (1/22), and that it could be engaging and help students stay on task (3/22).

The most commonly mentioned drawback of students typing along was that it could lead to cognitive overload, distracting students from learning and putting them at risk of falling behind (8/22).

> *(P05) I'd rather you understand why I'm typing and what I'm typing versus frantically copying it down.*

> *(P11) I think the idea that you're copying down code when you're trying to take in these multiple streams of information can be really overloading.*

Some instructors (3/22) still felt that typing along could be worth it—for example, P01 felt that typing along would slow students down, but that this might help them examine the code in more detail, "*dissecting it line for line.*" On the other hand, P14 felt that the effort students put into typing along could give them a false sense of accomplishment.

> *(P14) What I see them do is follow along and think because they follow along, they understand it. But that's obviously false to me.*

P14 highlights that the final artifact—working code—can feel like an accomplishment, when it is possible to copy down the code without actually understanding the code or learning from it.

Another drawback of typing along was that students might make errors in transcription (4/22). This could mean that they fall behind while they try to debug, or they might waste class time as they ask the instructor to help them debug or to scroll back to earlier parts of the code to check it against their copied version.

> *(P04) One challenge of live coding is like when one of your students has a problem, and because you are taking a class, you can't go to that student and solve that particular problem.*
>
> *(P05) it just gets kind of disjointed when someone's like "can you scroll up to line 13" while I'm trying to talk about line 97 or whatever, because they missed an ampersand on line 13.*

Even though the students' transcription errors could be distracting, some instructors still welcomed them as learning opportunities.

> *(P02) It's an act of learning, of like, oh, I forgot my parentheses, or I, you know, need to make sure that I have spacing correct, and things like that.*
>
> *(P06) We'll just do it [debug the error] right there in the middle just because again, I feel like it's useful for all the students to see what mistakes crop up and how to recover from them.*

Notably, P02 and P06 were teaching with Python, which may lend itself to fewer transcription errors than languages like Java or C with trickier syntax.

In addition to just typing along, many instructors (9/22) felt it would be beneficial for students to take notes. With the exception of P14 and P08, instructors did not have a specific preference for how students should go about taking notes. P14 preferred his students take hand-written notes, citing results from learning science: "*[you should be] rewriting the notes that are your ideas, and putting those notes in conversation with the course topics.*" P08 had a specific note-taking strategy they asked their students to adopt that made use of screenshots from the instructor's code, which was live-streamed to each student's computer. Essentially, they had their students keep notes in a Google Doc that consisted of screenshots of the instructor's code annotated with textual explanations. They felt the benefit of their approach was that (1) it forced students to step back and explain the code in plain English; and (2) it allowed students to easily come back to and interpret their notes later. While P08 did not bar other note-taking methods, such as handwritten notes, they still required their students to keep digital notes in addition.

4.4.2 Influencing Student Behaviors through the Availability of Course Materials. If instructors wanted students to adopt a specific behavior while they live coded (e.g., typing along with them) they most typically would just directly suggest to students that they do so. A small number of instructors had a way to enforce participation: P20 asked students to turn in their code from lecture for a participation grade (and P06 wished for an easy way to do so); and P08 required their students to keep their course notes in a Google Doc. But more commonly, instructors would try to influence student behaviors through the selective release of course materials such as lecture notes or recordings.

Instructors often reasoned about releasing course materials in terms of the effects on their lectures. For example, we can consider the completed code that the instructor writes during a live coding session. Releasing the completed code after the lecture could be seen as disincentivizing student attendance and classroom engagement (6/22). But at the same time, students having access to the code could be positive, as it could make them less preoccupied with getting everything exactly right when typing along with the instructor (3/22). While most instructors released their completed code, many expressed ambivalence.

