Using Distributed Cognition Theory to Analyze Collaborative Computer Science Learning

Elise Deitrick[♡], R. Benjamin Shapiro[†♡], Matthew P. Ahrens[♡],
Rebecca Fiebrink[♣], Paul D. Lehrman[♡], & Saad Farooq[♡]
[♡] Tufts University [♣] Goldsmiths, University of London
[†] r@benshapi.ro

ABSTRACT

Research on students' learning in computing typically investigates how to enable individuals to develop concepts and skills, yet many forms of computing education, from peer instruction to robotics competitions, involve group work in which understanding may not be entirely locatable within individuals' minds. We need theories and methods that allow us to understand learning in cognitive systems: culturally and historically situated groups of students, teachers, and tools. Accordingly, we draw on Hutchins' Distributed Cognition [16] theory to present a qualitative case study analysis of interaction and learning within a small group of middle school students programming computer music. Our analysis shows how a system of students, teachers, and tools, working in a music classroom, is able to accomplish conceptually demanding computer music programming. We show how the system does this by 1) collectively drawing on individuals' knowledge, 2) using the physical and virtual affordances of different tools to organize work, externalize knowledge, and create new demands for problem solving, and 3) reconfiguring relationships between individuals and tools over time as the focus of problem solving changes. We discuss the implications of this perspective for research on teaching, learning and assessment in computing.

Keywords

Learning, Research Methods, Music

1. INTRODUCTION

Nearly thirty years ago, Pea, Soloway, and Spohrer [24] argued that becoming a programmer requires developing new kinds of knowledge. They wrote: "For programming, as in other domains from mathematics to the physical and engineering sciences, students are engaged through their learning activities in actively building a knowledge system of concepts and procedural skills." This perspective suggests a need to develop "characterization[s] of a student's current understanding in terms of the knowledge he or she is utilizing to make sense of the problem solving activities in computer programming" (ibid.). But all of these years later, we still know relatively little about *how* learners develop knowledge in computing, especially in group work.

Researchers have developed numerous approaches to studying computer science learning. One overarching approach has been to mine student code, including logs of changes in students' code over time. Such work includes charting changes in students' use of language constructs and practices of recycling code from their own or others' projects [7]. Others have used students' code to investigate how students transfer program design patterns from game programming to making science simulations [3]. And still others have used time-series records of students' code to find "sink states" where students seem to be stuck [5]. Analysis of massive amounts of BlueJ data permitted researchers to identify mismatches between teachers' perceptions of common student needs and students' actual needs [9].

But investigators have not limited themselves to analyses of code alone, applying interview and think-aloud methods wherein students explain their code to researchers or to peers as they author, debug, use, or retrospectively discuss it [20, 10, 19]. Analysts of students' computational thinking have looked at discourse in small groups and within whole classroom conversations [30]. In most of this work, the unit of analysis has been the individual student or the individual student's idea: how a student's utterances (verbal or textual) represent particular disciplinary (e.g., computational, biological, or physical) ideas.

But shifts in how CS is learned make this individual frame problematic. Computer science education is increasingly collaborative, employing approaches like pair programming [22], peer teaching [25], and decomposition of projects into pieces to be done by individuals and then stitched together [13]. These structures mirror those found in industrial software development, and they can be quite effective at improving learning and retention [26]. However, there is little published work describing *how* these small groups solve problems and the *processes through which* participants develop computational thinking and programming skills over time. Moreover, assessing what students can do individually does not necessarily predict all that a team made up of those same individuals can accomplish, or what they can learn from one another. Teams may be able to accomplish more than the sum of their individual members' skills would suggest, but they may also get bogged down in group dynamics that hamper their capabilities. Further, we should not take for granted that groups of youth learning together will all emerge from their groupwork with identical understandings [28]; different group members may participate in projects in different ways, work on different pieces of overall problems, and have different goals for their own skill development. As we as a field continue to adopt collaborative learning, we need theories and methods that the richness of what happens in student groups. Specifically: We need research on the ways in which groups work together so that we can get to the process of learning: the micro-genesis of students' computer science thinking, at both the individual and the collective levels.

2. DEVELOPING AND APPLYING RELEVANT THEORY

Much Computing Education Research (CER) links pedagogies (or pedagogical interventions) to outcomes, but does not sufficiently explain why — i.e., through what mechanisms, and with what potential limitation — those outcomes are achieved. *Why* questions raise all manner of theoretical and methodological questions for CER researchers, who currently operate from a wide variety of perspectives [21, 32].

