Intern. J. Comput.-Support. Collab. Learn
https://doi.org/10.1007/s11412-023-09392-2

Contesting sociocomputational norms: Computer programming instructors and students' stancetaking around refactoring

Morgan M. Fong[1] · **David DeLiema**[2] · **Virginia J. Flood**[3] · **Oia Walker-van Aalst**[4]

Received: 24 January 2022 / Accepted: 5 April 2023

Abstract
Working solutions to problems are not definitive end points. As a result, code that is technically correct can still be treated as needing revising – a practice in computer programming known as refactoring. We document how late elementary to middle school students and their undergraduate instructors weigh the possibility of refactoring working code in an informal summer computer science workshop. We examined a 20-min stretch of classroom activity in which multiple coding approaches were explicitly evaluated as alternative routes to the same code output. Our theoretical framework draws on the stance triangle, amplifying and attenuating inequity, and an extension of sociomathematical norms. Using the method of interaction analysis, we transcribed and analyzed stretches of talk, gesture, and action during whole class dicourse and small group interactions involving 4–6 students. We investigated how instructors and students introduced, characterized, applied, and contested socio*computational* norms through stancetaking in classroom discourse, which shaped whose voices contributed to the discussion and whose ideas were treated as impactful and praiseworthy in the classroom. Because it is within these discourse spaces that instructors and students interpret and reinterpret sociocomputational norms about what is valued in programming approaches, educational researchers and teachers might attend to these conversation dynamics as one route to fostering more supportive and inclusive learning spaces.

Keywords Computer science education · Interaction analysis · Inequity · Refactoring · Sociocomputational norms · Stance

Extended author information available on the last page of the article

Introduction

"I could rewrite it, but I decided to leave well enough alone."[1]
"Good, better, best. Never let it rest."[2]

What counts as a solution or an end point to a problem is not always straightforward. In learning settings, even when a student arrives at a viable approach to a problem, other students and instructors may make the case that more is needed. Given uncertain or even contrasting criteria around which solutions count as good enough, better, or best, the opposing quotes above point to intractable questions around whether to invest time, effort, and risk toward refactoring, the process of improving an already working approach (Alves et al., 2016; Papert, 1980; Reason, 1990). In this paper, we examine these contrasting pathways in a classroom of late elementary to middle school students learning computer programming. In particular, we document how students and instructors weigh the possibility of pursuing new approaches to working code. Joining growing efforts to examine the social context of the computer science and robotics classrooms (e.g., Elliott, 2020; Hennessy et al., 2023; Ryoo et al., 2020; Silvis et al., 2022; Vakil, 2020), we attend in particular to a facet of inequity in classrooms (Esmonde & Booker, 2016; Philip & Gupta, 2020), namely who has access to the conversational floor and whose ideas are treated as impactful and praiseworthy (e.g., Boaler, 2008; Shah & Lewis, 2019) in teacher–student discourse that, over time, assigns unequal value to decisions about whether and how to revise code.

We conduct this inquiry with explicit attention to persistent tensions between disciplinary norms in computer science (CS) and students' agency to carve out their own approach (Papert, 1980; Philip & Sengupta, 2021; Ryoo et al., 2020; Turkle & Papert, 1990). Computer science curriculum and notions of computational thinking have routinely valorized the process of generating abstractions in code that are more efficient or generalizable than prior versions (Brennan & Resnick, 2012; Wing, 2008). Despite this disciplinary commitment, few studies have empirically scrutinized how instructors and youth new to coding collaboratively navigate the prospect and process of refining code that already works. We draw on prior computer-supported collaborative learning (CSCL) and learning sciences scholarship that attends to equitable participation in group discourse (Danish et al., 2020; Philip & Gupta, 2020; Sinha et al., 2015), in particular to research on moment-to-moment shifts in status and inequity throughout peer dialogue in classrooms (Philip et al., 2018; Shah & Lewis, 2019; Simpson et al., 2017). Toward this end, we focus on the following research question: How do students and instructors in one classroom propose, debate, and ultimately decide during public discourse whether to refactor their code? In answering this question, we attend to the manner in which students come to problematize a pathway to a solution that is already working for some of them, how instructors and students shape the process of defining "inefficiency," and how participants in the classroom endorse and contest one another's stances.

[1] This phrase "Leave well enough alone" may have originated in Aesop's fable, "The Fox and the Hedgehog." The example we use here is borrowed from the Cambridge English Dictionary (https://dictionary.cambridge.org/us/dictionary/english/leave-well-enough-alone).

[2] This phrase has often been attributed to St. Jerome; however, this attribution has been contested.

Refactoring in computing and beyond

The process of revising working code, or refactoring, has played a central role in the discipline of computer programming. Fowler's (2019) textbook on refactoring, written for professional programmers, provides the following accessible definition:

> "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written." (p. 9)

Refactoring is characterized by a continuous revision process in service of reaching the same functional requirements or outcomes. Reasons to refactor include improving reusability, communicability, maintainability, adaptability, efficiency, elegance, and knowledge (Ionescu et al., 2020; Suryanarayana et al., 2014; Thompson, 2020). In the computer science discipline, refactoring is not a fringe activity; refactoring is an important strategy to keep software systems up to date and extensible (Demeyer et al., 2005). However, refactoring is inherently uncertain and intersubjective. Programmers, couched in a context with particular learning or design goals, and often in discussion with other programmers, have to decide whether their code is efficient, communicable, or adaptable enough. These considerations involve not just the code in its current form but also the systems that surround the code, judgments of how others might use and understand the code, and projections about how the software might need to change moving forward.

Deliberation about these dynamics is core to programming and could provide a point of departure for CSCL designs that focus on productive failure (Kapur, 2008; Kapur & Kinzer, 2009). For example, possibilities for refactoring are ill-structured and might naturally provoke discussion about a range of problem-solving strategies. This inherent uncertainty, and the range of strategies one could take to navigate these ill-structured spaces, might provide a powerful start to a productive failure learning design. Moreover, while the word "refactoring" arose in the context of computer programming, the open-ended practice of revising an already functional solution occurs across disciplines. For example, in refining a scientific model, a student or scientist may recognize that the model makes accurate predictions about the target phenomenon but is hard for others to interpret or use. That is, the model could become more generalizable or accessible over time (Schwarz et al., 2009). Enyedy (2005) captures this revision process in a study of how second and third graders use progressive symbolization to iterate and refine topological maps into more formalized and abstract forms.

Similarly, mathematicians aim not only for accuracy in mathematical definitions but also for minimality and elegance (Van Dormolen & Zaslavsky, 2003). Mathematics students might also be encouraged to adopt these practices (Vinner, 2002). For example, Kobiela and Lehrer (2015) document sixth-graders' efforts to define geometric shapes, attending in particular to how students negotiate the criteria for what constitutes a sufficient or acceptable definition. In this study by Kobiela and Lehrer, students argued about whether the word "side" in a definition of a triangle needed to be phrased "straight side" if the word had already been defined that way in their classroom. When negotiating definitions, students might consider if the definition should be efficient or if certain properties should be foregrounded or backgrounded (e.g., a square's diagonal versus its sides) (Kobiela & Lehrer, 2015, p. 427). In this way, learning to judge the soundness of

a mathematical definition, and whether it warrants revising, involves sociomathematical norms, or "normative understandings of what counts as mathematically different, mathematically sophisticated, mathematically efficient, and mathematically elegant in a classroom" (Yackel & Cobb, 1996, p. 461). Sociomathematical norms—or *sociocomputational* norms in the context of computing—play a role in guiding classroom expectations about quality. Both in the classroom and in professional industries, what counts as an "elegant" or "good" solution is inherently intersubjective; these norms are socially and culturally negotiated and evolve given changing technology and circumstances (Nader, 1996; Philip et al., 2018; Stevens & Hall, 1998).

In summary, refactoring has been identified as a recurring practice not only in computing, but also in other science, technology, engineering, and mathematics (STEM) domains. Refactoring as an activity is inherently uncertain, dependent at least in part on the domain-specific and context-specific norms that constitute elegance, efficiency, or other valued attributes.

The integral but potentially contentious place for refactoring in CS education

Computer science educational researchers have called for courses that explicitly address refactoring (e.g., Romeike & Göttel, 2012). Some have even argued that computer science curricula have overlooked refactoring at the undergraduate level (Stoecklin et al., 2007). In a study that proposes a software tool for refactoring, the authors note that there is little agreement on best practices for teaching refactoring (Demeyer et al., 2005). At the same time, the K-12 Computer Science Framework emphasizes the value of refactoring. In recognizing the goal of abstraction and the process of debugging, the framework also invites learning experiences focused on the gradual refinement of code, such as the "iterative process of designing, implementing, and reviewing programs" and "remixing other programs within a community" (2016, p. 104). Building on a few recent efforts to raise awareness to refactoring practices in CS education (Danielak, 2022; Techapalokul & Tilevich, 2019), we extend this work by looking at the under-examined complexities of classroom discourse that surround discussions about revising code.

In alignment with CSCL scholarship on the social construction, appropriation, and impact of classroom norms (Danish et al., 2020; Overdijk et al., 2014; Siyahhan et al., 2010), we advance a critical perspective on refactoring practices in computer science education that attends to the ways that classrooms determine what constitutes valuable code. The work of introducing disciplinary conventions, values, or norms of revision/refactoring often takes place in collaborative, extended discourse when students have been presented with the task of problematizing a pathway to a solution that might work but could be improved. While we know a great deal about structuring group problem-solving processes from the CSCL literature (e.g., Cress et al., 2021; Radkowitsch et al., 2020; Scheuer et al., 2010), we know little about how CS disciplinary norms are negotiated and contested, and even less about how those negotiations shape refactoring decisions for new coders. When considering whether to refactor, students take as a point of departure code that is already working to a certain extent. Because nothing in principle has broken down in their process of pursuing an approach to a coding activity, students may be reluctant to leave a known solution. Thus, a pedagogical challenge for supporting students' learning about refactoring is that moments of success (e.g., working code) might not spark reflection and

course correction. Instead, moments of friction, impasse, and breakdown prompt individuals across levels of experience to dedicate effort and time to reflect on and consider adapting their process in a variety of scenarios (Kapur & Kinzer, 2009; Koschmann et al., 1998; Weiner, 1985). In a similar vein, research on conceptual change points to a moment of dissatisfaction as a catalyst for thinking about a topic in a new way (Posner et al., 1982, p. 214). Broadly speaking, students may gravitate toward the tendency of cognitive conservatism (Reason, 1990), or a reluctance to approach an activity in a new way unless a problem arises. For example, Danielak (2022) documents how an undergraduate programmer's "design choices were made early and persisted" (p. 29) throughout a semester of coding—a kind of "design inertia" (p. 29)—which ultimately introduced considerable complexity into the student's code. Instructors might have particular programming solutions in mind that align with longer-term learning goals, but students may prefer in the shorter term to write their code a different way.

Indeed, one of the foundational texts on computer programming education drew direct attention to these tensions surrounding refactoring (Papert, 1980). In several anecdotes, Papert (1980) problematized when to ask elementary to middle school students to refactor, noting a balancing act between allowing students to use a known set of commands and nudging students to write their code in more efficient ways. The following extended excerpt describes this dynamic:

> "Deborah decided to restrict her Turtle commands, creating a microworld within the microworld of Turtle commands. She allowed herself only one turning command: RIGHT 30. To turn the Turtle through 90 degrees, she would repeat RIGHT 30 three times and would obtain the effect of LEFT 30 by repeating it eleven times. To an onlooker it might seem tedious to obtain simple effects in such complicated ways. But for Deborah it was exciting to be able to construct her own microworld and to discover how much she could do within its rigid constraints. She no longer asked permission to explore. And one day, when the teacher offered to show her a 'simpler way' to achieve an effect, she listened patiently and said, 'I don't think I'll do it that way.'"

Papert's anecdote highlights a student's authority to push back on instructor expectations and instead make a personal choice about when to refactor. As Papert alluded to with critiques of instructionism, this tension between students' preferences and instructors' expectations might be amplified by instruction that is guided by a scope and sequence that makes certain assumptions in advance about what will be coherent and valuable to students and when (Sikorski & Hammer, 2017).

These dynamics are especially concerning against the backdrop of recent scholarship on computing learning spaces that documents the resistance high school computing students face to asserting their agency during learning (Ryoo et al., 2020), including CSCL scholarship that documents girls' efforts to inequitably position one another along dimensions of power and status during a summer camp (Simpson et al., 2017). This contemporary work resonates with concerns raised over 30 years ago about how programming education environments limit epistemological pluralism, defined as "accepting the validity of multiple ways of knowing and thinking" (Turkle & Papert, 1990). Rightful presence (Calabrese Barton & Tan, 2019)—students' power and authority to shape their approach to learning and "resist an unquestioning acceptance of established norms" (p. 621)—is not guaranteed in computing learning spaces. Inequities are a prevalent part of learning, attenuated and amplified by the moment-to-moment contours of discourse in the computing classroom (Shah & Lewis, 2019). Inequitable dynamics around marginalization and domination in pair programming spaces designed to promote equity have been connected to students

working quickly to finish tasks (Lewis & Shah, 2015), which might be exacerbated by a focus on "efficient" refactoring. Moreover, efficiency in problem solving has been tied to exclusionary practices along racial and gendered lines in undergraduate mathematics courses, such as college instructors suggesting that students take a lower-level course or not enroll in the next course in the sequence when they do not complete problems rapidly, or even laughing at students who struggle with seemingly easy problems (Leyva et al., 2021a, b). Exclusionary practices like these create unnecessary pressure and have disproportionately impacted mathematics students from marginalized groups (Leyva et al., 2021a, b), heightening the urgency to examine these dynamics in CS education spaces.

Given the factors described above around instructional (Papert, 1980), emotional (Koschmann et al., 1998), and cognitive (Posner et al., 1982; Reason, 1990) reasons to question when students will be interested in revising a working solution, we might expect that refactoring considerations are especially contentious during collaborative learning, where dominant sociocomputational norms may attenuate students' agency. For example, Vakil (2020) used two case studies of high school students to document how students grapple with the tension between "values of a discipline" (p. 107) (i.e., sociocomputational norms) and their own potentially differing values while simultaneously maneuvering the social norms in their classrooms. This tension raises a central question for the discipline. As instructors bring students into a "community of learners," a stepping stone toward a community of practice (Suzuki & Kato, 1995), how can they help students to understand, explore, critique, and respond to the field's traditional sociocomputational norms without expecting uncritical acceptance of those norms? In all, we join recent calls in the learning sciences and in CSCL (e.g., Gomez et al., 2021; Sengupta et al., 2021; Simpson et al., 2017; Wang et al., 2021) to pay more analytic attention to the ways in which issues of equity arise when diverse ways of knowing and participating in computer science are contested and negotiated. We adopt a situated perspective to document how instructors and students in one stretch of classroom interaction navigate the disciplinary convention of refactoring code.

In summary, situations in which instructors and students weigh the possibility of refactoring may prove to be contested spaces. If this conjecture about CS classroom discourse is correct, we argue that it is essential to study through a critical lens the question of who contributes to these discussions and whose ideas and coding approaches are treated as impactful and praiseworthy.

Theoretical framework and central constructs

We take as a point of departure educational research frameworks that recognize that classroom spaces assign (unequal) value to particular student actions and outcomes over others (e.g., Holland et al., 2001). In particular, we are interested in understanding whose voices are present and whose ideas are treated as impactful and praiseworthy in moment-to-moment discourse about refactoring. Among the wide range of facets of inequities in education—including historical injustice, suppression of future opportunities, and political oppression (e.g., Bang & Vossoughi, 2016; Gutiérrez & Jurow, 2016; Philip & Azevedo, 2017)—we join others in recognizing that conversations in classrooms, both in terms of who participates and whose ideas are valued, are central components of (in)equity (Philip & Gupta, 2020). Through attention to micro moments of social interaction in the classroom, we explicitly take up the call from Langer-Osuna and McKinney de Royston (2017) to develop "conceptual and analytic tools that

examine how these processes and related patterns of inequality are instantiated, perpetuated, or transgressed through interaction at the classroom level" (p. 645–646; see also Philip & Gupta, 2020). Because we are focusing on classroom discourse, our theoretical framework draws on three constructs that have been widely used in education research studies of social interaction: (1) stancetaking (Du Bois, 2007), (2) sociomathematical norms (Yackel & Cobb, 1996), and (3) the amplification and attenuation of participatory and relational inequity (Shah & Lewis, 2019). Below, we articulate how these three features of our theoretical framework provide a unique and synergistic lens on studies of inequities in classroom discourse about refactoring.