> *(P01) I'm of two minds on that issue. Because in my larger classes, where I'm not doing attendance every day, there's nothing that forces them to be in class. And me giving them the example after I do it encourages them to think "Well, I'll just get it after class and I don't have to show up." So that's the downside. [...] And yet to say I'm not going to provide the example we did in class afterwards... that's not really a reasonable position to take either.*
>
> *(P06) Maybe after a couple of lectures, like when we finish a topic, then I'll publish the sort of solution notebooks. With the idea that if they messed up their notes or if they didn't follow along or they missed something, they could always go back and see it. But sometimes I feel like when I do that, that just causes them to not feel like they need to follow along. They'll just be like 'Oh, I'll just wait till the solutions come out.' So I go back and forth on that.*

P22 approached this tradeoff by giving the completed code to students in their more advanced courses, but not the introductory courses, explaining that *"I think part of the difference here is the message to the freshman about paying attention and being engaged instead of being on their phones."*

The five instructors who used notebooks described two versions of each notebook: the "skeleton", which was missing some of the code and was presented to the class; and the completed "solution" notebook that contained the code written in lecture. Four of the five instructors provided the skeleton notebook and suggested that students fill it out with them during class. On the other hand, P11 tried to limit access to the skeleton notebook to discourage typing along.

> *(P11) I think that when I distribute the code [notebook] to students, they feel the need to copy down what I'm writing [...] So my default for this quarter is I do not distribute the code to students by default. If they really want it, if they find that it's very helpful to have the editable notebook in front of them, they can get it. It's just an extra step that they have to ask for that.*

The instructors usually provided the solution notebook *after* the lecture, as they thought that if students had access to the solution notebook during lecture, they would be less likely to type along. Despite this concern, P04 felt that they should make the solution notebook available before each lecture for accessibility reasons.

> *(P04) I usually suggest my students to download those empty [skeleton] notebooks, and I announce in the class, these instructor [solution] notebooks are only for those students who have issues visualizing what's on the projector. So if you are struggling with copying the code, you have the instructor's notebook.*

While P20 did not release the solution notebooks before lectures, they had colleagues who did.

Finally, some instructors (5/22) provided students with lecture recordings, which have the benefit of completely capturing the progression of code in a live coding session. More commonly, instructors were hesitant to record their lectures because they felt it would negatively impact attendance (10/22). To counteract the effect on attendance, P10 adopted a strategy where they only released each day's lecture recording if more than half the class was present. P12 released lecture recordings, but not the completed code from their lectures, as they wanted students who missed the lecture to physically type in the code instead of copy-and-pasting it (their live coding was often directly related to homework assignments).

5 Discussion

Our work offers new insights into the dynamics of live coding lectures that are not captured by other works characterizing live coding [3, 33, 52, 63]. For example, we highlight the ambivalence instructors may feel about releasing course materials like completed code, as they balance their desire for students to have reliable lecture notes with their desire for students to actively engage during lecture by typing along or taking their own notes (§ 4.4.2). Previous work examines live coding in the context of projects or examples [50, 52, 68], but we found that some instructors use live coding to introduce new material, sometimes forgoing lecture slides altogether (§ 4.1.2). While prior work has examined how coding exercises interact with live coding [56], we found that live coding can also be paired with multiple-choice questions or peer instruction, both to frame the activity and to walk through solutions (§ 4.2.2). Finally, while other studies have found that instructors value live coding for its adaptability in responding to students [3, 63], our findings suggest that this ideal does not always hold up in practice, as cultivating student participation can be difficult (§ 4.3.2).

For the remainder of this section, we discuss the implications of our findings for three stakeholder groups: CS instructors, computing education researchers, and designers of educational technology systems. We also discuss the limitations of our study.

5.1 Implications for Instructors

There is no one-size-fits-all approach to teaching; effective pedagogy requires instructors to adapt to their particular classroom context [58]. Instructors can look to our results to better understand the details of how and why others incorporate live coding into their pedagogical practice, which may help instructors to reflect on or improve their own teaching. To that end, we discuss below how our results add nuance to what is known about the benefits and drawbacks of live coding.

5.1.1 Perceived Benefits of Live Coding. Our study highlights many potential benefits and uses of live coding that have been previously discussed in the literature, including that live coding demonstrates the coding process [18, 49, 52], that it helps teach debugging [18, 40, 52], and that it teaches students how to apply programming concepts in context [40, 52]. Our results add depth and nuance to some of these benefits. For example, live coding is often perceived as increasing student engagement [16, 40, 50, 52]; however, our results suggest that live coding is not a magic bullet for engagement, but rather a source of potential participation opportunities to be cultivated through an instructor's pedagogical tactics (e.g., asking for predictions, collective debugging). Our results also highlight that demonstrating errors in live coding is not only useful for teaching debugging, but can also be an opportunity for instructors to introduce and correct common misconceptions in an authentic context (§ 4.1.3). Additionally, instructors may keep in mind the potential affective benefits of live coding by showing students that errors are normal parts of the programming process (§ 4.1.3) and cultivating an exploratory and creative attitude towards programming (§ 4.1.5).