Studying group interactions in CS learning environments is a potent opportunity to adapt, apply, and refine theories of learning from the other fields into CER. There are many possible ways to do this: Some theoretical perspectives on group work give primacy to questions about cognition, such as about how disciplinary ideas are conceptualized. For example, discourse analysis of student group talk can reveal subtle discursive (e.g., metaphorical) processes through which members of groups can converge on shared understandings [28]. But conceptual development is not the only possible focus for research. Questioning power and equity within groups orients us to look at which group members hold disproportionate sway over joint activity, and the mechanisms through which this power is constructed, wielded, and perpetuated. For example, one inquiry into the distribution, content, and positioning of talk within groups of elementary students learning computer science found that even when a group looked equitable in terms of how much different group members were talking, they were quite inequitable at the level of peer perceptions of computing competence [29]. These dynamics can replicate existing societal inequalities [1] and may be difficult to intervene in, even given explicit role assignments [34]. Finally, social network analyses, drawing on the evolution of code similarity across the classroom, can enable insights into how social dynamics enable knowledge flow and co-construction in collaboration and competition [4].

2.1 Theories of Cognition

Most research on cognition in computing (and STEM generally), even research that attends to social dynamics in learning, ultimately focuses on how *individuals* develop computational ideas. As we shall now explain, this stance is deeply problematic, and new theories that adopt distributed, social, and material definitions for cognition are necessary to account for the richness of how learning actually happens. The current ubiquity of mentalist (i.e., focused on individual minds) approaches to understanding learning is unsurprising, as thinking and learning have historically been understood as things that individuals do. But social [33], situated [2, 12, 18], and cultural [11, 27] understandings of cognition and learning challenge this tradition. They suggest that human activities can best be understood as culturally- and historically-situated, technologically-mediated, socially-enacted processes. Knowledge development, in this view, is intrinsically linked to social context, and research on learning that strips away the contextuality of activity is incompatible with socio-cultural theories of learning [8]. This is not merely a theoretical concern: a century of research on learning has shown that it is extremely difficult for learners to transfer knowledge from one context to another [23, 6].

One reason for this is that different conditions (settings, problems, partnerships) can activate different knowledge in learners' minds; learners frequently develop and demonstrate different, even contradictory, ideas about the world, and which ideas are operational at any moment is highly context dependent [31]. Attempts to generalize from data borne from collaborative interaction to claims about students' decontextualized and individualized knowledge are, therefore, neither valid nor practical. Instead, the CER community must adapt and develop theories and methods for studying students' groupwork as activities within which functional roles and conceptual development may be distributed across the group, and in which we may not be able to locate knowing in individuals alone, but in the group (and its setting) as a whole system.

The term *cognition* traditionally refers to the brain's mechanisms for information processing, error correction, memory, perception, and communication. However, research on situated, embodied, and social cognition challenges this model, highlighting ways in which real-world cognition cannot be easily located solely within the skull. Consider Lave's classic example of The Cottage Cheese Problem [18]: "Dieters were asked to prepare their lunch to meet specifications laid out by the observer. In this case, they were to fix a serving of cottage cheese, supposing that the amount allotted for the meal was three-quarters of the two-thirds cup the program allowed. The problem solver began the task muttering that he had taken a calculus course in college. Then after a pause he suddenly announced that he had 'got it!' He filled a measuring cup two-thirds full of cottage cheese, dumped it out on a cutting board, patted it into a circle, marked a cross on it, scooped away one quadrant, and served the rest."

An alternative method to solving this problem is to multiply $^2/_3$ by $^3/_4$ to find that the answer is $^1/_2$, and then to locate a suitable measuring cup. Were the dieter doing this work in a mathematics classroom where there is no cottage cheese, measuring cup, or cutting board, this would likely have been the preferred problem solving approach. But the physicality of the materials makes another solution approach ready-to-hand [14], one in which the cognition happens in the interaction between the dieter and the physical materials. The mathematical result is the same but the means — distributed across dieter, cheese, cup, and board, and no longer purely symbolic — is quite different. Theories of embodied and social cognition provide a framework within which we can analyze the multi-facetedness of the dieter's actual work. It isn't mental work alone, and any definition of cognition that ignores the situated, material, embodied character of the work is, therefore, necessarily ill-fitting to actual everyday cognition. Instead, to better "carve nature at its joints," (as Plato put it) we must bring setting, body, tools, and culture into our definition — and our empirical study — of cognition, treating all of these constituents of activity as just as crucial to cognitive accomplishment as what happens inside the head.

Distributed Cognition [16] (DCOG) theory generalizes this perspective on real-world cognition, arguing that all of the cognitive functions (e.g., memory) that have historically been analytically located in the head can also be seen in the emergent properties of interactions between people and tools in culturally- and historically-produced settings. In proposing a framework for studying "Cognition in the Wild," Distributed Cognition theorists highlight several key aspects of cognitive systems that researchers should attend to [15]:

- The distribution of cognitive processes across social groups, including ongoing redistribution of activity to balance cognitive load.
- That culture intimately shapes cognition by offering tools, settings, and social norms through which to work or through which work can break down.
- Effortful coordination between internal (to the head) and external (tools and environment) structure, including through the use of the body of the problem-solver.
- The event-driven path dependence of cognition, where "the products of earlier events can transform the nature of later events."