Stancetaking: evaluate, position, (mis)align

Du Bois' stance triangle framework (2007) is a powerful tool for documenting how classroom participants voice their opinions on issues and (dis)agree with one another as a conversation progresses. Moreover, the theoretical lenses of stancetaking and positioning (see Davies & Harré, 1990) have been applied productively in CSCL and learning sciences scholarship (e.g., Watkins et al., 2016), including within studies that attend to equity and power. For example, stancetaking has been used to document college students' argumentation about ethics in engineering and whose voices are suppressed along the way (Philip et al., 2018), and to examine CSCL contexts in which particular positions and storylines in the classroom shape who benefits most from collaborative learning designs (Simpson et al., 2017). We carry these threads forward here by utilizing the stancetaking framework of Du Bois. Stance is formed through dialogue between people: "I *evaluate* something, and thereby *position* myself, and thereby *align* with you" (emphasis added, Du Bois, 2007, p. 163). Du Bois' statement makes explicit that stance is an action that is inherently value-laden and builds recursively on prior stances to form new stances. Stancetaking is not solely a linguistic practice. Body movement, outward signs of affect, and the incorporation of physical materials in the environment constitute stance (Goodwin, 2006, 2007).

By attending to multimodal and linguistic features of social interaction, we are specifically interested in conversation turns in which students and instructors adopt a stance on refactoring. We articulate the components of stance—evaluating, positioning, and aligning—in the following way: Students and instructors may evaluate refactoring (the object or the focus of a given strip of conversation) by drawing on particular criteria (e.g., noting that a particular coding approach is fast, arguing that the approach is too much work, having a personal preference for a particular approach). Through these acts of evaluation, students position themselves as someone supportive, opposing, or neutral/ambivalent about refactoring. Given that others in the classroom might be evaluating refactoring possibilities, and thus positioning themselves within a broader argumentation landscape, we are interested in tracking whether students align (what Du Bois refers to as convergence) or misalign (what Du Bois refers to as divergence) with prior stances on the topic (p. 164). Students mark this (mis)alignment specifically in their language (e.g., "yeah but…" or "No, you could…") and/or through multimodal cues (e.g., shaking one's head in disagreement). These three components form a stance. For example, a student might evaluate the notion of refactoring a hard coded sequence into a loop as a waste of time, which positions that student in opposition to refactoring, and which might misalign with a peer who had previously evaluated the loop as a time saver.

Sociocomputational norms

Given our attention to stancetaking during discussions of refactoring, we are interested in understanding the extent to which these stances relied on different criteria for what constitutes effective approaches to coding challenges, including different interpretations of those criteria. To capture these dynamics, we drew inspiration from Yackel and Cobb's (1996) notion of sociomathematical norms. Like social norms (e.g., a recurring expectation in a classroom that students explain their solutions), sociomathematical norms are formed over time and, like stancetaking, develop over the course of social interaction (Lopez & Allal, 2007). However, sociomathematical norms are distinct from social norms in that they are domain specific, such as mathematical notions of "efficiency" and "elegance." Prior work has described in rich detail how in one third grade science classroom, teacher-initiated but socially defined sociomathematical norms developed alongside students' computational thinking to shape what counted as a "good" measure of distance (Dickes et al., 2020). Separately, researchers have explicitly called for computing education environments to adopt "professional norms" around "precision" in programming (Kolikant & Pollack, 2004), a move that underscores the power of disciplinary norms to shape evaluations of the quality of student work. With a critical eye toward these disciplinary structures (drawing inspiration from Philip & Sengupta, 2021), in our paper, we extend and explore the construct of sociomathematical norms in the context of computer science by examining sociocomputational norms: normative understandings of what counts as computationally sophisticated, computationally efficient, and computationally elegant. In our analysis, we illustrate how sociocomputational norms are explicitly described by instructors and students in whole class discourse. In addition, we show how recurring stances taken throughout peer-to-peer and instructor–student conversations during coding communicate notions of efficiency, elegance, etc., especially in terms of who holds others accountable to their stance, and thus amount to gradually refined sociocomputational norms for the classroom. In this way, we are interested in attending to both explicitly professed sociocomputational norms (e.g., "good code is concise") and to enacted stances that endorse particular sociocomputational norms (e.g., "loops save you so much time").

Amplifying and attenuating participatory and relational inequity

Because stancetaking (and the sociocomputational norms that stances endorse, contest, and refine) arise within social interaction between instructors and students over time, these frameworks can play a role in helping researchers understand moment-to-moment inequities in discourse (Philip & Gupta, 2020). Our approach in this paper responds to broader calls in CSCL and the learning sciences to examine inequities in classroom discourse (Esmonde & Booker, 2016; The Politics of Learning Writing Collective, 2017) and to build on empirical findings in CS education research that document (resistance to) inequitable discourse dynamics (Ryoo et al., 2020; Shah et al., 2020; Tsan et al., 2021), such as middle school pair programmers' departures from their roles as "navigators" and "drivers" during coding (Denner et al., 2021). We sought to remain open to capturing the shifting dynamics within the classroom that might ultimately elevate some students' approaches and dismiss others. To attend to the possibility of these shifts over time, we drew specifically on Shah and Lewis's (2019) paper examining the moment-to-moment fluidity of inequity in peer-to-peer and instructor-student dialogue. Shah and Lewis (2019) document this process by attending to the ongoing amplification and

attenuation of inequity in pair programming. Their approach distinguishes participatory from relational equity. Participatory equity is the extent to which participants in dialogue both have opportunities to engage in communication and take advantage of those opportunities. This involves both access to the "conversational floor" and utilization of the conversational floor. On the other hand, relational equity (originally developed by Boaler, 2008) is the extent to which people respect each other's ideas, including ideas different from their own, evidenced by whose ideas impact/shape the classroom discussion and whose ideas are outwardly valued/praised in conversation. In the case of refactoring, relational equity becomes visible through whose ideas about refactoring (a) are considered in classroom discussion, (b) inform how students actually write their code, and (c) serve as benchmarks for endorsement or praise.

As Shah and Lewis point out, relational and participatory equity are intertwined: a classroom where peers respect each other will naturally lead to more opportunities for everyone to offer evaluations and positionings, including misaligning stances. On the other hand, a classroom where peers and instructors do not respect each other's contributions might lead to fewer opportunities for students to participate in the conversation. Shah and Lewis explicitly identify inequity as a constant feature of peer-to-peer and instructor–student relationships, and thus an important area of focus in computing learning environments. For this reason, Shah and Lewis use the terms amplifying and attenuating to describe inequity as a persistent tendency that is constantly being negotiated and either exacerbated (amplified) or mitigated (attenuated). In our work, we remain open to the possibility that stretches of interaction will blend facets of attenuated and amplified inequity.

Summary of theoretical framework

All together, these constructs allow us to investigate how sociocomputational norms around refactoring code into more efficient, elegant, and communicable abstractions take shape in the discourse of a computer science classroom. We investigate how instructors and students introduce, characterize, apply, and contest these sociocomputational norms through stancetaking in classroom discourse, with implications for participatory and relational inequities in the classroom. These constructs (stancetaking, sociocomputational norms, and participatory and relational inequity) amount to a theoretical framework that aligns with our commitment as interaction analysts (Jordan & Henderson, 1995) to study how meanings are negotiated on a public stage, without making assumptions about participants' inner experiences. We unpack this commitment and provide additional details on our methods below.

Methods

Statement of positionality

Our analysis presents a critical perspective on a learning community that members of our research team both participated in and helped design. The research team consisted of the authors of the paper: Morgan, David, Virginia, and Oia.

Prior to the workshop, David had partnered with the nonprofit community learning center to develop computing curriculum and programs for the center's students. David participated in the focal classroom as a researcher who took field notes at the back of the room and occasionally joined students at their tables to help with coding and/or debugging.

Indeed, David was part of the classroom discussion we analyze below. Oia and Virginia helped set up recording equipment in the focal classroom, but spent their time in different classrooms taking field notes and collecting video data.

Morgan joined the team after data collection as an undergraduate research assistant. Based on her courses as a computing major at the time, Morgan felt debugging was not discussed enough as a topic or skill, and participated in the project to better understand how students debugged and how instructors might be better equipped to teach debugging.

Through our experiences observing and participating in the nonprofit's classrooms, and our extensive conversations with instructors before and after the workshops, we recognized that this learning environment offered strong values about what it means to be a coder (e.g., learning to debug code independently). At the same time, we had been reading extensively about critical perspectives on computing education (much of which we cite in our literature review above). As we studied the video-recorded conversations in the classroom about debugging (our focal research topic), we noticed layers of marginalization that we decided to write about, especially because refactoring had not been "on the radar" in our interview questions or in instructors' reflections on debugging. In addition, David and Morgan had spent time in CS-specific education spaces (at the K-12 and college level) that had strongly signaled the value of "efficient" code. For David, who participated the most in co-designing pedagogical practices at the nonprofit, this translated into ignoring the possibility that refactoring to get more efficient code might disempower many students along the way. For Morgan, who was also involved in peer tutoring, this translated into pushing students toward a "correct" solution. In addition, as scholars who identify as White and Asian who have historically been given privileged positions in CS spaces, we further felt compelled to reflect critically on how inequities can arise in CS education spaces. Lastly, our identities as researchers of debugging in this research–practice partnership motivated us to bring to the attention of our research and practice team our observations about how refactoring was approached in the classroom and what alternatives we might consider.

All four authors participated in the data selection and analysis, which we describe next. We view our critical perspective on this classroom session as reflexively emerging from our own contributions to the curriculum and classroom discourse, and not simply as a detached analysis of teaching and learning outside of our influence. We provide this critical examination of inequity in the classroom with the goal of challenging ourselves, our collaborators (both instructors and designers), and the broader research community to pursue a deeper and more equitable understanding of refactoring, to better support students in computing learning environments.

Setting and participants

The specific setting for this research was a 2 week summer computer programming workshop (M–F, 9 am to 4 pm) held at a nonprofit community learning center. Students in the workshop either demonstrated financial need or attended schools with high proportions of students from low-income families (schools classified as Title 1 in the United States), and all students submitted an application to attend the workshop where they expressed their interest in programming (there was slightly higher demand than space available in the workshop). There were 14 students enrolled in the focal classroom. In response to an open-ended prompt on a demographic survey, six students identified as Latino/a, three students identified with multiple categories (e.g., Latino and Asian, Latina and African American), two students identified as Asian, one student identified as White, and two students did not provide any identification. Six of the students identified as girls, and eight identified as boys. Table 1 provides details on

Table 1 Pseudonyms, grade level, and coding experience of students in our focal classroom. Focal students have an asterisk (*) after their pseudonym

Pseudonym	Grade level	Coding experience at community learning center[a]	PixelBots experience
Nya*	Fifth	Two workshops	One workshop
Pat	Sixth	Three workshops	One workshop
Max*	Seventh	Five workshops	One workshop
Rio	Eighth	No workshops	No workshops
Ash*	Fifth	Two workshops	One workshop
Gil	Fifth	Two workshops	One workshop
Sal	Fifth	Two workshops	One workshop
Ana	Fifth	Two workshops	One workshop
Eva	Fifth	Two workshops	One workshop
Ora	Seventh	No workshops	No workshops
Lou*	Sixth	One workshop	No workshops
Ian	Fifth	Five workshops	One workshop
Kay*	Fifth	Two workshops	One workshop
Zoa	Sixth	One workshop	No workshops

[a]Of the students with prior coding experience at the community learning center, most had attended earlier eight session weekend workshops that year, and some would have attended two week workshops from the previous summer

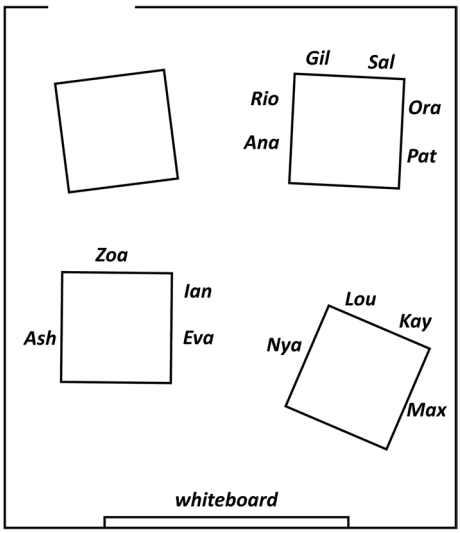


Fig. 1 Classroom seating and configuration. During the focal lesson, Ash, Zoa, Ian, and Eva sat together at the front left table. Nya, Lou, Kay, and Max sat together at the front right table. Ana, Rio, Gil, Sal, Ora, and Pat sat together at the back right table

students' pseudonyms, prior coding experience at the nonprofit community learning center, and grade level. The focal students, indicated by an "*" in Table 1, all had some experience at the community learning center and most had experience with the PixelBots platform (details below) more specifically. As a result, the focal students had different comfort levels with different programming constructs, which we detail in the next section. We attempted to

pick pseudonyms that matched students' self-identified gender. Figure 1 provides the seating arrangement of the students. All the participants in this study were introduced to the research ahead of the summer workshop, given time to reflect on whether they wanted to participate, and then added to a classroom in which research would take place only after a parent and child signed consent/assent forms, following our institutional review board (IRB)-approved procedure. Even with this process in place, we recognize that the context of maintaining consent during video-based research is ongoing and complex (Vossoughi & Escudé, 2016).

The lead instructors in the classroom, Ben (self-identified as a man) and Meg (self-identified as a woman) were both working toward undergraduate computer science degrees at local universities. Ben and Meg (both pseudonyms) taught at the nonprofit community learning center for two consecutive eight session weekend workshops prior to this summer workshop. In this particular classroom, Meg and Ben often taught from the front of the classroom with the aid of a projector (whiteboard in Fig. 1). David, one of the co-authors of this paper, participated in the classroom as a researcher/instructor, devoting most of his time to typing field notes at the back of the classroom (empty table, upper left in Fig. 1) but occasionally supported students in one-on-one or small group debugging sessions at their tables. The last group of participants relevant to our analysis, a team of three software developers, had been working on the PixelBots software platform for over a year prior to the summer session and occasionally visited classrooms at the non-profit community learning center.

Software and curriculum

The PixelBots platform was inspired by Logo, one of the first programming environments designed explicitly for children learning programming (Papert, 1980). In Logo, students write in a custom programming language to control the movement and drawing actions of a "turtle" around a grid. In PixelBots, students write in the Javascript programming language to control the movement and painting actions of an animal avatar around a grid. PixelBots was developed by a computing education nonprofit and is available at https://www.pixelbots.io/ to instructors and students interested in using the software (see DeLiema et al., 2020, 2022 for prior publications focused on PixelBots). Next, we describe some of the features available to students in the PixelBots platform.

Figure 2 is a screenshot of the PixelBots challenge students worked on in our focal lesson. An API in the center of the screen provided the syntax for the commands students could use when programming a specific PixelBot (e.g., "paint(color)") in the editing space to the right. This challenge in PixelBots asked students to recreate a pre-drawn image (i.e., four pink corners and an orange square) that students attempted to program. (Other challenges used a blank grid for free creation). Directly below the grid, students could adjust the opacity of the pre-drawn template image by dragging the slider from the white circle (i.e., completely transparent) to the black circle (i.e., completely opaque). Below the opacity slider, students could reset the state of the canvas by clicking the red counterclockwise arrow, run their code by clicking the green play button, see the end state of their code by clicking the yellow end button, and check their code output for correctness by clicking the blue "CHECK" button. Below these buttons, students could adjust the speed at which their code runs by dragging the slider from left (i.e., slowest) to right (i.e., fastest).