Through our analysis, we identified another benefit of live coding that is not widely discussed in the literature: its *versatility* [52]. Live coding can be adapted to many different styles of instruction: instructors can use a flipped classroom approach [30] and spend part of the lecture using live coding to demonstrate practice problems [37] (§ 4.1.2); instructors can introduce new concepts using slides and then show examples with live coding (§ 4.1.2); instructors can use computational notebooks to intersperse content with short live coding examples (§ 4.2.1); or instructors can use live coding to transition in or out of other active learning activities (§ 4.2.2). Additionally, live coding may be particularly fitting for adapting lectures to address student questions (§ 4.3.1), as instructors can use live coding to test "what if" questions or construct demonstrative examples on the fly.

5.1.2 Mitigating Drawbacks of Live Coding. Our study highlights how instructors might mitigate some known drawbacks of live coding. For example, it is known that live coding can make it difficult for students to take notes [16, 55, 62]. This drawback may be less problematic if instructors do not introduce new concepts using live coding (§ 4.1.2) and instead rely on slides, notebooks, or the whiteboard to introduce new material (§ 4.2.1). Alternatively, instructors might suggest an approach using code snapshots with annotations, like P08 asked their students to do during their lectures (§ 4.4.1).

Another common challenge in live coding is that it can be difficult for students to keep up with the pace of the lecture [16, 52, 55]. Instructors may partially address this by asking students not to type along or, alternatively, asking them to type along, but reassuring them that they will have access to the completed code after the lecture, in case they run into errors or cannot keep up (§ 4.4.2).

Lastly, live coding can be time-consuming as a mode of lecture delivery [40, 52, 68]. This effect may be compounded for instructors who are not as comfortable with the material and have difficulty recovering from mistakes (§ 4.1.3). These difficulties may be partially addressed by keeping backup code or by using the live examples approach, making smaller modifications to prewritten code (§ 4.1.2).

5.2 Implications for Computing Education Research

5.2.1 Theoretical Framing of Live Coding. Our results highlight a variety of practices in terms of *what* is being live coded during lectures. In this section, we briefly consider how this variation in context can inform how we apply two theories commonly applied to live coding: cognitive apprenticeship (CA) and cognitive load theory (CLT) [52].

Consider the case where an instructor uses live coding to introduce new concepts (§ 4.1.2). If the instructor's primary goal is to teach concepts and facts, rather than procedural knowledge, then CA might not be as directly applicable [14]. On the other hand,

CLT may help us reason about how the pacing and modality of live coding might influence cognitive load and affect student learning [64]. Researchers have found mixed results comparing student cognitive load during instruction using live coding versus static code [48, 68]—future work may examine how contextual factors, such as student prior knowledge, might influence these results.

We might also consider the case where the instructor's main goal is to teach a procedure or process (§ 4.1.4). CLT may still be relevant—especially for well-defined procedures—but it does not directly address how to solicit tacit knowledge. In contrast, CA can provide insights into the types of tacit knowledge we hope to impart, as well as guidance on the instructional methods that can be used alongside live coding to do so [14].

5.2.2 Contextualizing Past Work and Exploring Future Directions. Several recent studies have found no learning gains from live coding compared to presenting static code examples [16, 53, 55, 69]. Although our study shows live coding is used across the CS curriculum (§ 4.1.1), the above studies were conducted in the context of introductory programming courses using either Python [53, 55, 69] or Java [16]. The efficacy of live coding in upper-level CS courses remains underexplored. Additionally, none of the above works controlled for student behavior, leaving open the question of whether students should type along during live coding lectures (§ 4.4.1). Our study highlights the variation in how instructors operationalize live coding in their lectures, and future work can examine whether some of these variations might lead to better results.