DCOG has many strengths over traditional approaches to understanding cognition. Perhaps most powerful is its ability to account how a diversity of tools and representations are key elements in socio-technical systems of problem-solving; specifically, that external tools and representations are not just objects used by people, but can do cognitive or social work, such as transforming one kind of information into another or prompting a group to talk about their knowledge in particular ways. DCOG challenges researchers to take careful account of how individuals interact with one another, tools, and setting, and to observe the ways in which those interactions produce the cognitive functions of information processing, memory, sensing, error correction, etc.

In this view, *learning* is no longer just change in individuals' conceptual models (a constructivist take on what learning is) or behavior (a behaviorist take), but also includes changes in the relationships between individuals and in their individual and joint relations to tools and settings, which can also be modified over time. For example, introducing a new tool (say, a slide rule or a computer) to an engineering team that previously had been required to do mental or paper-based calculations is a form of learning: the team becomes able to do existing work faster and possibly also becomes able to take on newer, harder problems. The group has learned, but that learning cannot be fully located in any of the individuals alone, nor in the individuals without their tools (who possibly might only be able demonstrate knowledge through the use of the tools). Rather, the learning consists in how their emergent system of interaction reconfigures itself to solve problems in new ways, such as by re-distributing cognitive load onto tools. Using DCOG in the analysis of learning can offer insights into how activity (including learning) happens that are not possible within the traditional cognition-is-inside-the-head paradigm. To study learning through the paradigm of classical cognition is necessarily to devalue the many nuanced ways in which real-world knowledge and real-world learning is intimately bound to social and material context. Instead, DCOG offers access to the social, material, cultural, embodied, and mental richness of activity and learning.

3. CASE STUDY

In order to illustrate the affordances of DCOG for analyzing collaborative CS learning, we now present a qualitative case study analysis of one group of youth programming a computer music system. As we shall show, DCOG theory enables us to richly analyze how that group works and learns, including understanding the specific student knowledge and tool design weaknesses that cause a breakdown within the group work to occur, and to carefully understand the influence of a teacher's intervention when the breakdown occurs.

3.1 Context

The example presented here is a few brief minutes of student group work that occurred within a computer music summer camp that we conducted in Summer 2014. We chose this example because of its richness as a case of students' prior knowledge and tool design decisions shaping learning, though many other comparably rich examples exist within our data set (described below). We present it here not so much as a characterization of what all students' participation in the summer camp was like, but, rather, to show how DCOG offers a framework to understand learning in collaborative CS group work.

The summer camp was a research vehicle for us to investigate how computer music can be a productive medium for engaging under-represented populations of students in parallel and distributed computing through designing, building, programming, and performing with tangible computer music instruments. We hosted the camp at a community center in a lower/mixed socio-economic area of a large Northeast U.S. city. The camp was two weeks (9 days) long, and met for 3 hours in the morning each day. Fourteen rising sixth- and seventh-grade students participated, of whom 3 were girls. Only 12 youth consented to participate in data collection. All students were African-American and/or Latino. All camp activities occurred in the music room of the community center, which was well-appointed with musical instruments (a piano, guitars, drums, and DJ equipment). At least four members of our research team were always present in the room and available to students for help on their projects.

Students in the camp used a prototype computer music tool called *BlockyTalky*. BlockyTalky runs on small physical computing devices, called BlockyTalky Units or BTUs. BTUs can be hooked up to various sensors, and they have holes to mate with LEGO bricks. This enables users to create their own prototype tangible computer music input devices. Each BTU runs a web server that hosts a graphical blocks-based programming interface for musical programming. The BlockyTalky language has music-specific blocks, such as a block to create rhythmic phrases and another to define melodic phrases. Melodic phrases are specified by stringing together blocks that take two parameters each: the first parameter is how long a note should play (e.g., a quarter note) and the second is what the pitch should be, specified as a note letter and the octave to play it in. For example, `C4` is the note middle-C, while `C5` is the C one octave above.

3.2 Data set

All camp sessions were recorded using video cameras placed across the space, creating about 215 hours of video data; to maximize audio quality every participant wore a wireless microphone. The cameras were positioned to capture maximum group activity around the table the group was working on. However, students sometimes knocked these cameras out of alignment. The audio/video data presented in this paper were collected early in the second day of the camp.

3.3 Prelude

Our camp curriculum dedicated the first week of the camp to enabling youth to learn their way around BlockyTalky by modifying and creating instruments. This was intended to prepare students for the second week's goal: musical performances using instruments of the students' own making.