The design of the PixelBots platform created numerous opportunities for students to consider refactoring. First, because some challenges in PixelBots provided a target image to paint, students in the classroom often worked on their own laptops to paint the same

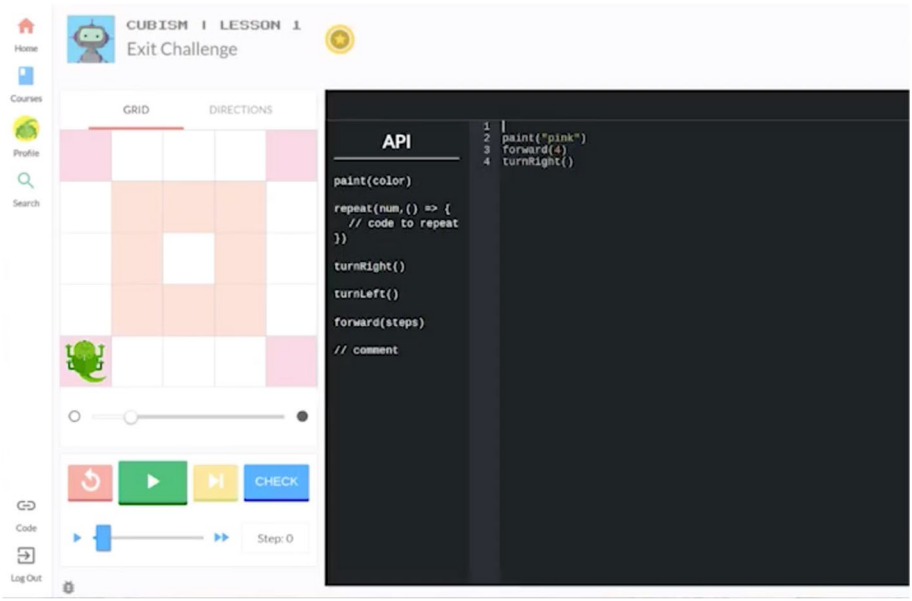


Fig. 2 Sample screenshot of PixelBots platform

image that other students were trying to paint (e.g., four pink corners in Fig. 2). For this reason, students often compared the different routes they were taking to paint the same image. Second, the API changed throughout the workshop, for example, by concealing certain functions or limiting the use of parameters. These evolving constraints invited students to solve similar challenges in new ways. Third, for some challenges, the PixelBots programming interface visibly tracked the number of lines in students' programs, which often drew students' attention, and invited them to compare the efficiency of their programs with respect to the number of lines written. In some cases in the workshop, the PixelBots challenge required students to complete it within a certain number of lines (though the students in the data analyzed below had not yet encountered this feature).

In the focal activity, students are tasked with painting the target image in Fig. 2. We outline some of the possible approaches to and refactoring opportunities in the focal activity below. We use the term "approach" rather than "solution" to highlight the fact that students may not have working code when discussing their programs with others.

One possible approach for programming the outer corners is to write each step:

1. paint('pink')
2. forward(4)
3. turnRight()
4. paint('pink')
5. forward(4)
6. turnRight()
7. paint('pink')
8. forward(4)
9. turnRight()
10. paint('pink')

We refer to this kind of approach as hard code. Students might notice that lines one to three are repeated twice. Students can highlight lines one to three, copy and paste them twice, then copy and paste line one to write line ten. We refer to this kind of approach as copy-and-paste. Students familiar with programming may start to notice the repeating pattern and refactor their code to use a loop:

```
1.  for(var i = 0; i < 4; i++) {
2.      paint('pink')
3.      forward(4)
4.      turnRight()
5.  }
```

We refer to the above approach as using a for-loop. Since "for" is a Javascript keyword, this syntax will work in the PixelBots platform. However, the software developers working on PixelBots introduced the repeat-loop syntax that became an important distinction in the activity. Students may achieve the same result as the for-loop by using the repeat-loop syntax:

```
1.  repeat(4, () => {
2.          paint('pink')
3.          forward(4)
4.          turnRight()
5.  })
```

Depending on how students approach the outer corners, their PixelBot ends up in different places. As students move to paint the inner orange square, they may hard code the commands, use copy-and-paste, or use a for or repeat-loop. Students might also notice some repetition within the repeat-loop itself, and use nested loops:

```
1.  repeat(4, () => {
2.        repeat(2, () => {
3.          paint('deepOrange')
4.          forward(1)
5.        })
6.        turningRight()
7.  })
```

The overarching scope of the curriculum contained a thread of refactoring practices woven throughout it, and thus provides important context for our analysis. At this stage of the research project, the nonprofit had piloted classroom activities with PixelBots, and this marked the second version of PixelBots curriculum focused on cubist art. The bulk of the curriculum was written by a curriculum developer at the nonprofit community learning center, but all of the instructors collaborated on the design of the debugging-focused activities, which included journaling to set debugging goals and strategies, instructors modeling and then prompting for the use of particular debugging strategies during coding, and a five-part debugging process (see Dahn et al., 2020; Dahn & DeLiema, 2020; DeLiema et al., 2020, 2022 for additional details). Recognizing that debugging had been a consistent source of discussion among the educators and learning designers at the nonprofit,

and seeing connections to the research literature on productive failure (Kapur & Kinzer, 2009), the nonprofit leaders and collaborating researchers had worked together to decide on debugging as a focal practice for their design-based research and received a grant from the National Science Foundation to pursue the work. At this point in the team's work, each of the above designs had been piloted in the previous two weekend workshops (eight sessions each) at the nonprofit. The instructors in the focal classroom, Meg and Ben, spent time familiarizing themselves with the curriculum in the week leading up to the workshop and before each class. In addition, the coding challenges students navigated each day were designed to work in synchrony with the lesson plans in the curriculum. In this way, the written lesson plans and accompanying coding challenges structured the work of the students and instructors over the 2 weeks. These skill-building challenges were intended to provide students with an array of coding techniques they could then deploy on custom projects (described below). In this way, the curriculum designers intended for students to master foundational programming concepts that would have high utility on project-based coding activities.

For their final projects, students created animal cubism paintings that took advantage of square and rectangle functions to paint creatures composed of block-like shapes. As such, the final project highlighted the process of decomposing an animal drawing into basic shapes so that general, shape-producing functions could be called to paint the animal. For this reason, many of the activities leading up to the animal cubism painting invited students to write code in a first-pass solution before rewriting the code into "more efficient" (as quoted in the curricular materials) or abstract shape-producing functions. Here is just a sample of some of the language written into the lesson plans with phrases italicized for emphasis:

- "Highlight the theme of *'Less is more'*" (day 02)
- "Hint 2: What chunk of code/lines are *reused*? How could you *rewrite* this program?" (day 02)
- "Can we use functions to *refactor this code*?" (day 02)
- "Students learn to define functions with parameters to *generalize bot's actions with arguments*" (day 03)
- "You will notice all three challenges paint the *same pixel*, in the upper right corner. I want you to focus on *refactoring the program to use the least amount of lines as possible*" (day 03)
- "Students will prototype a picture of an animal on PixelBots *using rectangles and triangles*" (day 04)

In short, refactoring toward particular valued ends was one of the core sociocomputational norms that structured the lesson plans and coding challenges in this workshop. The process of learning about loops, parameters, functions, and eventually multiparameter functions was motivated in the lessons by sociocomputational norms such as reuse, generalization, and using the fewest lines possible. Challenges that students worked on throughout the course constrained the number of lines they could use, demanding a process of refactoring a first solution into something more efficient with respect to the length of the program. However, the written lesson plans presented the value of these sociocomputational norms around refactoring in a neutral way—seemingly under the assumption that students would readily pursue refactoring—and discussions about pedagogical practices ahead of and throughout the workshops did not focus on refactoring.

Data sources

Students typically sat in groups of four to six at three tables spread throughout the room (see Fig. 1). We used GoPro cameras to capture social interaction that took place during coding. Our camera configuration involved two GoPro cameras raised on small stands at two corners of each table (six cameras in total), and one or two cameras in the upper corners of the room. To capture the details of students' code, we used Screencastify software run on Google Chrome laptops to record students' screens and capture audio near the laptop. Combining these data sources provides a multiperspective view of whole class discourse and the substance of conversations between students about their code. However, due to Screencastify needing an actively running laptop to record, some recordings were cut short or missing due to students closing their laptops during the workshop.

Selection of data

The current study of refactoring emerged from a broader design-based research investigation of how students learn to program and debug in an informal computer science workshop. Following the data collection activities for the workshop noted above, an author of this paper, Oia, was performing a substantive review of the corpus (Erickson, 1992) to build a collection of clips of interactions where instructors and students debug together when she noticed the focal lesson where refactoring was negotiated. In this lesson, students engage in discussion about how to balance the pursuit of a new coding approach against the cost of investing time to learn the new technique and the risk of introducing new bugs along the way. We purposefully selected this focal lesson for closer analysis because it presented an opportunity to closely examine and discover how this process of social negotiation can unfold (Erickson, 1992; Maxwell, 2013).

After reviewing the lesson for discussions about refactoring, we chose to focus our analysis on a 20 minute episode of classroom activity for multiple reasons. First, students and instructors in this discussion brought up an especially diverse range of issues related to refactoring, such as speed, efficiency, and elegance. Thus, the episode presented an opportunity to examine how students and instructors negotiated many different sociocomputational norms around refactoring. Second, this lesson featured a very public, intense, and persistent discussion on the merits of various approaches (copy-and-paste versus loops versus nested loops) with a range of different stakeholders participating (students, professional programmers, instructors, researcher–instructors). Third, because the topic of refactoring was taken up by different stakeholders in the classroom and handled in whole-class discourse at several moments, this discussion provided an especially rich opportunity to trace how distinct stances taken on refactoring impacted one another and changed over time. Fourth, this activity took place on the second day of the workshop and served as an early example of the instructors explicitly placing value on the pathways students took to the solution. Given our interest in understanding how inequity can arise in computer science learning environments, this provided us with an opportunity to look closely at how preferences for particular coding approaches took root in the classroom.

Data analysis

Our approach to analyzing the focal lesson drew primarily on the tradition of interaction analysis (Derry et al., 2010; Jordan & Henderson, 1995; Stahl, 2006). After choosing the

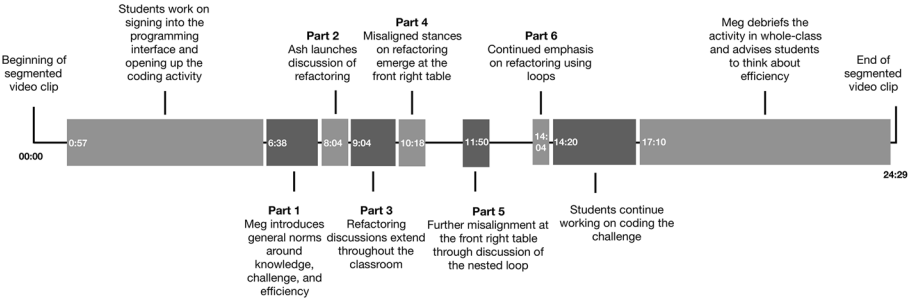


Fig. 3 Timeline of classroom events

20 minute stretch of classroom activity to analyze, we used Adobe Premiere to synchronize the audio and video from multiple camera angles and then visually arrange the clips to display them at once on the screen. We created one overview video (bringing together the cameras from the corners of the room) and one video for each student in the classroom (using a camera at their table and their laptop screen recording). This allowed us to document the details of conversation at each table in addition to whole-class discourse. We then orthographically transcribed (i.e., not including gestures, prosody, etc.) all the whole-class talk during the 20 min stretch and all the talk at each of the three tables. For several months, our research team met to watch video data and describe our observations relative to refactoring.

From our discussions, the unit of analysis we chose are stretches of interaction where refactoring arose and how the value of refactoring was publicly negotiated (see the timeline in Fig. 3 for a summary of these events; the bolded events roughly correspond to the phrasing of headings in the findings' subsections). The first was how Meg, an instructor, framed the challenge students worked on and how students ended up evaluating refactoring. Meg's framing introduced students to general classroom norms that were gradually instantiated as sociocomputational norms through students' and instructors' evolving interpretations. We selected the focal students to follow for the rest of the key moments based on their strong opinions on their respective approaches, sustained discussion of the sociocomputational norms, and different status levels within the classroom. Additionally, we chose students based on the availability of screen recording data to see and track their coding progress.

For these key moments, we followed the interaction analysis tradition of creating transcripts based on representational conventions from conversation analysis and representational conventions from scholars who attend to multimodality. In particular, we used a transcription style inspired by Jefferson (2004) and Goodwin (2018) for our interaction analysis because it allows us to capture the methods and resources participants used to make their ideas and stances publicly visible to one another. This includes the use of prosody, gaze, head nods and posture, facial expressions, talk, gestures, and other resources. Images included in transcripts were modified to enhance the visual anonymity of participants. However, to make it easier to distinguish different participants, adults' hair is colored black, whereas students' hair is white. A transcription key is provided in Appendix A. Our subsequent analysis of these transcripts strove to not make inferences about participants' private intentions, thoughts, or feelings. Rather, we took an emic approach and anchored our claims in what the participants made publicly relevant over the course of the interaction (Jordan & Henderson, 1995). With these commitments in mind, and through our use of three focal constructs (sociocomputational norms, stancetaking, and participatory and relational inequity), our examination of classroom inequity does not account for how students felt about or reflected privately on moments of participatory

and relational inequity in classroom discourse, but rather documents what is observable in the video data regarding who participates in refactoring discussions and whose ideas are treated outwardly as impactful and praiseworthy.

Our detailed analyses of the processes involved in how this programming classroom approaches refactoring allow us to expand current theory on how difficult and value-laden disciplinary decisions are negotiated in educational settings (a form of analytic generalization, see Yin, 2009). Our three-part framework, integrating stancetaking, sociocomputational norms, and relational and participatory inequity, provide a generative lens on research questions and analyses both in other computer science learning settings, as well as outside of refactoring practices in computer science education. When norms are an explicit part of educational conversations, studies in other settings can examine the microlongitudinal variations in community members' stances on those norms. Because these analyses would track (mis) alignments in stances over time, they naturally pair with inquiries into whose voices join the discussion (attenuated participatory inequity) and whose voices have traction in the discussion (attenuated relational inequity). In this way, we hope that this three-part framework can aid other researchers in understanding the moment-to-moment process through which particular voices shape the sociodisciplinary norms of a community.

Findings

We present below a chronological analysis of the way in which the classroom comes to consider the possibility of refactoring and how participants negotiate the value of different coding approaches. We specifically document how the students and instructors surface, defend, and contest their stances on refactoring. In each subsection, we provide a summary of the actions taken by different instructors and students that lead up to and provide crucial context for a particular key moment. For key moments where refactoring was publicly negotiated, we then provide multimodal transcripts, followed by moment-by-moment analysis with respect to stancetaking. After the analysis of the transcripts, we revisit and summarize the actions observed in the transcript with respect to sociocomputational norms and participatory/relational inequity.

This focal classroom session occurred on the second day of the summer coding workshop. On the first day, the students had familiarized themselves with PixelBots by working with a partner and then independently to code basic PixelBot movements and painting actions. Because loops become a prominent part of the discussion on the second day, we note here that the instructors had briefly introduced the syntax and meaning of loops on the first day of class, but most of the students had not practiced using them, specifically the PixelBots repeat-loop syntax. Now on the second day, the PixelBots session is just getting started.

Part 1: Introducing general norms around knowledge, challenge, and efficiency

One of the instructors, Meg, stands at the front of the room and introduces the activity to the whole class. We see the instructor introducing three general norms: (1) be resourceful with what you know; (2) embrace difficulty/struggle, do not be afraid of it; and (3) work quickly and efficiently. Meg first introduces the norm of being resourceful with what you know by describing the purpose of the activity: "This is to show me and show yourself how much you learned about PixelBots." And she continues to emphasize: "Use what you

know." Meg frames what the students already know as a central resource. Second, Meg introduces the norm of embracing difficulty/struggle using a call and response chant. Meg initiates with, "Coders, I have a challenge for you!" and then students respond with: "Bring it! Bring it!" The chant frames the upcoming activity not only as difficult but also as a difficulty that the students should embrace.

Meg then introduces the norm of working quickly and efficiently. As the students are getting started with the challenge, Meg introduces a time constraint: "I'll give you guys 10 minutes to work on this." Several students respond, some indicating the time is too short, some indicating it is too long, and Meg encourages them to keep going. This negotiation over the duration of the upcoming work, while outwardly playful (there is smiling and laughing), is left unresolved (a timer is set with an unsaid number of minutes). This facet of deciding how long the challenge should take sets in motion a number of subsequent considerations around time-efficient solutions to the coding challenge.