While our study (§ 4.1.4) and previous work [52] highlight how live coding is valued for demonstrating process, recent studies suggest that just observing the programming process via live coding may not be enough to alter students' programming behaviors [53, 55]. Might there be a way to improve live coding to help communicate and teach processes and procedures? Consider P14's approach to teaching test-driven development (TDD), where they would live code an example while at the same time writing out on the whiteboard (1) the three steps of TDD, and (2) a task list (§ 4.2.1). Regarding (1), writing the steps of TDD on the board can be seen as scaffolding to help students connect the live coding demonstration with a more abstract process. Loksa et al. have shown how a similar metacognitive scaffold describing the problem-solving process can help students in an asynchronous programming task [28], but future work might explore the role of this type of scaffolding in live coding lectures. Regarding (2), the task list can be seen as subgoal labels [9] for the programming task, an approach that can be beneficial for students when learning from worked examples [34–36, 38]. While live coding, instructors may already employ subgoal labeling while thinking aloud through a problem (§ 4.1.4) or when outlining their process in code comments (§ 4.2.1). Future research might explore how subgoal labeling intersects with live coding, for example, examining the efficacy of speaking aloud subgoal labels compared to writing them in code comments or on a whiteboard.

Finally, our study highlights potential psychological benefits of live coding, including that it might make students more comfortable with making mistakes (§ 4.1.3) and that it might help foster an exploratory and creative attitude around programming (§ 4.1.5). Future work may explore whether live coding can foster a sense of belonging in students, increase student self-efficacy, or help students develop a growth mindset.

5.3 Implications for System Designers

Our results can help inform the design of future systems that can facilitate or improve CS lectures that use live coding. For example, we found that many of the touchpoints for student participation in live coding may not be fully taken advantage of (§ 4.3.1). Thus, future work may consider how to solicit student participation during live coding in a scalable way, perhaps allowing all students to participate rather than just the ones most comfortable speaking out during lecture (§ 4.3.2). While much work has examined how to scale programming exercises in a live lecture context [11, 66, 72], designers may consider how they can solicit student input in response to the lightweight, ad hoc questions that arise during live coding (§ 4.3.1).

Another opportunity is to design a system that helps students take more effective notes during live coding. While this is the focus of one recent study, that work examines live coding lectures that focus on introducing new concepts rather than imparting lessons about the programming process [33]. If instructors are trying to teach effective programming processes, students may benefit from some other representation that better encodes the progression of code over time, especially as some instructors felt this was not feasible to capture in student notes (§ 4.1.4). Designers might explore different ways to represent code changes over time, such as version trees [24, 32] or timelines [31, 71], that might be more amenable to reflecting on process.

Finally, while many of our instructors found computational notebooks to be an effective choice for presenting live coding lectures (§ 4.2.1), not all programming languages are well supported in the notebook ecosystem (e.g., Java or C++). Wang et al. created Colaroid, a computational notebook where code cells support a multi-file snapshot of a coding project that links to an IDE [67]. While Colaroid was designed for asynchronous tutorials, designers may wish to explore how a similar system might be useful for supporting live coding instructors.

5.4 Limitations

Our work is exploratory and interpretive, so while our results may provide a framework for characterizing how and why live coding is used in the CS classroom, we cannot determine the frequency of any of the strategies or opinions presented. Our participants were largely US-based, which may influence our instructors' experiences and opinions. We did not observe our participants' actual lectures, which could make it difficult at times to understand exactly the type of content that live coding was used to teach. Our analysis focused on characterizing practices broadly, though future work may explore how each instructor's live coding practice is influenced by their underlying pedagogical philosophies and beliefs [29]. Finally, our results only reflect the instructor perspective, and not that of the students. We lack information on the efficacy of the different lecture strategies and how students perceive them.

6 Conclusion

In this work, we interviewed 22 instructors who use live coding in order to gain a better understanding of how live coding is operationalized in general practice. We found that live coding is highly versatile: it is used across the CS curriculum to introduce and apply concepts and to demonstrate the coding and problem-solving process. We found instructors used live coding in conjunction with other modes of lecture delivery like slides and whiteboards, and that computational notebooks were a convenient option for instructors when available. Instructors valued live coding for providing many opportunities for student engagement, but at the same time, instructors found student participation to be lower than they would have liked. Finally, we found that instructors had divergent opinions on whether students should type along with them while they live coded, and instructors viewed the selective release of course materials, like the completed code from lecture, as an opportunity to nudge students towards desired in-class behaviors. Our results can help practitioners understand the many ways they can employ live coding in their lectures. Additionally, our study points to future directions for computing education research, including studying live coding beyond CS1, examining the effects of live coding on student affect, measuring whether typing along with the instructor affects student learning, and exploring how to augment live coding practices to better impart procedural knowledge.

Acknowledgments

This work was supported in part through an instructional grant from the Center for Excellence in Teaching and Learning (CETL) at Virginia Tech.

References

[1] Elizabeth F Barkley, Claire H Major, and K Patricia Cross. 2014. *Collaborative learning techniques: A handbook for college faculty.* John Wiley & Sons.

[2] Kent Beck. 2022. *Test driven development: By example.* Addison-Wesley Professional.

[3] Caroline Berger, David Weintrop, and Niklas Elmqvist. 2025. "I Feel Like I'm Teaching in a Gladiator Ring": Barriers and Benefits of Live Coding in Classroom Settings. doi:10.48550/arXiv.2504.02585 arXiv:2504.02585 [cs].

[4] Ann Blandford, Dominic Furniss, and Stephann Makri. 2016. *Qualitative HCI research: Going behind the scenes.* Morgan & Claypool Publishers.

[5] Virginia Braun and Victoria Clarke. 2021. One size fits all? What counts as quality practice in (reflexive) thematic analysis? *Qualitative Research in Psychology* 18, 3 (July 2021), 328–352. doi:10.1080/14780887.2020.1769238 Publisher: Taylor & Francis Ltd.

[6] Joanna Brooks, Serena McCluskey, Emma Turley, and Nigel King. 2015. The Utility of Template Analysis in Qualitative Psychology Research. *Qualitative Research in Psychology* 12, 2 (April 2015), 202–222. doi:10.1080/14780887.2014.955224

[7] Neil C. C. Brown and Greg Wilson. 2018. Ten quick tips for teaching programming. *PLOS Computational Biology* 14, 4 (04 2018), 1–8. doi:10.1371/journal.pcbi.1006023

[8] Russel E. Bruhn and Philip J. Burton. 2003. An approach to teaching Java using computers. *ACM SIGCSE Bulletin* 35, 4 (Dec. 2003), 94–99. doi:10.1145/960492.960537

[9] Richard Catrambone. 1998. The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General* 127, 4 (Dec. 1998), 355–376. doi:10.1037/0096-3445.127.4.355

[10] Charles H. Chen and Philip J. Guo. 2019. Improv: Teaching Programming at Scale via Live Coding. In *Proceedings of the Sixth (2019) ACM Conference on Learning @ Scale (L@S '19).* Association for Computing Machinery, New York, NY, USA, 1–10. doi:10.1145/3330430.3333627

[11] Yan Chen, Walter S. Lasecki, and Tao Dong. 2021. Towards Supporting Programming Education at Scale via Live Streaming. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW3, Article 259 (Jan. 2021), 19 pages. doi:10.1145/3434168

[12] Michelene T. H. Chi and Ruth Wylie. 2014. The ICAP Framework: Linking Cognitive Engagement to Active Learning Outcomes. *Educational Psychologist* 49, 4 (Oct. 2014), 219–243. doi:10.1080/00461520.2014.965823

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

[14] Allan Collins, John Seely Brown, Ann Holum, et al. 1991. Cognitive apprenticeship: Making thinking visible. *American educator* 15, 3 (1991), 6–11.

[15] Catherine H Crouch and Eric Mazur. 2001. Peer instruction: Ten years of experience and results. *American journal of physics* 69, 9 (2001), 970–977.

[16] Hanxiang Du, Dion Udokop, and Bo Pei. 2025. Live Coding Prompts Engagement, But Not Necessarily Grades. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025).* Association for Computing Machinery, New York, NY, USA, 283–289. doi:10.1145/3641554.3701794

[17] Rodrigo Duran, Albina Zavgorodniaia, and Juha Sorva. 2022. Cognitive Load Theory in Computing Education Research: A Review. *ACM Trans. Comput. Educ.* 22, 4 (Sept. 2022), 40:1–40:27. doi:10.1145/3483843

[18] Alessio Gaspar and Sarah Langevin. 2007. Restoring "coding with intention" in introductory programming courses. In *Proceedings of the 8th ACM SIGITE conference on Information technology education (SIGITE '07).* Association for Computing Machinery, New York, NY, USA, 91–98. doi:10.1145/1324302.1324323

[19] Nasser Giacaman. 2012. Teaching by Example: Using Analogies and Live Coding Demonstrations to Teach Parallel Computing Concepts to Undergraduate Students. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum.* 1295–1298. doi:10.1109/IPDPSW.2012.158

[20] Byron S Gottfried. 1997. Teaching computer programming effectively using active learning. *age* 2, 1 (1997), 1–8.