At this stage, the details around what makes these norms domain-specific are vague. Additionally, this negotiation reflects a stretch of attenuated participatory inequity between students and instructors because the students have access to the conversational floor, chiming in and even pushing back on Meg's statements in a playful discussion. In all, Meg and her students' framing of the activity introduces nascent features of three norms that the students and instructors later specify and incorporate as sociocomputational norms in their stances on refactoring. At this stage of our analysis, we had also noticed that these norms may not have aligned congruently: some amount of tension is perhaps likely when students are using what they know, which might be more time consuming, or fearlessly trying a novel approach, which might save time.

Part 2: Launching refactoring—hackers and the loop

When faced with lots of repeating code, programmers have an opportunity to use loops (see Software and curriculum for sample solutions). Alternatively, students might find that the copy-and-paste function makes writing redundant code less of a hassle, reducing the need to use a loop in the first place. Ash, in the transcript below, kicks off the classroom's discussion about refactoring by making this very argument. Just as the class starts to work on the challenge, Ash, sitting at the front left table, gasps, leans back in his seat, raises his arms in a display of victory, and loudly declares to the class that he has discovered a way to cheat (see Fig. 4).

Ash makes his discovery known to the classroom with a pronouncement: "I fihhgured ouht a way to che::at I'm- I'm a hacKER (.) I'm a hacker now" (Fig. 4.1, 4). In this strip of talk, Ash evaluates his discovery as a "way to cheat" and explicitly calls himself a "hacker," a category of programmers that in popular culture connotes clever and rebellious approaches to coding [see, Ames, (2018), for a critical perspective on this history]. Ash outwardly demonstrates amazement by gasping audibly and speaking as though out of breath, and his exclamation is accompanied by a physical display of celebration: Ash throws his hands up and bobs his head back and forth, similarly to a runner finishing a race (Fig. 4a). Additionally, Ash looks up from his laptop and speaks loudly, publicizing his achievement. Layered together, these features present the discovery as a valuable technique worthy of sharing with others, including instructors in the classroom who might ratify the technique as amazing and hackeresque. In all, Ash evaluates the "cheat" and the title of "hacker" as deserving of celebration, publicity, and status through a number of cues.

01 Ash: [HHh *((gasp))* I fihhgured ouht a way to che::at
02 [
03 *[((Software developers and Meg look at Ash))*
04 Ash: [I'm- I'm a hacKER (.) I'm a hacker now
05 *[((Developers smile while alternating looks at each other and Ash))*
06 *((Meg and the developers approach the table))*
07 Dex: I gotta see what this (could/would) be
08 Joe: Hhh *((laugh))*
09 Meg: [What's- what's your hack
10 *[((Meg leans down to look at Ash's screen))*
11 *((Ash highlights a line of code))*
12 *((Meg flips her hand open))*
13 Meg: [#What's your hack?#
14 [
15 *[((Ash clicks command + C))*
16 Dex: Ah:h (he found) copy and paste ( ) a hack
17 Meg: #Oh:h [okay#
18 *[((Ash pastes the line of code))*
19 *((Ash raises his arms and bobs his head again and the developers laugh))*
20 Ash: [Copy and paste is the WORLD best
21 *[((Ash deletes the line he pasted))*
22 Zoa: You (told) us nothing new
23 *((The developers start to wander away))*
24 Ash: Yeah [YOU (COULD/CAN) ALL DO IT NOW (.)
25 COPY AND PASTE (.) THE FASTEST WAY TO FINISH
26 *[((Ash raises his arms again))*
27 Meg: Hey if you guys are copying and pasting a lot
28 think about [maybe using: ↓a loop (1.3)
29 [
30 Meg: if you're repeating a lot of the same [code]
31 Ash: [>Nayah<]
32 Ash: [I like copy and pas::te so (.) SO much better
33 *[((Ash leans back, shakes his head, and lifts his hands off the computer))*
34 *((Eva looks at Meg and laughs))*
35 Zoa: Less typing [(mmm/y'know/though)
36 Meg: [But here's
37 Meg: [the API for a loop (.) so (.)
38 *[((Meg points at repeat loop in API while Ash runs his code))*
39 Ash: Yep (.) I (d') want it (.)
40 Meg: Mhmm (.) [so keep doing it how] you're doing
41 *[((Meg points to Ash's screen))]*
42 it and maybe you'll notice [a pattern] later
43 Ash: [Right]
44 (1.4)
45 Ash: [tu::rn (1.2) (do- do::o) the copy and paste better (.)
46 Much faster
47 *[((Meg walks away from Ash to look at Eva's screen))*

Ash, left-most, leans back, looks up, raises his arms, and bobs his head side to side as Meg and the developers (black hair) walk around the room.

All three developers and Meg huddle around Ash.

Meg makes a looping gesture.

Fig. 4 Transcript detailing Ash's discovery of copy-and-paste

Ash's pronouncement attracts the attention of the adults in the room. The software developers, who had been wandering around the room, smile (Fig. 4.5) and with Meg, they walk toward Ash (Fig. 4.6). They "gotta see" (Fig. 4.7) for themselves. By physically surrounding Ash, Meg and the software developers attenuate participatory inequity by giving Ash the conversation floor. Meg asks with a smile: "What's your hack" (Fig. 4.13). When Ash demonstrates the act of copying and pasting a line of code (Fig. 4.11, 15), Dex, a developer, describes the copy-and-paste action as something that Ash found a hack (Fig. 4.16), to which Meg responds, "#Oh:h okay#" (Fig. 3.17), while nodding and smiling. In contrast to how instructors respond to other students' coding approaches later in this activity, here the instructor's acknowledgement (e.g., "Oh:h okay") does not outwardly valorize Ash's proposal, misaligning with Ash's strong positive evaluation.

Following this moment, Ash elaborates on and intensifies his stance. He once again raises his arms and bobs his head (Fig. 4.19). He announces to the whole classroom that copy-and-paste is the "WORLD best" (Fig. 4.20) and "THE FASTEST WAY TO FINISH" (Fig. 4.25). By evaluating copy-and-paste as the world's best, Ash strengthens his stance that using copy-and-paste is better than using any other approach to the challenge.

And by evaluating copy-and-paste as the fastest way to finish, Ash highlights its efficiency with respect to time. Moreover, in voicing this stance to the full classroom, Ash broadens its potential impact in the classroom. In noting in his evaluation the speed of completing the challenge, Ash starts to provide specific contours to the sociocomputational norm of efficiency.

During this stretch of time, Ash receives additional pushback on his position. In between his evaluations of using copy-and-paste, Zoa, a student sitting next to Ash, states, "You (told) us nothing new" (Fig. 4.22). In this moment of misalignment, Zoa evaluates copy-and-paste as an already known and thus unremarkable and low-impact proposal. After Ash states his case again in an even louder voice (Fig. 4.24–25), Meg, still facing Ash's table, suggests that if students "are copying and pasting a lot" (Fig. 4.27), then they should "think about maybe using: ↓a loop" (Fig. 4.28), accompanying the statement with a looping gesture (Fig. 4.c). The stance evaluates using "a lot of the same code" (Fig. 4.30) as a reason to change course and think about using a loop. By recommending a different course of action, "think about maybe using: ↓a loop" (Fig. 4.28), this talk evaluates copy-and-paste as an approach that could (and should) be promptly reconsidered and improved on. This position presents a new angle on the sociocomputational norm of efficiency: sequential commands repeatedly used might be replaced with a single loop.

Ash again misaligns with Meg's stance, offering a "> Nayah <" (Fig. 4.31) and highlighting personal preference, "I like copy and pas::te," before elaborating with a broader rationale that copy-and-paste is "so (.) SO much better" (Fig. 4.32). This talk is accompanied by a strong misaligning and embodied stance: Ash leans back, shakes his head, and lifts his hands off his computer (Fig. 4.33). In these initial stances, Ash is consistently positioning himself as someone in support of copy-and-paste and opposed to the loop and refactoring. Zoa enters the dialogue again briefly with a statement about "less typing," though it is not clear whether Zoa is referring to the loop or copy-and-paste. Meg again misaligns with Ash's stance (signaled with "but") while showing him where he can find the loop syntax in the API (Fig. 4.36–38). After Ash again affirms that he does not want to use the loop (Fig. 4.39), Meg temporarily aligns with his approach ("Mhmm (.) so keep doing it how you're doing it") before again signaling misalignment by leaving open the possibility that Ash will see a pattern later (Fig. 4.40–42). As Meg starts to walk away, Ash again evaluates copy-and-paste as "better" (Fig. 4.45) and "much faster" (Fig. 4.46).

This stretch of interaction makes evident how contrasting stances on refactoring arise. In the most direct sense, Ash's favorable evaluation of copy-and-paste as hackeresque, time efficient, and the overall best technique differs from Meg and Zoa's evaluation that this approach is well known and immediately worthy of students considering replacing it with a different approach. The participants in this exchange anchor their stances around distinct sociocomputational norms of efficiency. Ash's stance, which he defends in four separate turns, frames as valuable the use of a known approach to complete a task quickly, whereas Meg's stance frames repeated uses of a line of code as inefficient and a clear point of departure for exploring a new technique. Along the way, the participants directly mark in language and body movement the misalignment in their stances. In all, this stretch of dialogue reflects attenuated participatory inequity and some degree of attenuated relational inequity between student and instructor: Ash, the student, maintains access to the conversational floor (both interpersonally with Meg and with respect to the whole class), and Meg, the instructor, both responds directly to Ash's proposal and grants Ash leeway to continue using the copy-and-paste approach while keeping an open mind about using the loop. On the other hand, there is a degree of amplified relational inequity to this stretch of interaction. Meg swiftly evaluates the loop as the preferred, instructor-endorsed

approach to the challenge, in line with the longer-term learning objectives centered in their curricular materials. In this way, Ash's support of using copy-and-paste is neither praised for its ingenuity nor treated as impactful enough for instructors to suggest that students code using copy-and-paste. That is, after Ash shares the coding approach widely with the class, Ash is immediately asked to consider replacing his approach to the coding activity. This closes down an opportunity to explore in discourse how each approach appeals to different sociocomputational norms, both in terms of domain-specific interpretations of a particular norm (e.g., taking the "fastest" route or avoiding using "a lot" of the same code) and in terms of the relative importance of each norm. The misalignment that emerges here foreshadows how the instructors, and students at other tables in the room, continue to consider whether or not to refactor their code into a loop.

Part 3: Refactoring discussions extend throughout the classroom

Ash's comment sparked reflection on refactoring across the classroom that provides crucial context for the remainder of the classroom session. We highlight in brief the emergence of these discussions about refactoring without examining them in fine-grained interaction analysis detail.

At the front right table, Meg notices one student coding with a repeat-loop (Max) and another student hard coding a sequence of commands (Kay). Meg remarks to the whole class, "Interesting, I see some people using loops, and some people are not," before gesturing with open palms rising and falling like a weighted scale moving up and down. Meg's statement carries forward the discussion that Ash initiated; Meg is inviting students across the classroom to identify themselves relative to different approaches to the challenge. Furthermore, Meg divides all possible approaches into two buckets: using loops and not using loops. In using the phrase "some people are not," Meg does not provide a name for these other approaches, and instead describes them relative to the absence of the loop. In terms of impact on students' approaches to the coding challenge, loops are now voiced in whole-class discourse as a named approach, while alternative coding approaches (e.g., copy-and-paste) are reified as "not" loops. At this same table, Kay, who has been hard coding a sequence of commands, hears Meg and re-evaluates her code. She notices a pattern: "Wait! I just realized something. It's paint, turn right, forward, paint, turn left, forward–I don't care" (see Fig. 5.a for a screenshot of the code Kay is reading). Kay, who had experience coding with for-loops in previous workshops, rhythmically describes a paint-turn-move pattern, suggesting that she may see an opportunity to use a for-loop to refactor the code. However, Kay sharply cuts herself off, positioning herself as someone who "doesn't care" about this prospect.

Nearly simultaneously, the other instructor, Ben, engages in conversation at the back right table with a student, Sal, about why using the repeat-loop is better than using copy-and-paste. With a smile, Ben describes how Sal is using the copy-and-paste approach, and asks, "Is that the fastest strategy, Sal?" Sal says, "Yeah," and Ben mirrors Meg's earlier comment to Ash: "If you're doing the same thing over and over again, is there a better way to do it?" When Sal responds, "Yeah it makes things faster so you won't have to type it," Ben again pushes back: "Yeah, but you don't even have to type it. You don't even have to copy-and-paste it." Meg also walks to the back right table and begins a discussion with two students, Rio and Ana, about noticing patterns and using loops, while Ben begins to

instruct Sal on how to use the loop. In this stretch of interaction at the back table, Ben and Meg evaluate using the repeat-loop as the better way.

The short exchanges in Part 3 document how the topic of refactoring spread throughout the classroom. First, we see a continued refinement of possible meanings behind the sociocomputational norm of efficiency. Both Sal and Ben evaluate efficiency in terms of time and effort required to complete the challenge (less typing versus no typing at all). Meanwhile, Meg extends her stance that inefficiency is evident when recurring events create a pattern that could be coded with a shorter repeat-loop instead. Second, there is continued misalignment between instructors and students in their stances on refactoring. Multiple students (Kay and Sal) publicly voice stances in support of continuing to code the challenge in the way they had started. On the other hand, both instructors (Meg and Ben) make explicit requests for students to revise their coding approaches and to evaluate using the loop as either requiring less typing or requiring fewer lines of code. Despite appealing to different sociocomputational norms of efficiency, both instructors align in their positioning of themselves in support of repeat-loops as a valued refactoring move. Third, this stretch of classroom time reflects attenuated participatory inequity. Students and instructors are engaged in back-and-forth conversation about these coding approaches. Similarly, by building classroom discussion around students' approaches to the coding challenge, the instructors enact a degree of attenuated relational inequity, demonstrating that students' coding approaches have an impact on what the classroom centers in their reflections on code. On the other hand, this discussion reflects a degree of amplified relational inequity in terms of impact and praiseworthiness. In terms of impact, students are precipitously asked to yield their coding approach to the instructors' preferred approach. In terms of praiseworthiness, the coding approaches students endorse are not valorized and unpacked, but rather, described either in terms of the absence of a loop or in terms of inefficiency (e.g., "you don't even have to copy-and-paste it"). While this discourse nudges students toward the long-term learning objectives written into the curriculum, the binary sorting of "loops" versus "not loops" reduces the possibility to explore the variety of ways "not loops" could speak to alternative sociocomputational norms, including simply working as solutions that get the job done or getting comfortable with a new technique.

Part 4: Misaligned stances on refactoring with respect to preference and efficiency

At the front right table, Nya and Max, who are both tinkering with the PixelBots repeat-loop in their code, have not outwardly commented on the topic. The discussion at the table picks up when Kay opens with a negative evaluation of the loop.

Kay spontaneously says, "I don't ↑feel like using a for-loop it's tOO much ↓WO::RK" (Fig. 5.2–3) positioning herself against using the for-loop by evaluating it as something she doesn't "feel like" doing because "it's too much work," intensifying her point with a head shake (Fig. 5b). Because Kay had learned how to code loops using the for-loop syntax in a previous PixelBots workshop, she is using the phrase "for-loop" instead of "repeat-loop." In response, Nya provides subtle misalignment, dismissing Kay's point with a "TSs" remark and a smile (Fig. 5.5–6). Lou then expands on Kay's evaluation, picking up on the theme of "work," with the statement, "LI:fe is too much work" (Fig. 5.9). Lou's parallel sentence structure and generalization do not outwardly align or misalign with Kay's stance. In contrast, Max, overlapping with Lou, evaluates his use of the repeat-loop: "It's ↑actually a lot ↓less work" (Fig. 5.10). In saying "actually," Max marks direct misalignment with Kay. All four students at the table have now entered the

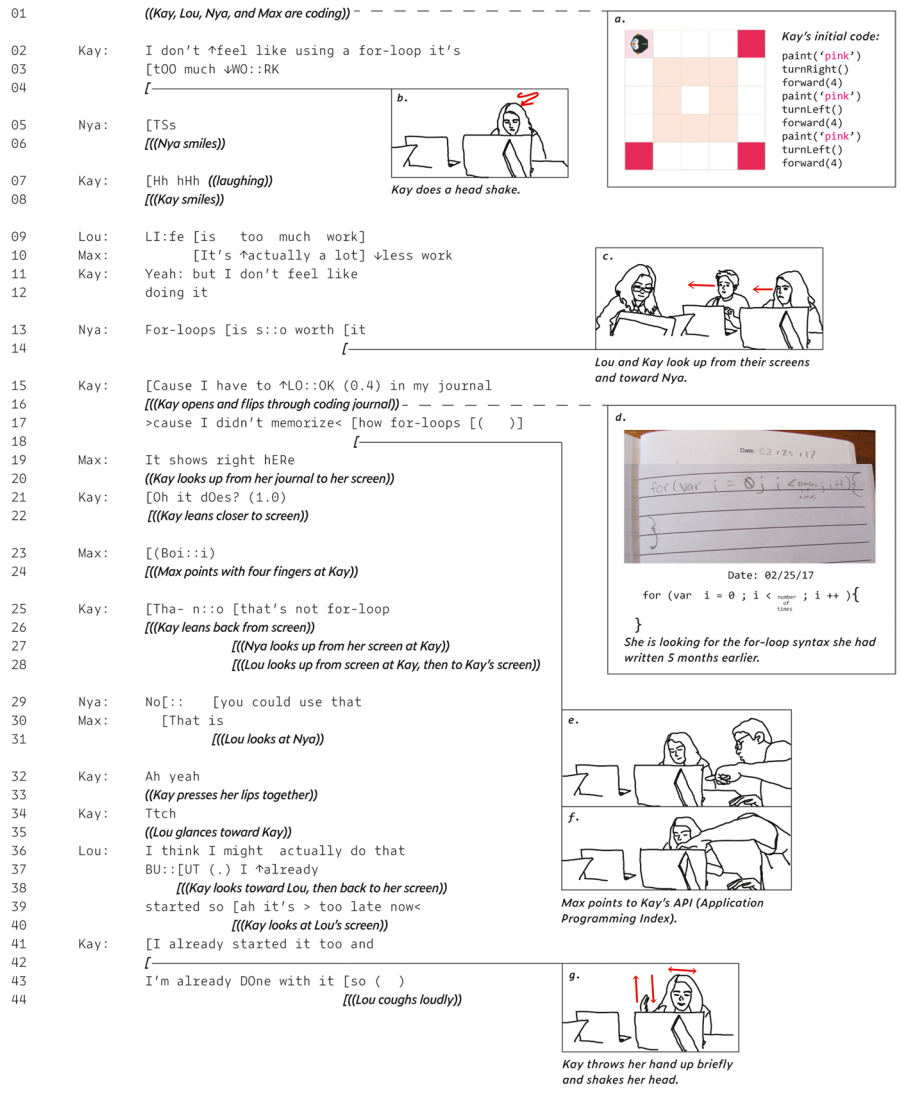


Fig. 5 Transcript detailing the conversation at the front right table regarding the loop

discussion: Nya and Max mark direct misalignment with Kay, while Lou expands on the topic at hand in a general way. What complicates this dialogue is that Kay and Lou are presently using the sequential coding approach that is known to them, while Nya and Max are presently using the repeat-loop approach that is known to them. The appeal to efficiency in terms of less work, at least in the short term, is relative to skill: what might be less work for Nya and Max is not necessarily less work for Kay and Lou.

In the coming turns, Kay maintains her stance. Though she temporarily signals alignment with Max ("Yeah:"), she repeats her evaluation, "but I don't feel like doing it" (Fig. 5.11–12), evaluating refactoring not relative to efficiency but now to personal

preference. Nya immediately responds, "For-loops is s::o worth it" (Fig. 5.13); Lou and Kay both look up from their work and toward Nya (Fig. 5c). Nya's positive, broad evaluation of using the for-loop ("so worth it"), not the repeat-loop, creates direct misalignment with Kay.

Kay returns to the sociocomputional norm of efficiency to defend her stance. Kay describes having to "↑LO::OK (0.4) in my journal" (Fig. 5.15) to find the for-loop syntax she wrote five months ago (Fig. 5d), and demonstrably flips one journal page at a time (Fig. 5.16). The sociocomputational norm of efficiency to which Kay is appealing is the amount of effort required to find, perhaps relearn, and then enact the for-loop approach, not efficiency in terms of the number of commands needed to run a program or the computational cost of running the program. Picking up on this thread, Max counters by leaning toward Kay's laptop (Fig. 5e) to point at and call out the repeat-loop syntax on her screen (Fig. 5f, 19), then pointing to Kay in a matter-of-fact way (Fig. 5.23–24).

When Kay accurately pushes back that the repeat-loop syntax on screen is not the same as the for-loop syntax in her journal (Fig. 5.25), Nya notes that the repeat-loop syntax would work, saying, "No:: you could use that" (Fig. 5.29). Max asserts that the two types of loops are the same (Fig. 5.30). Up to this point, Nya and Max consistently position themselves in favor of using the loop, regardless of which syntax, and have publicly misaligned with each of Kay's negative evaluations of using the loop. Kay seems to agree with Nya and Max that the repeat-loop syntax could work (Fig. 5.32–33) before voicing a sound, "Ttch" (Fig. 5.34), that perhaps marks misalignment again. Lou then rejoins the conversation, referring to the loop possibility: "I think I might actually do that BU::UT (.) I ↑already started so ah it's > too late now < " (Fig. 5.36–37,39). In this statement, Lou momentarily aligns with the positive evaluations of using the loop, but then returns to a negative evaluation of using the loop that emphasizes time efficiency ("it's too late now"). Kay directly aligns with Lou by stating, "I already started it too" (Fig. 5.41), and evaluates the prospect of using the loop relative to the work she's already accomplished: "and I'm already DOne with it so" (Fig. 5.43) with a toss of her hand (Fig. 5g).

At this point, the students temporarily cease talking about the loop. None of the students have changed their approach to the challenge. Throughout this exchange, the students adopt contrasting stances toward the loop. On one hand, Nya and Max provide a number of positive evaluations from their experience using the loop: "a lot less work," "so worth it," "it shows right here," and "you could use that." On the other hand, Kay and Lou provide negative evaluations of the possibility of using a loop: "too much work," "I don't feel like doing it," "I already started," and "I'm already done with it." These stances are not passive and independent expressions; the students are specifically responding dialogically to prior stances. Phrases like "actually," "yeah but," "no that's not," "no you could," "I might… but," and "…too" signal that the students are actively tracking and responding to how their stances relate to one another. Prior evaluations are elaborated with new information (e.g., Kay says the loop is "too much work," followed later with "cause I have to look in my journal") and then explicitly countered (e.g., Max shows that alternative syntax is readily available on Kay's screen).

In the above dialogue, we argue that participatory inequity between the students at this table is attenuated because each student at the table accesses the conversational floor and offers their evaluations on the topic. Similarly, this dialogue reflects a degree of attenuated relational inequity in that students are actively demonstrating that their ideas have enough impact to drive the focus of not only their collective reflections on code, but also

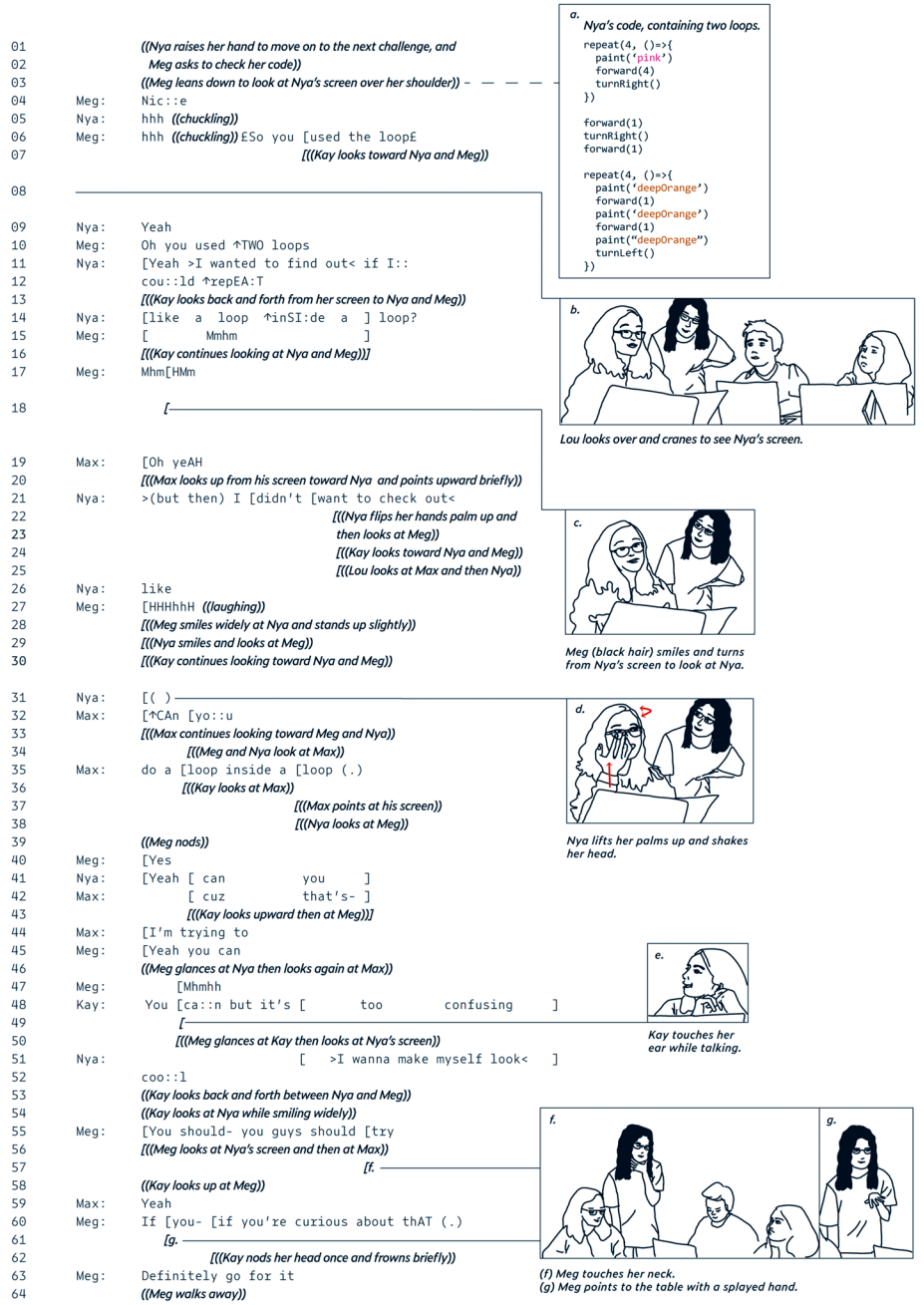
how they choose to proceed with the activity. In this exchange, the students are collaboratively reifying their sociocomputational norms. In terms of efficiency, one angle focuses on the amount of effort needed to (re)learn the loop while the other focuses on the amount of effort required to implement it once learned. Additional sociocomputational norms around embracing difficulty versus resourcefulness as well as around generic value ("so worth it") raised during the exchange show that students are appealing to distinct rationales in their evaluations when encouraging or resisting refactoring.

However, this discussion also reflects a degree of amplified relational inequity. As this dialogue extends the process of nuancing the classroom's sociocomputational norms around efficiency, using what you know, and embracing difficulty, it bears noting that some stances on these norms align with the public stances instructors have presented to the whole class, and as a result, may have a privileged status. That is, at the time that Nya, Max, Lou, and Kay were discussing the value of using loops, Meg had already made a whole-class statement about using loops versus not using loops (see the above subsection). In addition, about 30 seconds after Kay's last statement in the above transcript, Ben stops by this group's table, leans in to look at Max's code, and remarks, "Nice use of the loop," before walking away to a different table. This presents another angle on the amplified relational inequity in this context. Nya and Max align with the instructors, who have considerable influence in this classroom context to determine what coding activities students pursue and what is publicly reified as a valuable approach to those activities. Kay's stances on what would be inefficient for her and what she would prefer not to use are repeatedly countered by two of her peers, while these same peers' approach receives public praise from an instructor.

Part 5: Further misalignment through discussion of nested loops

Shortly after the dialogue examined above, Nya finishes coding a solution that uses two repeat-loops (see Fig. 6a for a screenshot of Nya's code) and raises her hand. In this classroom, raising one's hand is typically done either to request help or to solicit approval from an instructor to move on. Lou asks if Nya is "done already," not only indicating he thinks Nya is ready to move on, but also assuming Nya does not need help. Nya confirms that she's finished, and then Lou asks, "Oh, the loop?" In these turns, Lou foregrounds efficiency in terms of rapid completion ("done already"), and makes visible his awareness of the loop approach Nya used. When Meg visits the table to check on why Nya is raising her hand (Fig. 6.1–3), the following exchange takes place:

Meg looks at Nya's screen and remarks, "Nic::e" (Fig. 6.4). Both Nya and Meg chuckle (Fig. 6.5–6), before Meg notes that Nya used the loop (Fig. 6.6). Meg then marks surprise ("Oh") that Nya "used ↑TWO loops" (Fig. 6.10). These statements draw the attention of Kay and Lou (Fig. 6.7, b). With upward prosody indicative of a questioning or uncertain tone, Nya says to Meg, "Yeah > I wanted to find out < if I:: cou::ld ↑repEA:T like a loop ↑inSI:de a loop?" (Fig. 6.11–12, 14). Meg confirms twice (Fig. 6.15, 17) while smiling at Nya (Fig. 6c). Max then joins in ["Oh yeAH" (Fig. 6.19–20)], and Nya expresses more uncertainty about pursuing this approach (Fig. 6.21), flipping her palms up and looking at Meg (Fig. 6.22–23). Alongside Meg's laughter and smiling (Fig. 6.27–28), Nya seems to continue expressing uncertainty (Fig. 6d) as Max directly asks: "↑CAn yo::u do a loop inside a loop (.)" (Fig. 6.32–33, 35). Meg and Nya acknowledge Max's question by looking at Max (Fig. 6.34). Meg again confirms that nested loops are possible (Fig. 6.39–40), evaluating it as a tenable



01 ((Nya raises her hand to move on to the next challenge, and
02 Meg asks to check her code))
03 ((Meg leans down to look at Nya's screen over her shoulder)) - — — —
04 Meg: Nic::e
05 Nya: hhh ((chuckling))
06 Meg: hhh ((chuckling)) £So you [used the loop£
07 [((Kay looks toward Nya and Meg))
08 —
09 Nya: Yeah
10 Meg: Oh you used ↑TWO loops
11 Nya: [Yeah >I wanted to find out< if I::
12 cou::ld ↑repEA:T
13 [((Kay looks back and forth from her screen to Nya and Meg))
14 Nya: [like a loop ↑inSI:de a] loop?
15 Meg: [Mmhm]
16 [((Kay continues looking at Nya and Meg))]
17 Meg: Mhm[HMm
18 [—
19 Max: [Oh yeAH
20 [((Max looks up from his screen toward Nya and points upward briefly))
21 Nya: >(but then) I [didn't [want to check out<
22 [((Nya flips her hands palm up and
23 then looks at Meg))
24 [((Kay looks toward Nya and Meg))
25 [((Lou looks at Max and then Nya))
26 Nya: like
27 Meg: [HHHhhH ((laughing))
28 [((Meg smiles widely at Nya and stands up slightly))
29 [((Nya smiles and looks at Meg))
30 [((Kay continues looking toward Nya and Meg))
31 Nya: [( ) —
32 Max: [↑CAn [yo::u
33 [((Max continues looking toward Meg and Nya))
34 [((Meg and Nya look at Max))
35 Max: do a [loop inside a [loop (.)
36 [((Kay looks at Max))
37 [((Max points at his screen))
38 [((Nya looks at Meg))
39 ((Meg nods))
40 Meg: [Yes
41 Nya: [Yeah [can you]
42 Max: [cuz that's-]
43 [((Kay looks upward then at Meg))]
44 Max: [I'm trying to
45 Meg: [Yeah you can
46 ((Meg glances at Nya then looks again at Max))
47 Meg: [Mhmhh
48 Kay: You [ca::n but it's [too confusing]
49 [—
50 [((Meg glances at Kay then looks at Nya's screen))
51 Nya: [>I wanna make myself look<]
52 coo::l
53 ((Kay looks back and forth between Nya and Meg))
54 ((Kay looks at Nya while smiling widely))
55 Meg: [You should- you guys should [try
56 [((Meg looks at Nya's screen and then at Max))
57 [f. —
58 ((Kay looks up at Meg))
59 Max: Yeah
60 Meg: If [you- [if you're curious about thAT (.)
61 [g. —
62 [((Kay nods her head once and frowns briefly))
63 Meg: Definitely go for it
64 ((Meg walks away))

Fig. 6 Transcript detailing Nya and Max's inquiry about the possibility of using nested loops

target (Fig. 6.55, 60) and aligning with Nya and Max. Throughout this dialogue, Kay and Lou continue looking at Nya, Meg, and Max (Fig. 6.13, 16, 24–25, 30, 36, 43).