[21] Mark Guzdial. 2015. What's the best way to teach computer science to beginners? *Commun. ACM* 58, 2 (2015), 12–13.

[22] Robin K Hill and Mark Guzdial. 2019. Pondering variables and direct instruction. *Commun. ACM* 62, 4 (2019), 6–6.

[23] Derek Hwang, Vardhan Agarwal, Yuzi Lyu, Divyam Rana, Satya Ganesh Susarla, and Adalbert Gerald Soosai Raj. 2021. A Qualitative Analysis of Lecture Videos and Student Feedback on Static Code Examples and Live Coding: A Case Study. In *Proceedings of the 23rd Australasian Computing Education Conference (ACE '21).* Association for Computing Machinery, New York, NY, USA, 147–157. doi:10.1145/3441636.3442317

[24] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17).* Association for Computing Machinery, New York, NY, USA, 1265–1276. doi:10.1145/3025453.3025626

[25] Nigel King. 2004. Using templates in the thematic analysis of text. *Essential guide to qualitative methods in organizational research* 256 (2004).

[26] Aditi Kothiyal, Rwitajit Majumdar, Sahana Murthy, and Sridhar Iyer. 2013. Effect of think-pair-share in a large CS1 class: 83% sustained engagement. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (San Diego, San California, USA) *(ICER '13).* Association for Computing Machinery, New York, NY, USA, 137–144. doi:10.1145/2493394.2493408

[27] Aditi Kothiyal, Sahana Murthy, and Sridhar Iyer. 2014. Think-pair-share in a large CS1 class: does learning really happen?. 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, 51–56. doi:10.1145/2591708.2591739

[28] Dastyni Loksa, Amy J. Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. 2016. Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems.* ACM, San Jose California USA, 1449–1461. doi:10.1145/2858036.2858252

[29] Julie Luft and Gillian Roehrig. 2007. Capturing Science Teachers' Epistemological Beliefs: The Development of the Teacher Beliefs Interview. *Electronic Journal of Science Education* 11 (Jan. 2007).

[30] Mary Lou Maher, Celine Latulipe, Heather Lipford, and Audrey Rorrer. 2015. Flipped Classroom Strategies for CS Education. 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, 218–223. doi:10.1145/2676723.2677252

[31] Mark Mahoney. 2023. Storyteller: Guiding Students Through Code Examples. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023).* Association for Computing Machinery, New York, NY, USA, 1131–1135. doi:10.1145/3545945.3569843

[32] Daniel Manesh, Douglas Bowman Jr., and Sang Won Lee. 2024. SHARP: Exploring Version Control Systems in Live Coding Music. In *Proceedings of the 16th Conference on Creativity and Cognition* (Chicago, IL, USA) *(C&C '24).* Association for Computing Machinery, New York, NY, USA. doi:10.1145/3635636.3656195

[33] Daniel Manesh, Tong Wu, Yan Chen, and Sang Won Lee. 2025. Understanding and Improving Student Note-Taking in Live Coding Lectures. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V.1 (ICER '25).* Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3702652.3744224

[34] Lauren E. Margulieux, Richard Catrambone, and Mark Guzdial. 2016. Employing subgoals in computer programming education. *Computer Science Education* 26, 1 (Jan. 2016), 44–67. doi:10.1080/08993408.2016.1144429

[35] Lauren E. Margulieux, Mark Guzdial, and Richard Catrambone. 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the ninth annual international conference on International computing education research (ICER '12)*. Association for Computing Machinery, New York, NY, USA, 71–78. doi:10.1145/2361276.2361291

[36] Lauren E. Margulieux, Briana B. Morrison, and Adrienne Decker. 2020. Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education* 7, 1 (May 2020), 19. doi:10.1186/s40594-020-00222-7

[37] Samim Mirhosseini, Austin Z. Henley, and Chris Parnin. 2023. What Is Your Biggest Pain Point? An Investigation of CS Instructor Obstacles, Workarounds, and Desires. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 291–297. doi:10.1145/3545945.3569816