By asking about nested loops and stating both interest in and existing effort to explore nested loops ("I'm trying to"), Nya and Max introduce a fourth potential approach to the classroom's set of considerations (nested loops added to copy-and-paste, hard code, and repeat-loop). These stances position Nya and Max as curious about the possibility of refactoring using the nested loops. Meg's confirmation of nested loops as possible positions herself as supportive of their inquiry and in alignment with Nya and Max's curiosity. In the next turn of talk, Kay responds by affirming that nested loops are possible ("You ca::n"), temporarily aligning with Nya, Max, and Meg, but follows with, "but it's too confusing" (Fig. 6.48, e), evaluating nested loops in a contrasting way and thus creating misalignment by positioning herself against the nested loops approach. Aside from Meg glancing at Kay (Fig. 6.50), Kay's comment is not outwardly addressed in the coming turns. Instead, and overlapping with Kay, Nya contributes, "> I wanna make myself look < coo::l" (Fig. 6.51–52), which Kay then responds to with a wide smile (Fig. 6.54). This statement from Nya, however playful, positions Nya in support of using nested loops and evaluates it in relation to how pursuing the loop will "look" to others, instantiating how the discussion and pursuit of the loop has become embedded in the social space of the classroom. Meg directs the students, in particular Nya and Max (her eye gaze is directed at them, Fig. 6.56), to try out the nested loops idea, linking this exploration to curiosity: "You should- you guys should try if you- if you're curious about thAT (.) Definitely go for it" (Fig. 6.55, 60, 63).

Meg's alignment with Nya and Max's inquiry into nested loops, through affirming statements and positive affect, presents a direct attenuation of relational inequity for Nya and Max. Meg both demonstrates that their ideas have impact by encouraging them to pursue that coding approach, and demonstrates that their ideas are praiseworthy. However, this discourse inadvertently continues to amplify relational inequity between Kay and the instructors. The absence of discussion of Kay's evaluation that nested loops are confusing marks that this stance has less traction in the group's discussion. In addition, there is amplified participatory inequity for Kay and Lou during this stretch of dialogue. All the turns of talk are between Meg, Nya, and Max; Kay's one contribution does not receive a response. From the perspective of sociocomputational norms, we also see that this interaction picks up on the notion of fearlessness Meg highlighted at the outset of the activity: When students pursue a coding approach that they want to "find out" and "check out," and which is viable and extends a position already publicly endorsed by the instructors, the instructors actively encourage this computational exploration.

Part 6: Continued emphasis on refactoring using loops

Soon after this conversation, David, a researcher–instructor in the room (and as noted in the Methods section, a co-author of this paper), walks over to the table. David proposes to Lou that the loop "will save you a lot of time" and is "really cool," aligning with the stance on efficiency that highlights quickness and with Nya's statement about being seen as "cool." David nudges Lou to work with Nya to "learn how to do it." David then turns to Kay and assists in helping refactor her code using the repeat-loop. David aligns with Kay, noting the difference in syntax between the repeat-loop and the for-loop: "It's a slightly different syntax." The researcher–instructor's focus on refactoring Kay's code into a loop amplifies relational inequity; what could be seen as neutral instruction about a loop can be

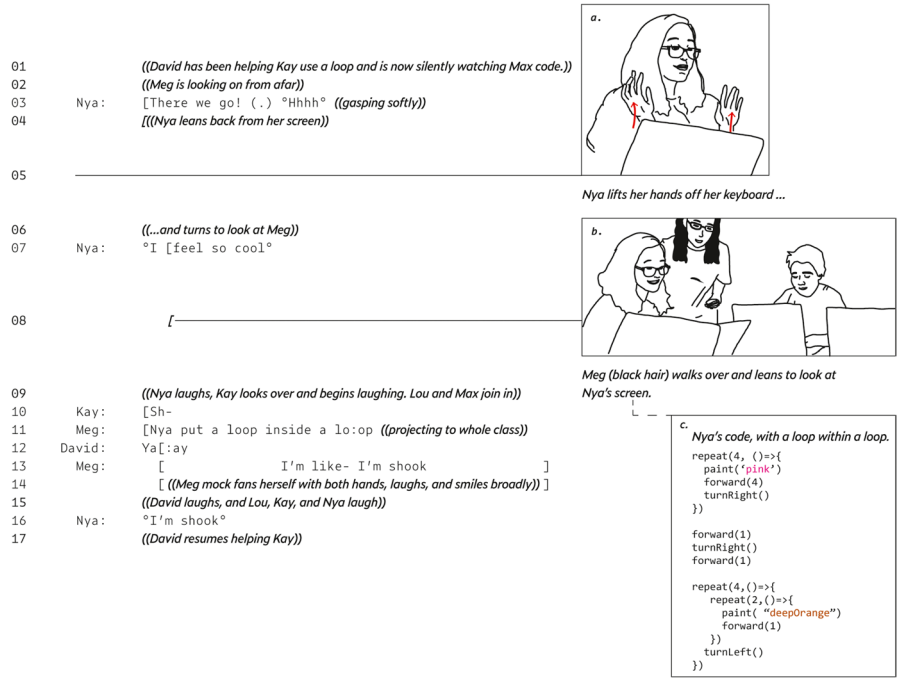


Fig. 7 Transcript detailing public celebration of Nya using the nested loops approach

seen in this context as stopping Kay from continuing to pursue an approach that she had repeatedly defended.

In the middle of Kay's refactoring, Nya completes the challenge using nested loops (see Fig. 7c for her code). Once her code finishes running and correctly paints the target image, Nya broadcasts her excitement by gasping, leaning back from her screen, and visibly lifting her hands off her keyboard (Fig. 7.3–4, a). This reaction is not unlike Ash's after his copy-and-paste discovery. She turns toward Meg (Fig. 7.6), who had been walking between tables, then marks the moment with a hushed statement, "°I feel so cool°" (Fig. 7.7), appealing to the evaluation and status she stated earlier. After Nya's statement, Meg walks over to the table and again leans in to look at Nya's screen (Fig. 7b). Nya laughs, and the other students join in (Fig. 7.9). Meg broadcasts to the whole class, "Nya put a loop inside a lo:op" (Fig. 7.11), and David shares excitement with, "Ya:ay" (Fig. 7.12). Meg then places emphasis on being "°shook°" (Fig. 7.13), colloquial for shocked, by mock fanning herself (Fig. 7.14), while the others at the table lightheartedly laugh along (Fig. 7.15). Nya aligns with Meg by also stating, "°I'm shook°" (Fig. 7.16). Again, the instructors and Nya publicly celebrate the nested loops solution, evaluating it as doable, path-breaking, praiseworthy, and an overall better solution. In the coming minutes, Ben revisits the table and alludes to the sociocomputational norm of efficiency with respect to quickness by saying to Nya, "Oooh very fast." Later, Nya responds to Kay's question about why she used nested loops with a similar evaluation on efficiency and ease of approach: "Cause it saves you time. You just loop the loop."

Ben, who had been helping other students at the back right table figure out how to use the repeat-loop in Part 3, moved to the front left table while the discussion at Kay, Nya, Lou, and Max's table progressed in Part 4. At the front left table, Ben begins to guide

Ash towards using the loop, similar to how David began to support Kay. Ash jokes with Ben by dragging the opacity slider on the screen to feign completing the challenge. In response, Ben remarks, "I know your trick, you're not going to fool me again." Ben then says, "Let me see some repeat-loops." In contrast to Meg's earlier hedges in discussion with Ash ("maybe think about using a loop") and Nya and Max ("if you're curious about that"), Ben requests repeat-loops specifically. Ben then temporarily leaves Ash to work on the loop and returns to supporting other students. Ash starts to explore using the loop, but in this class session, does not publicly align with the instructors' evaluation of the loop as more efficient.

End of the activity

By the end of this session, Nya and Max use nested loops; Ash, Eva, Ian, and Kay attempt using the loop; Rio uses copy-and-paste; and Lou, Ora, and Pat use hard code (without using copy-and-paste). The remaining students' approaches are unknown due to missing data. Only Nya, Max, and Rio complete the challenge. As Meg gathers the class back together for a debrief of the activity, Meg invites students to raise their hands to share their solutions with the whole class and calls on Max to start the discussion. After Max shares his nested loops approach, Meg states that students do not "have to do the loop inside the loop, but the important thing to know is that if you ever find that you're copy and pasting code a lot, think about a better way to do it—or a more efficient way to do it." With students in the class chiming in along the way, Meg then breaks down the components of the loop and collaboratively writes code with a loop on a laptop that is screen projected at the front of the room on the whiteboard.

Discussion

Summary of the refactoring classroom episode

Refactoring is not only a widespread practice in professional programming spaces (Demeyer et al., 2005; Fowler, 2019; Ionescu et al., 2020), but also a common and valued feature of programming learning environments (Romeike & Göttel, 2012; Stoecklin et al., 2007). However, studies of the micro details of how refactoring conversations arise in the classroom, including with critical attention to how students learn to decide when, how, and why to refactor, are rare in the computer science education research literature, a hole that warrants attention given persistent theoretical and empirical work on inequities regarding whose voices shape programming learning environments most (Ryoo et al., 2020; Shah & Lewis, 2019; Turkle & Papert, 1990). To narrow this gap, we sought in this case study to characterize how students and instructors in a classroom—including a co-author of this paper—adopted contrasting stances on the value of refactoring during classroom discourse, and we attended to participatory and relational inequity in their conversations as well as the classroom's refinement and application of particular views on sociocomputational norms. In this session, Meg, the instructor, began by framing the coding activity around general classroom norms: be resourceful with what you know; embrace difficulty/struggle, do not be afraid of it; and work quickly and efficiently (Part 1). Over time, instructors and students shifted these norms into computing-specific

terms. What started broadly as goals of "embracing difficulty" and "working efficiently" became tied to disrupting a known or working solution to pursue what most of the students in the room considered a newer coding approach that instantiated a particular type of efficiency (e.g., loops can be written in fewer lines of code) (Parts 2–4). In this way, as the activity progressed, particular sociocomputational instantiations of these norms gained unequal value across the room. The instructors and the researcher–instructor outwardly valued a solution they deemed efficient and elegant, nudging students to refactor their code using loops (Parts 5–6).

On the one hand, we note as a strength that students in this classroom debated code efficiency, voiced their preferred stances, and foregrounded a viable rationale for moving from copy-and-pasting to using loops. Additionally, the students, not just the instructors, engaged in defining and nuancing the classroom's sociocomputational norms. Indeed, throughout our analysis, we documented frequent instances of attenuated participatory inequity and facets of attenuated relational inequity, evidenced by students utilizing the conversational floor to voice their stance on refactoring, and by students and instructors taking students' written code and public stances as impactful points of departure for continued discussions about refactoring. In short, this was a classroom where students and instructors were contributing to the conversation and responding (on the whole) to each other's coding approaches and ideas. These features of the class session, however, needed to be further examined relative to how classroom discourse amplified participatory and relational inequity over time. That is, what began as a suggestion (from Ash) to use the copy-and-paste technique turned into collective reflection on whether to use a loop, and swiftly transitioned into the instructors anchoring the coding challenge around a preferred way for students to complete it. From the perspective of amplified relational inequity, particular sociocomputational norms (e.g., fewer lines of code; fearlessness around particular coding approaches) were foregrounded and endorsed in stances that gave less praise (e.g., responses without words such as "nice," "cool", or "shook") and less impact (e.g., students precipitously being asked to consider alternative coding approaches) to ideas not centered on loops. In addition, from a participatory inequity standpoint, discussions between instructors and students eventually gave more voice to students coding or refactoring using loops, including in the whole-class debrief activity.

In line with CSCL scholarship attending to the central role that (negotiations of) norms play in collaborative learning (e.g., Danish et al., 2020; Overdijk et al., 2014; Siyahhan et al., 2010), the detailed analysis in this paper speaks to the need for a careful reconsideration of how coding activities are framed, how students are invited to modify already working approaches, and how sociocomputational norms are introduced, nuanced, and applied. In all, we argue that programming instructors and designers should reflect carefully on when and how to approach discussion with students about refining working solutions and disciplinary standards of elegance, efficiency, and communicability (K-12 Computer Science Framework Steering Committee, 2016; Papert, 1980).

Implications for instructors and researchers

This analysis raises fundamental questions about how and when instructors of students new to programming introduce and support refactoring. Ruminating on similar considerations, Papert (1980) cautioned educators to wait until students were interested in refactoring code; the quote from our literature review captured a student stating, "I don't think

I'll do it that way." However, this strong centering of students' agency can be in tension with other considerations, such as a community's learning goals, project or activity goals, and resource and time constraints. An alternative move involves designing activities with constraints (e.g., see Abrahamson & Sánchez-García, 2016 work in mathematics and sports)—such as line limits used in future PixelBots activities—that require students to explore new programming techniques. For example, Rich et al. (2018) proposed a learning trajectory around the concept of repetition, beginning with students investigating "the idea that repetition is used for many tasks," and then expanding to address "the need to repeat instructions via a simple, countable loop" and eventually "recognizing the power of repetition ('Repeating things can have a cumulative effect.') and how and when to stop a repetition" (p. 52). Rich and colleagues acknowledge that trajectories like these still need to be treated as cyclically revisable hypotheses about what will make sense to students and when (e.g., Sikorski & Hammer, 2017). While instructor-designed constraints may push students toward an expanded array of coding techniques at the right moment in their learning trajectory, these moves may background opportunities for students to notice and discuss when and why to refactor. This is because the decision over whether a given coding approach is worthwhile is offloaded to the instructor or the task designer who sets up a particular coding challenge to require a particular coding technique.

An alternative approach to teaching refactoring in CS education is to acknowledge that refactoring is inherently uncertain, and in turn, necessitates dedicated time and pedagogical care toward inviting students into the process of explicitly reasoning about refactoring. We suggest a blend of the two approaches described in the previous paragraph, where instructors and students design goals and coding activities that over time expand students' knowledge of coding approaches, but where classroom conversations along the way center students' agency to discuss whether to refactor. What would it mean to give students, especially students new to the practice, foundational experiences and tools helpful to navigating refactoring, including a critical awareness of how and why refactoring practices developed in the professional programming world? While some learning trajectories suggest that discussions about refactoring are "advanced" territory (Rich et al., 2019, p. 749), we anticipate that conversations around sociocomputational norms and refactoring will happen early and often in programming learning environments. If this conjecture is right, educators could support new programmers to examine the perennially complex nature of deciding when to revise working code. One approach may be to create programming learning activities in the spirit of productive failure designs (Kapur, 2008; Kapur & Kinzer, 2009), in which students could grapple with ill-structured problems that ultimately promote and value the kinds of discussions around sociocomputational norms we saw in our data. These discussions might prioritize educational values such as students' rightful presence (Calabrese Barton & Tan, 2019), epistemological pluralism (Turkle & Papert, 1990), funds of knowledge (Shaw et al., 2020), and computational action (Tissenbaum et al., 2019). Toward these ends, students may experience attenuated participatory and relational inequity in classrooms when refactoring is presented as inherently difficult to evaluate, a topic worthy of sustained argumentation in the classroom and reflective of the goals centered by students and educators in CS education. Perhaps a transparent discussion honoring the heterogeneous interpretations of efficiency, fearlessness, and knowledge would have supported a deeper inquiry into the benefits and drawbacks of each of the approaches taken by the students in our focal classroom. In this way, we argue that students can better understand the value of refactoring when it is less of a

top-down expectation and more of a chance to explore for themselves the circumstances when it is most useful. We hope that the interaction analysis and theoretical framework in this paper invite design-based research-practice partnerships that attend carefully to participatory and relational inequity relative to the sociocomputational norms that motivate refactoring.