[38] Briana B. Morrison, Lauren E. Margulieux, Barbara Ericson, and Mark Guzdial. 2016. Subgoals Help Students Solve Parsons Problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, Memphis Tennessee USA, 42–47. doi:10.1145/2839509.2844617

[39] Aadarsh Padiyath and Tamara Nelson-Fromm. 2026. *Reflecting on Thematic Analysis in Computer Science Education Research: A Field Guide for Researchers and Reviewers*. Association for Computing Machinery, New York, NY, USA, 790–796. https://doi.org/10.1145/3770762.3772512

[40] John Paxton. 2002. Live programming as a lecture technique. *Journal of Computing Sciences in Colleges* 18, 2 (Dec. 2002), 51–56.

[41] Rob Phillips. 2005. Challenging the primacy of lectures: The dissonance between theory and practice in university teaching. *Journal of University Teaching and Learning Practice* 2, 1 (2005), 1–12.

[42] Leo Porter, Cynthia Bailey Lee, and Beth Simon. 2013. Halving fail rates using peer instruction: a study of four computer science courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 177–182. doi:10.1145/2445196.2445250

[43] Leo Porter, Cynthia Bailey Lee, Beth Simon, and Daniel Zingaro. 2011. Peer instruction: do students really learn from peer discussion in computing?. In *Proceedings of the Seventh International Workshop on Computing Education Research* (Providence, Rhode Island, USA) *(ICER '11)*. Association for Computing Machinery, New York, NY, USA, 45–52. doi:10.1145/2016911.2016923

[44] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. 2016. A Multi-institutional Study of Peer Instruction in Introductory Computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 358–363. doi:10.1145/2839509.2844642

[45] Leo Porter, Saturnino Garcia, John Glick, Andrew Matusiewicz, and Cynthia Taylor. 2013. Peer instruction in computer science at small liberal arts colleges. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (Canterbury, England, UK) *(ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 129–134. doi:10.1145/2462476.2465587

[46] Leo Porter and Beth Simon. 2013. Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 165–170. doi:10.1145/2445196.2445248

[47] Michael Prince. 2004. Does active learning work? A review of the research. *Journal of engineering education* 93, 3 (2004), 223–231.

[48] Adalbert Gerald Soosai Raj, Pan Gu, Eda Zhang, Arokia Xavier Annie R, Jim Williams, Richard Halverson, and Jignesh M. Patel. 2020. Live-coding vs Static Code Examples: Which is better with respect to Student Learning and Cognitive Load?. In *Proceedings of the Twenty-Second Australasian Computing Education Conference (ACE'20)*. Association for Computing Machinery, New York, NY, USA, 152–159. doi:10.1145/3373165.3373182

[49] Adalbert Gerald Soosai Raj, Jignesh M. Patel, Richard Halverson, and Erica Rosenfeld Halverson. 2018. Role of Live-coding in Learning Introductory Programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research (Koli Calling '18)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/3279720.3279725

[50] Marc J. Rubin. 2013. The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 651–656. doi:10.1145/2445196.2445388

[51] Johnny Saldaña. 2013. *The coding manual for qualitative researchers* (2. ed ed.). SAGE Publ, Los Angeles, Calif.

[52] Ana Selvaraj, Eda Zhang, Leo Porter, and Adalbert Gerald Soosai Raj. 2021. Live Coding: A Review of the Literature. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 164–170. doi:10.1145/3430665.3456382

[53] Anshul Shah, Vardhan Agarwal, Michael Granado, John Driscoll, Emma Hogan, Leo Porter, William Griswold, and Adalbert Gerald Soosai Raj. 2023. The Impact of a Remote Live-Coding Pedagogy on Student Programming Processes, Grades, and Lecture Questions Asked. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 533–539. doi:10.1145/3587102.3588846

[54] Anshul Shah, Fatimah Alhumrani, William G. Griswold, Leo Porter, and Adalbert Gerald Soosai Raj. 2024. A Comparison of Student Behavioral Engagement in Traditional Live Coding and Active Live Coding Lectures. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 513–519. doi:10.1145/3649217.3653537

[55] Anshul Shah, Emma Hogan, Vardhan Agarwal, John Driscoll, Leo Porter, William G. Griswold, and Adalbert Gerald Soosai Raj. 2023. An Empirical Evaluation of Live Coding in CS1. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23, Vol. 1)*. Association for Computing Machinery, New York, NY, USA, 476–494. doi:10.1145/3568813.3600122

[56] Anshul Shah, Thomas Rexin, Fatimah Alhumrani, William G. Griswold, Leo Porter, and Gerald Soosai Raj. 2025. An Empirical Evaluation of Active Live Coding in CS1. *ACM Trans. Comput. Educ.* 25, 3 (Aug. 2025), 34:1–34:33. doi:10.1145/3743686

[57] Amy Shannon and Valerie Summet. 2015. Live coding in introductory computer science courses. *Journal of Computing Sciences in Colleges* 31, 2 (Dec. 2015), 158–164.

[58] Lee S Shulman. 1986. Those who understand: Knowledge growth in teaching. *Educational researcher* 15, 2 (1986), 4–14.

[59] Beth Simon, Michael Kohanfars, Jeff Lee, Karen Tamayo, and Quintin Cutts. 2010. Experience report: peer instruction in introductory computing. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA) *(SIGCSE '10)*. Association for Computing Machinery, New York, NY, USA, 341–345. doi:10.1145/1734263.1734381

[60] Beth Simon, Julian Parris, and Jaime Spacco. 2013. How we teach impacts student learning: peer instruction vs. lecture in CS0. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 41–46. doi:10.1145/2445196.2445215

[61] Adalbert Gerald Soosai Raj, Jignesh Patel, and Richard Halverson. 2018. Is More Active Always Better for Teaching Introductory Programming?. In *2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. 103–109. doi:10.1109/LaTICE.2018.00006 ISSN: 2475-1057.

[62] Ben Stephenson. 2019. Coding Demonstration Videos for CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 105–111. doi:10.1145/3287324.3287445

[63] Xiaotian Su and April Yi Wang. 2025. The Stress of Improvisation: Instructors' Perspectives on Live Coding in Programming Classes. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '25)*. Association for Computing Machinery, New York, NY, USA, Article 525, 6 pages. doi:10.1145/3706599.3719993

[64] John Sweller, Paul Ayres, and Slava Kalyuga. 2011. *Cognitive Load Theory*. Springer New York, New York, NY. doi:10.1007/978-1-4419-8126-4

[65] John Sweller, Paul A Kirschner, and Richard E Clark. 2007. Why minimally guided teaching techniques do not work: A reply to commentaries. *Educational psychologist* 42, 2 (2007), 115–121.

[66] April Yi Wang, Yan Chen, John Joon Young Chung, Christopher Brooks, and Steve Oney. 2021. PuzzleMe: Leveraging Peer Assessment for In-Class Programming Exercises. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (Oct. 2021), 415:1–415:24. doi:10.1145/3479559

[67] April Yi Wang, Andrew Head, Ashley Ge Zhang, Steve Oney, and Christopher Brooks. 2023. Colaroid: A Literate Programming Approach for Authoring Explorable Multi-Stage Tutorials. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–22. doi:10.1145/3544548.3581525

[68] Andrea Watkins, Craig S. Miller, and Amber Settle. 2024. Comparing the Experiences of Live Coding versus Static Code Examples for Students and Instructors. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 506–512. doi:10.1145/3649217.3653562

[69] Andrea Watkins, Amber Settle, Craig S. Miller, and Eric J. Schwabe. 2025. Live But Not Active: Minimal Effect with Passive Live Coding. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1190–1196. doi:10.1145/3641554.3701786

[70] Greg Wilson. 2016. Software Carpentry: lessons learned. *F1000Research* 3 (2016), 62. https://doi.org/10.12688/f1000research.3-62.v2

[71] Benjamin Xie, Jared Ordona Lim, Paul K.D. Pham, Min Li, and Amy J. Ko. 2023. Developing Novice Programmers' Self-Regulation Skills with Code Replays. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23, Vol. 1)*. Association for Computing Machinery, New York, NY, USA, 298–313. doi:10.1145/3568813.3600127

[72] Ashley Ge Zhang, Yan Chen, and Steve Oney. 2023. VizProg: Identifying Misunderstandings By Visualizing Students' Coding Progress. 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 596, 16 pages. doi:10.1145/3544548.3581516

[73] Daniel Zingaro. 2010. Experience report: Peer instruction in remedial computer science. In *EdMedia*. Association for the Advancement of Computing in Education (AACE), 5030–5035.