Over 30 years ago, Turkle and Papert (1990) conducted case studies of student programmers and argued that computer science learning environments should embrace epistemological pluralism. Through this charge, they foregrounded students' agency to choose a coding style over defaulting to canonical programming forms. More recently, Sengupta et al. (2021) wrote, "Voicing Code in STEM: A Dialogical Imagination," in which they argued that code is a form of expression, enacted through the assembly of multiple semiotic resources, and laden with the facets of power, communicative purpose, and identity found in expressions with human language. In the same year, Tissenbaum et al. (2021) pushed back on how "computing education has continued to center economic paradigms" (p. 1165), and instead envisioned four alternative endpoints for computing education: "impacting local communities and immediate needs," "data literate athletes and healthy citizens," "means of personal expression and social creative expression," and "blue collar computing." As the field of CS educators continues to expand the array of possible valued endpoints, we hope that the framework and findings from our paper can contribute to CS educators and researchers' efforts to center questions about sociocomputational norms and participatory and relational inequity. For instance, for each of the four valued endpoints Tissenbaum et al. (2021) cover, what sociocomputational norms might educators and students focus on in their coding inquiries, what participation structures can best guarantee that students have voice and impact along the way, and lastly, how do these considerations shape what role (if any) refactoring would play in students' coding activities? To continue to foreground the focus in CSCL and learning sciences literatures on the nuances of stancetaking in collaborative work (e.g., Philip et al., 2018; Simpson et al., 2017), we hope that CS education research continues to align with a recent wave of research that foreground discourse in computer science and robotics classrooms (Elliott, 2020; Ryoo et al., 2020; Shah & Lewis, 2019; Silvis et al., 2022).

Attending to inequity in computing and beyond

In our analysis, we found that participatory and relational inequity shifted throughout the stances students and instructors enacted in the classroom. In Parts 1–4, participatory inequity was attenuated as many students entered the conversational floor to voice their stances on how to navigate the coding challenge, even as they marked misalignment with instructors. In these early moments, the sociodisciplinary norms in the classroom were still open for negotiation, leaving space for students to use their preferred approach to solve the challenge. However, as the activity progressed, relational inequity remained amplified as the instructors and researcher–instructor consistently endorsed the loop (Parts 3–6). The instructors' endorsement of loops promoted sociocomputational norms that aligned with the curricular learning objectives but curtailed what was emerging as a potentially rich conversation about what constitutes efficiency, when short-term efficiency might yield to long-term efficiency, and how to balance these considerations during learning. With consistent nudging from instructors that one pathway (loops) through the activity was most valuable, the students in the

classroom, eventually including Ash and Kay, stopped defending their stances against the loop and started working on how to write a loop. The influence of the instructors and researcher–instructor, and their vocal support of a few students who aligned with using the loop, cut short the possibility for a generative discussion about the sociocomputational norm of efficiency, resulting in what Philip et al. (2018) have described as students' "failed bids for ideological expansion" (p. 215), a form of "*too early* ideological convergence, without adequate engagement with ideologically expansive stances, [which] constrains learning" (p. 185, original italics). With a focus on promoting a particular type of efficient coding and celebrating those approaches to coding, the instructors and the researcher–instructor may have inadvertently limited students' outward interest in pursuing and defending stances on the activity that misaligned with the discipline's specific version of sociocomputational norms.

As an alternative approach, instructors could consider centering values such as rightful presence (Calabrese Barton & Tan, 2019), using status treatments (Cohen & Lotan, 1995), and attending to how students assert their agency in CS classrooms (Ryoo et al., 2020; Vakil, 2020), all in service of elevating the positions of students who are choosing coding approaches that may not align with disciplinary norms, to make space for these students' continued discussion and exploration of the value of these approaches with their peers. The students' efforts to contest these norms suggest that there are opportunities in computing classrooms, for newcomers and experts alike, to have an expanded discussion about where these norms come from, how they might operate in the classroom, and how they can navigate these conversations in the professional world. For example, in Part 3, how might the discussion between Ben and Sal ended differently if Ben instead agreed that copy-and-paste was a fast way to type repeating lines of code? In Part 5, how might Kay be perceived differently by her peers if Meg agreed that nested loops were confusing?

Taking a step back even further, this tension in differing interpretations of sociocomputational norms between students and instructors are reminiscent of more general tensions between students and "authentic" computer scientists. Though our analysis focused on just one classroom session, we hope that this reflection on the broader social and cultural context of computing can benefit others moving similar work forward. Moreover, we argue that couching this work in broader computing contexts is essential against the backdrop of calls within CS education to enact professional disciplinary norms (Kolikant & Pollack, 2004) and the recognition that there are persistent economic framings of the value of programming (see also Lee et al., 2022; Tissenbaum et al., 2021). Computer science is not a neutral set of programming techniques separate from the values of society. Indeed, Philip and Sengupta (2021) have argued for learning theories to grapple with power through a lens that attends to features of context (i.e., the historically situated features of learning design), consequentiality (i.e., agency in the classroom to enact a particular stance), and contrapuntality (i.e., resistance to empire in society). They invite us to consider: "Who is simultaneously integral and invisiblized through imperialism in our conceptualizations of learning?" (p. 336). From a consequential lens, students whose stances misaligned with the instructors continually experienced amplified relational inequity. As the rationales for their stances faded into the background, and as the lesson became more focused on the singular expectation that students should pursue a loop approach, we would also note that the lesson carried an implicit contrapuntal message: default to canonical forms of knowledge. While this message aligns with canonical sociocomputational norms and may seem like a positive outcome for students in a classroom, considerations around

refactoring in the professional coding world are immensely complex (Ionescu et al., 2020; Stoecklin et al., 2007) and tangled up in economic decisions and inequities in the workforce (Philip & Sengupta, 2021). These angles on the data raise important questions about whether uncontested sociocomputational norms have a meaningful place in computer programming learning experiences.

Limitations

As with most interaction analysis studies, one limitation of this paper is that we examined a brief stretch of interaction and cannot make claims about how social and cultural influences from outside the classroom may have impacted interactions or how these interactions may have shaped long-term outcomes related to students' learning or computing identity formation. This limitation could be addressed by future ethnographic and longitudinal studies of interaction that attend to emerging and stable practices (see Keifert, 2021). In addition, the analysis attends to relational and amplified inequity in terms of publicly observable features of interaction (Shah & Lewis, 2019); follow-up work could additionally conduct video-cued reflections in which participants have an opportunity to observe the data and share their thoughts on what transpired (e.g., see DeLiema et al., 2023). This methodological move would allow students to comment on how they feel in moments of amplified participatory and relational inequity in classroom discourse. Even if these video-cued reflections were used as a unique data source (e.g., not just as triangulated confirmation of what could be seen in the classroom video), it might also shed light on what students notice and find meaningful about negotiating stances such as this in the classroom. In addition, this study focused on instructors and a researcher–instructor who were still in the early stages of developing their teaching practice in computer science. Veteran instructors may approach refactoring in the classroom in different ways. Nonetheless, we hope that the cautionary findings from our study help support professional development for educators learning to teach CS, and perhaps also open up new insights for veteran CS educators. Lastly, as a study that emerged late in our design-based research partnership, we were unable to redesign the learning environment with attention to the participatory and relational inequities raised in this paper. Considerably more work is needed to imagine and study computing learning environments for newcomers that empower students to reflect on the complexity of refactoring decisions.

Appendix A

Table 2 Transcription Key

[]	Left and right brackets indicate overlapping talk or action
(0.0)	Numbers in parentheses indicate pause in seconds
(.)	Dot in parentheses indicates a brief pause
word-	A dash after a sound indicates a beat between utterances
::	Double colon indicates prolongation of prior sound
↑↓	Upward and downward arrow indicate a shift into higher or lower pitch, respectively
?	Question mark indicates a rise in pitch
WORD	Upper case letters indicate louder sounds
°word°	Degree signs surrounding talk indicates softer sounds
((action))	Description in bold, italicized font and surrounded by two pairs of parentheses indicates action
> <	Right then left carats indicate hurried or faster talk
hhh	Consecutive "h" letters indicate breathiness or gasping
(word)	Parentheses containing word(s) indicate possibilities for what was said
()	Empty parentheses indicate undecipherable speech. Wider space indicates a longer period of unknown speech
#word#	Pounds or hashtags surrounding talk indicates the speaker was smiling
£word£	British pound sterling symbols surrounding talk indicates suppressed laughter

Acknowledgements This work was supported by the National Science Foundation under grant nos. 1612770, 1607742, and 1612660. We wish to express deep gratitude to the students and educators who collaborated on this research. Our CSCL reviewers also provided generative feedback during the peer review process, and we wish to thank them for their contributions. We are also grateful for the time that Geoffrey Herman, Colleen Lewis, members of our NSF advisory board, and UMN graduate students in the "Debugging Failure" course gave to share their helpful feedback on early analyses and drafts of this paper.

References

Abrahamson, D., & Sánchez-García, R. (2016). Learning is moving in new ways: The ecological dynamics of mathematics education. *Journal of the Learning Sciences, 25*(2), 203–239. https://doi.org/10.1080/10508406.2016.1143370

Alves, N. S. R., Mendes, T. S., de Mendonça, M. G., Spínola, R. O., Shull, F., & Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology, 70*, 100–121. https://doi.org/10.1016/j.infsof.2015.10.008

Ames, M. G. (2018). Hackers, Computers, and Cooperation: A Critical History of Logo and Constructionist Learning. Proceedings of the ACM on Human-Computer Interaction, 2(CSCW), 1–19. https://doi.org/10.1145/3274287

Bang, M., & Vossoughi, S. (2016). Participatory design research and educational justice: Studying learning and relations within social change making. *Cognition and Instruction, 34*(3), 173–193. https://doi.org/10.1080/07370008.2016.1181879

Boaler, J. (2008). Promoting 'relational equity' and high mathematics achievement through an innovative mixed-ability approach. *British Educational Research Journal, 34*(2), 167–194. https://doi.org/10.1080/01411920701532145

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Proceedings of the 2012 Annual Meeting of the American Educational Research Association.

Calabrese Barton, A., & Tan, E. (2019). Designing for rightful presence in STEM: The role of making present practices. *Journal of the Learning Sciences, 28*(4–5), 616–658. https://doi.org/10.1080/10508406.2019.1591411

Cohen, E. G., & Lotan, R. A. (1995). Producing equal-status interaction in the heterogeneous classroom. *American Educational Research Journal, 32*(1), 99–120. https://doi.org/10.3102/00028312032001099

Cress, U., Rosé, C., Wise, A. F., & Oshima, J. (Eds.). (2021). *International Handbook of Computer-Supported Collaborative Learning* (Vol. 19). Springer International Publishing. https://doi.org/10.1007/978-3-030-65291-3

Dahn, M., & DeLiema, D. (2020). Dynamics of emotion, problem solving, and identity: Portraits of three girl coders. *Computer Science Education, 30*(3), 362–389. https://doi.org/10.1080/08993408.2020.1805286

Dahn, M., Deliema, D., & Enyedy, N. (2020). Art as a point of departure for understanding student experience in learning to code. *Teachers College Record, 122*(8), 1–42. https://doi.org/10.1177/016146812012200802

Danielak, B. (2022). How Code Takes shape: Studying a student's program evolution. *Cognition and Instruction, 40*(2), 266–303. https://doi.org/10.1080/07370008.2022.2044330

Danish, J. A., Enyedy, N., Saleh, A., & Humburg, M. (2020). Learning in embodied activity framework: A sociocultural framework for embodied cognition. *International Journal of Computer-Supported Collaborative Learning, 15*(1), 49–87. https://doi.org/10.1007/s11412-020-09317-3

Davies, B., & Harré, R. (1990). Positioning: The discursive production of selves. *Journal for the Theory of Social Behaviour, 20*(1), 43–63. https://doi.org/10.1111/j.1468-5914.1990.tb00174.x

DeLiema, D., Dahn, M., Flood, V. J., Asuncion, A., Abrahamson, D., Enyedy, N., & Steen, F. (2020). Debugging as a context for fostering reflection on critical thinking and emotion. In E. Manalo (Ed.), *Deeper Learning, Dialogic Learning, and Critical Thinking* (1st ed., pp. 209–228). Routledge. https://doi.org/10.4324/9780429323058-13

DeLiema, D., Hufnagle, A., Rao, V. N. V., Baker, J., Valerie, J., & Kim, J. (2023). Methodological innovations at the intersection of video-based educational research traditions: Reflections on relevance, data selection, and phenomena of interest. *International Journal of Research & Method in Education*, *46*(1), 19–36. https://doi.org/10.1080/1743727X.2021.2011196

DeLiema, D., Kwon, Y. A., Chisholm, A., Williams, I., Dahn, M., Flood, V. J., Abrahamson, D., & Steen, F. F. (2022). A multi-dimensional framework for documenting students' heterogeneous experiences with programming bugs. *Cognition and Instruction*, *41*(2), 158–200. https://doi.org/10.1080/07370008.2022.2118279

Denner, J., Green, E., & Campe, S. (2021). Learning to program in middle school: How pair programming helps and hinders intrepid exploration. *Journal of the Learning Sciences, 30*(4–5), 611–645. https://doi.org/10.1080/10508406.2021.1939028

Demeyer, S., Van Rysselberghe, F., Girba, T., Ratzinger, J., Marinescu, R., Mens, T., Du Bois, B., Janssens, D., Ducasse, S., Lanza, M., Rieger, M., Gall, H., & El-Ramly, M. (2005). The LAN simulation: A refactoring teaching example. Eighth International Workshop on Principles of Software Evolution (IWPSE'05), 123–131. https://doi.org/10.1109/IWPSE.2005.30

Derry, S. J., Pea, R. D., Barron, B., Engle, R. A., Erickson, F., Goldman, R., Hall, R., Koschmann, T., Lemke, J. L., Sherin, M. G., & Sherin, B. L. (2010). Conducting video research in the learning sciences: Guidance on selection, analysis, technology, and ethics. *Journal of the Learning Sciences, 19*(1), 3–53. https://doi.org/10.1080/10508400903452884

Dickes, A. C., Farris, A. V., & Sengupta, P. (2020). Sociomathematical norms for integrating coding and modeling with elementary science: A dialogical approach. *Journal of Science Education and Technology, 29*(1), 35–52. https://doi.org/10.1007/s10956-019-09795-7

Du Bois, J. W. (2007). The Stance Triangle. In R. Englebretson (Ed.), *Stancetaking in Discourse: Subjectivity, evaluation, interaction* (pp. 139–182). John Benjamins Publishing Company.

Elliott, C. H. (2020). "Run it through me:" Positioning, power, and learning on a high school robotics team. *Journal of the Learning Sciences, 29*(4–5), 1–44. https://doi.org/10.1080/10508406.2020.1770763

Enyedy, N. (2005). Inventing mapping: Creating cultural forms to solve collective problems. *Cognition and Instruction, 23*(4), 427–466. https://doi.org/10.1207/s1532690xci2304_1

Erickson, F. (1992). Ethnographic Microanalysis of Interaction. In M. D. LeCompte, W. L. Millroy, & J. Preissle (Eds.), *The Handbook of Qualitative Research in Education* (pp. 201–225). Academic Press.

Esmonde, I., & Booker, A. N. (Eds.). (2016). *Power and Privilege in the Learning Sciences: Critical and Sociocultural Theories of Learning* (1st ed.). Routledge.

Fowler, M. (2019). *Refactoring: Improving the design of existing code* (2nd ed.). Addison-Wesley Professional.

Gomez, K., Gomez, L. M., & Worsley, M. (2021). Interrogating the Role of CSCL in Diversity, Equity, and Inclusion. In U. Cress, C. Rosé, A. F. Wise, & J. Oshima (Eds.), *International Handbook of Computer-Supported Collaborative Learning* (Vol. 19, pp. 103–120). Springer International Publishing. https://doi.org/10.1007/978-3-030-65291-3

Goodwin, C. (2006). Retrospective and prospective orientation in the construction of argumentative moves. *Text & Talk, 26*, 443–461. https://doi.org/10.1515/TEXT.2006.018

Goodwin, C. (2007). Participation, stance and affect in the organization of activities. *Discourse & Society, 18*(1), 53–73. https://doi.org/10.1177/0957926507069457

Goodwin, C. (2018). *Co-operative Action*. Cambridge University Press.

Gutiérrez, K. D., & Jurow, A. S. (2016). Social design experiments: Toward equity by design. *Journal of the Learning Sciences, 25*(4), 565–598. https://doi.org/10.1080/10508406.2016.1204548

Hennessy, E. C., Gendreau, C. A., Bush, J. B., Nixon, J., & Recker, M. (2023). Toward a debugging pedagogy: Helping students learn to get unstuck with physical computing systems. *Information and Learning Sciences, 124*(1/2), 1–24. https://doi.org/10.1108/ILS-03-2022-0051

Holland, D., Lachicotte, W., Jr., Skinner, D., & Cain, C. (2001). *Identity and Agency in Cultural Worlds*. Harvard University Press.

Ionescu, T. B., Schlund, S., & Schmidbauer, C. (2020). Epistemic Debt: A Concept and Measure of Technical Ignorance in Smart Manufacturing. In I. L. Nunes (Ed.), *Advances in Human Factors and Systems Interaction* (pp. 81–93). Springer International Publishing.

Jefferson, G. (2004). Glossary of Transcript Symbols with an Introduction. In G. H. Lerner (Ed.), *Conversation Analysis: Studies from the First Generation* (pp. 13–34). John Benjamins Publishing Company.

Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *Journal of the Learning Sciences, 4*(1), 39–103. https://doi.org/10.1207/s15327809jls0401_2

K-12 Computer Science Framework Steering Committee. (2016). *K-12 Computer Science Framework*. ACM. https://k12cs.org/

Kapur, M. (2008). Productive failure. *Cognition and Instruction, 26*(3), 379–424. https://doi.org/10.1080/07370000802212669

Kapur, M., & Kinzer, C. K. (2009). Productive failure in CSCL groups. *International Journal of Computer-Supported Collaborative Learning, 4*(1), 21–46. https://doi.org/10.1007/s11412-008-9059-z

Keifert, D. T. (2021). Family culture as context for learning through inquiry. *Cognition and Instruction, 39*(3), 242–274. https://doi.org/10.1080/07370008.2021.1913162

Kobiela, M., & Lehrer, R. (2015). The codevelopment of mathematical concepts and the practice of defining. *Journal for Research in Mathematics Education JRME, 46*(4), 423–454. https://doi.org/10.5951/jresematheduc.46.4.0423

Kolikant, Y.B.-D., & Pollack, S. (2004). Establishing computer science professional norms among high-school students. *Computer Science Education, 14*(1), 21–35. https://doi.org/10.1076/csed.14.1.21.23497

Koschmann, T., Kuutti, K., & Hickman, L. (1998). The concept of breakdown in Heidegger, Leont'ev, and Dewey and its implications for education. *Mind, Culture, and Activity, 5*(1), 25–41. https://doi.org/10.1207/s15327884mca0501_3

Langer-Osuna, J. M., & McKinney de Royston, M. (2017). Understanding Relations of Power in the Mathematics Classroom: Explorations in Positioning Theory. In A. Chronaki (Ed.), *Proceedings of the Ninth International Mathematics Education and Society Conference* (Vol. 2, pp. 645–653). University of Thessaly Press.

Lee, U.-S.A., DeLiema, D., & Gomez, K. (2022). Equity conjectures: A methodological tool for centering social change in learning and design. *Cognition and Instruction, 40*(1), 77–99. https://doi.org/10.1080/07370008.2021.2010211

Leyva, L. A., McNeill, R. T., Marshall, B. L., & Guzmán, O. A. (2021a). "It seems like they purposefully try to make as many kids drop": An analysis of logics and mechanisms of racial-gendered inequality in introductory mathematics instruction. *The Journal of Higher Education, 92*(5), 784–814. https://doi.org/10.1080/00221546.2021.1879586

Leyva, L. A., Quea, R., Weber, K., Battey, D., & López, D. (2021b). Detailing racialized and gendered mechanisms of undergraduate precalculus and calculus classroom instruction. *Cognition and Instruction, 39*(1), 1–34. https://doi.org/10.1080/07370008.2020.1849218

Lewis, C. M., & Shah, N. (2015). How equity and inequity can emerge in pair programming. Proceedings of the Eleventh Annual International Conference on International Computing Education Research - ICER '15, 41–50. https://doi.org/10.1145/2787622.2787716

Lopez, L. M., & Allal, L. (2007). Sociomathematical norms and the regulation of problem solving in classroom microcultures. *International Journal of Educational Research, 46*(5), 252–265. https://doi.org/10.1016/j.ijer.2007.10.005

Maxwell, J. A. (2013). *Qualitative research design: An interactive approach* (Third edition.). SAGE Publications, Inc.

Nader, L. (1996). *Naked Science: Anthropological Inquiry into Boundaries, Power, and Knowledge*. Routledge.

Overdijk, M., van Diggelen, W., Andriessen, J., & Kirschner, P. A. (2014). How to bring a technical artifact into use: A micro-developmental perspective. *International Journal of Computer-Supported Collaborative Learning, 9*(3), 283–303. https://doi.org/10.1007/s11412-014-9195-6

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas* (1st ed.). Basic Books, Inc.

Philip, T. M., & Azevedo, F. S. (2017). Everyday science learning and equity: Mapping the contested terrain. *Science Education, 101*(4), 526–532. https://doi.org/10.1002/sce.21286

Philip, T. M., & Gupta, A. (2020). Emerging perspectives on the co-construction of power and learning in the learning sciences, mathematics education, and science education. *Review of Research in Education, 44*(1), 195–217. https://doi.org/10.3102/0091732X20903309

Philip, T. M., Gupta, A., Elby, A., & Turpen, C. (2018). Why ideology matters for learning: A case of ideological convergence in an engineering ethics classroom discussion on drone warfare. *Journal of the Learning Sciences, 27*(2), 183–223. https://doi.org/10.1080/10508406.2017.1381964

Philip, T. M., & Sengupta, P. (2021). Theories of learning as theories of society: A contrapuntal approach to expanding disciplinary authenticity in computing. *Journal of the Learning Sciences, 30*(2), 330–349. https://doi.org/10.1080/10508406.2020.1828089

Posner, G. J., Strike, K. A., Hewson, P. W., & Gertzog, W. A. (1982). Accommodation of a scientific conception: Toward a theory of conceptual change. *Science Education, 66*(2), 211–227.

Radkowitsch, A., Vogel, F., & Fischer, F. (2020). Good for learning, bad for motivation? A meta-analysis on the effects of computer-supported collaboration scripts. *International Journal of Computer-Supported Collaborative Learning, 15*(1), 5–47. https://doi.org/10.1007/s11412-020-09316-4

Reason, J. (1990). *Human Error*. Cambridge University Press.

Rich, K. M., Strickland, C., Binkowski, T. A., & Franklin, D. (2019). A K-8 Debugging Learning Trajectory Derived from Research Literature. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 745–751. https://doi.org/10.1145/3287324.3287396

Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2018). K–8 learning trajectories derived from research literature: Sequence, repetition, conditionals. *ACM Inroads, 9*(1), 46–55. https://doi.org/10.1145/3183508

Romeike, R., & Göttel, T. (2012). Agile Projects in High School Computing Education: Emphasizing a Learners' Perspective. *Proceedings of the 7th Workshop in Primary and Secondary Computing Education*, 48–57. https://doi.org/10.1145/2481449.2481461

Ryoo, J. J., Tanksley, T., Estrada, C., & Margolis, J. (2020). Take space, make space: How students use computer science to disrupt and resist marginalization in schools. *Computer Science Education, 30*(3), 337–361. https://doi.org/10.1080/08993408.2020.1805284

Scheuer, O., Loll, F., Pinkwart, N., & McLaren, B. M. (2010). Computer-supported argumentation: A review of the state of the art. *International Journal of Computer-Supported Collaborative Learning, 5*(1), 43–102. https://doi.org/10.1007/s11412-009-9080-x

Schwarz, C. V., Reiser, B. J., Davis, E. A., Kenyon, L., Achér, A., Fortus, D., Shwartz, Y., Hug, B., & Krajcik, J. (2009). Developing a learning progression for scientific modeling: Making scientific modeling accessible and meaningful for learners. *Journal of Research in Science Teaching, 46*(6), 632–654. https://doi.org/10.1002/tea.20311

Sengupta, P., Dickes, A., & Farris, A. V. (2021). *Voicing code in STEM: A dialogical imagination*. MIT Press.

Shah, N., Christensen, J. A., Ortiz, N. A., Nguyen, A.-K., Byun, S., Stroupe, D., & Reinholz, D. L. (2020). Racial hierarchy and masculine space: Participatory in/equity in computational physics classrooms. *Computer Science Education, 30*(3), 254–278. https://doi.org/10.1080/08993408.2020.1805285

Shah, N., & Lewis, C. M. (2019). Amplifying and attenuating inequity in collaborative learning: Toward an analytical framework. *Cognition and Instruction, 37*(4), 423–452. https://doi.org/10.1080/07370008.2019.1631825

Shaw, M. S., Fields, D. A., & Kafai, Y. B. (2020). Leveraging local resources and contexts for inclusive computer science classrooms: Reflections from experienced high school teachers implementing electronic textiles. *Computer Science Education, 30*(3), 313–336. https://doi.org/10.1080/08993408.2020.1805283

Sikorski, T.-R., & Hammer, D. (2017). Looking for coherence in science curriculum. *Science Education, 101*(6), 929–943. https://doi.org/10.1002/sce.21299

Silvis, D., Clarke-Midura, J., Shumway, J. F., Lee, V. R., & Mullen, S. (2022). Children caring for robots: Expanding computational thinking frameworks to include a technological ethic of care. *International Journal of Child-Computer Interaction*, *33*, 100491. https://doi.org/10.1016/j.ijcci.2022.100491

Simpson, A., Bannister, N., & Matthews, G. (2017). Cracking her codes: Understanding shared technology resources as positioning artifacts for power and status in CSCL environments. *International Journal of Computer-Supported Collaborative Learning, 12*(3), 221–249. https://doi.org/10.1007/s11412-017-9261-y

Sinha, S., Rogat, T. K., Adams-Wiggins, K. R., & Hmelo-Silver, C. E. (2015). Collaborative group engagement in a computer-supported inquiry learning environment. *International Journal of Computer-Supported Collaborative Learning, 10*(3), 273–307. https://doi.org/10.1007/s11412-015-9218-y

Siyahhan, S., Barab, S. A., & Downton, M. P. (2010). Using activity theory to understand intergenerational play: The case of Family Quest. *International Journal of Computer-Supported Collaborative Learning, 5*(4), 415–432. https://doi.org/10.1007/s11412-010-9097-1

Stahl, G. (2006). *Group Cognition: Computer Support for Building Collaborative Knowledge*. MIT Press.

Stevens, R., & Hall, R. (1998). Disciplined Perception: Learning to See in Technoscience. In M. Lampert & M. L. Blunk (Eds.), *Talking mathematics in school: Studies of teaching and learning* (pp. 107–150). Cambridge University Press.

Stoecklin, S., Smith, S., & Serino, C. (2007). Teaching students to build well formed object-oriented methods through refactoring. *SIGCSE Bulletin, 39*(1), 145–149. https://doi.org/10.1145/1227504.1227364

Suryanarayana, G., Samarthyam, G., & Sharma, T. (2014). *Refactoring for software design smells: Managing technical debt*. Morgan Kaufmann.

Suzuki, H., & Kato, H. (1995). Interaction-Level Support for Collaborative Learning: AlgoBlock—An Open Programming Language. *The First International Conference on Computer Support for Collaborative Learning*, 349–355. https://doi.org/10.3115/222020.222828

Techapalokul, P., & Tilevich, E. (2019). Code Quality Improvement for All: Automated Refactoring for Scratch. *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 117–125. https://doi.org/10.1109/VLHCC.2019.8818950

The Politics of Learning Writing Collective. (2017). The learning sciences in a new era of U.S. nationalism. *Cognition and Instruction*, *35*(2), 91–102. https://doi.org/10.1080/07370008.2017.1282486

Thompson, C. (2020). *Coders: The making of a new tribe and the remaking of the world*. Penguin Books.

Tissenbaum, M., Sheldon, J., & Abelson, H. (2019). From computational thinking to computational action. *Communications of the ACM, 62*(3), 34–36. https://doi.org/10.1145/3265747

Tissenbaum, M., Weintrop, D., Holbert, N., & Clegg, T. (2021). The case for alternative endpoints in computing education. *British Journal of Educational Technology, 52*(3), 1164–1177. https://doi.org/10.1111/bjet.13072

Tsan, J., Vandenberg, J., Zakaria, Z., Boulden, D. C., Lynch, C., Wiebe, E., & Boyer, K. E. (2021). Collaborative Dialogue and Types of Conflict: An Analysis of Pair Programming Interactions between Upper Elementary Students. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 1184–1190. https://doi.org/10.1145/3408877.3432406

Turkle, S., & Papert, S. (1990). Epistemological pluralism: Styles and voices within the computer culture. *Signs, 16*(1), 128–157.

Vakil, S. (2020). "I've always been scared that someday i'm going to sell out": Exploring the relationship between political identity and learning in computer science education. *Cognition and Instruction, 38*(2), 87–115. https://doi.org/10.1080/07370008.2020.1730374

Van Dormolen, J., & Zaslavsky, O. (2003). The many facets of a definition: The case of periodicity. *The Journal of Mathematical Behavior, 22*(1), 91–106. https://doi.org/10.1016/S0732-3123(03)00006-3

Vinner, S. (2002). The Role of Definitions in the Teaching and Learning of Mathematics. In D. Tall (Ed.), *Advanced Mathematical Thinking* (pp. 65–81). Springer Netherlands. https://doi.org/10.1007/0-306-47203-1_5

Vossoughi, S., & Escudé, M. (2016). What does the camera communicate? An inquiry into the politics and possibilities of video research on learning. *Anthropology & Education Quarterly, 47*(1), 42–58. https://doi.org/10.1111/aeq.12134

Wang, X. C., Flood, V. J., & Cady, A. (2021). Computational Thinking through Body and Ego Syntonicity: Young Children's Embodied Sense-Making Using A Programming Toy. In E. de Vries, Y. Hod, & J. Ahn (Eds.), *Proceedings of the 15th International Conference of the Learning Sciences* (pp. 394–401). International Society of the Learning Sciences. https://repository.isls.org//handle/1/7494

Watkins, J., Hammer, D., Radoff, J., Jaber, L. Z., & Phillips, A. M. (2016). Positioning as not-understanding: The value of showing uncertainty for engaging in science. *Journal of Research in Science Teaching, 55*(4), 573–599. https://doi.org/10.1002/tea.21431

Weiner, B. (1985). An attributional theory of achievement motivation and emotion. *Psychological Review, 92*(4), 548.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society a: Mathematical, Physical and Engineering Sciences, 366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Yackel, E., & Cobb, P. (1996). Sociomathematical norms, argumentation, and autonomy in mathematics. *Journal for Research in Mathematics Education, 27*(4), 458–477. https://doi.org/10.2307/749877

Yin, R. K. (2009). *Case Study Research: Design and Methods* (4th ed., Vol. 5). SAGE Inc.

Authors and Affiliations

Morgan M. Fong[1] · David DeLiema[2] · Virginia J. Flood[3] · Oia Walker-van Aalst[4]

✉ Morgan M. Fong
mmfong2@illinois.edu

✉ David DeLiema
ddeliema@umn.edu

✉ Virginia J. Flood
vflood@buffalo.edu

Oia Walker-van Aalst
oia.walker.van.aalst@berkeley.edu

[1] University of Illinois Urbana-Champaign, Champaign, IL, USA

[2] University of Minnesota, Twin Cities, MN, USA

[3] University at Buffalo, SUNY, Buffalo, NY, USA

[4] Present Address: University of California, Berkeley, CA, USA