During Day 1 of the camp, students clapped out and drew their own representations of rhythms before inputting their patterns into a web-based drum machine. The data we present here were collected early in Day 2 and revolve around a pair of participants.

Due to the novelty of our system and the students' limited computational and musical backgrounds, we began Day 2 by offering students an assortment of pre-made BlockyTalky instruments to play with and then modify. We expected students to begin creating their own instruments on Day 3, though some students had modified the pre-built projects beyond recognition by the end of Day 2! On Day 2 in particular, we hoped to see students begin to understand the idea of sequencing [17] as it cut across music and computation. In music, sequencing refers to defining a pattern of specific sounds to occur at specific times (e.g., defining a sequence of notes to create a melody). Computationally, we can understand this as a program that executes an ordered sequence of instructions, each separated by a specific length of time. In BlockyTalky, these two ideas of sequencing are complementary and both are needed to program a melody or rhythm.

Chris and Nathan began Day 2 by programming and playing music on a pre-built drum instrument for about 30 minutes before switching to a different pre-built unit that used a light sensor's detection of changing color values to trigger a melody. Soon after, the boys shifted focus to building a device to play a melody of their own choosing. The process through which they worked toward this goal is the focus of our case study. As we will show, the boys worked together with each other, BlockyTalky, a guitar, and a teacher's help as part of a continually adapting distributed cognition system.

3.4 Phase 1: Choosing what to program

In the first phase of the episode, a pair of students, Nathan and Chris, are the central actors in a distributed cognition system which makes a collective decision about what song to program. The system not only includes the pair of students, but also elements of the setting (guitars within easy reach), and a shared history of the boys' prior participation in a music camp at the community center together (including a repertoire of songs both boys know how to play on guitar). At the end of this phase, the system will have reached a decision about what to program.

This episode starts with Jake, an undergraduate facilitator from Tufts University, encouraging Chris and Nathan to modify the pre-built project. Jake then explains briefly how the existing code works.

[0:16] **Jake** Do you guys want to make your own song to play?
[0:19] **Nathan** Yeah
[0:19] **Jake** Alright, so have you guys played with these blocks at all yet?
[0:22] **Nathan** Not yet
[0:24] **Jake** Alright, let's stop. So basically what this does is ummm it basically just plays all these notes in order
[0:38] **Nathan** In order, okay
[0:40] **Jake** And then this block says on the next beat start looping so basically it just plays this string of notes over and over and over and you can choose

Jake then explains that every note in BlockyTalky needs a pitch and a duration. This will become relevant later when the system's demands for this information lead the boys to recognize a gap between their own knowledge and the system's requirements.

[0:54] **Jake** But basically you choose the length of how long you want the note to play here
[1:08] **Nathan** Ok
[1:08] **Jake** And then you choose the note itself right here

With this information in mind, the boys discuss how they want to modify the pre-built unit.

[1:24] **Nathan** Ok. I have an idea. Maybe if we look up the—
[1:28] **Chris** I know what we can do!
[1:29] **Nathan** We could look up the notes for a song
[1:31] **Jake** Right
[1:32] **Nathan** Yeah, let's look up the notes for a song
[1:32] **Chris** No no no we could do one of the songs we learned on guitar
[1:37] **Nathan** Smoke on the Water, yeah!
[1:44] **Nathan** *typing* smoke
[1:49] **Chris** We already know the notes for that
[1:52] **Nathan** We don't know how long to play them for

Nathan initially proposes ([1:24] and [1:29]) to look up the notes for a song. Chris makes a counter proposal — that they program a song they already know on the guitar ([1:32]). These seem like two mutually exclusive proposals but the boys do not see them that way. Nathan proposes a song that fits Chris's criteria of a song they know how to play on the guitar, *Smoke on the Water*, but then immediately proceeds to search for the notes on the computer. Chris protests this action by stating that they already know the notes for the song. Nathan follows up with the fact that they know the pitches of the notes, but they don't know the durations, essential details of programming notes in BlockyTalky that Jake mentioned earlier ([0:54]). This nuance reflects a difference in the kind of information that can be produced and processed by the different kinds of actors in the distributed cognition system at work here. While a beginning musician can know how to play a song in a rough sense (first put your fingers here, then put your fingers there...), these kinds of performances tend to get the sequence of pitches right but not the timing. It takes more guitar practice than these boys have had to be able to fluidly move between finger and hand positions at the proper tempo. In contrast, BlockyTalky demands more precise information, both pitch and timing. The boys have a way to produce the pitches but do not know the timing. As we see next, Nathan seems to view the mismatch between what they know and what BlockyTalky requires as problematic, while Chris does not. A negotiation now ensues about whether they need to know note durations or not. As this begins, Nathan retrieves a second computer to look up sheet music.

[3:23] **Chris** Nathan, we don't know how ... we don't have to know how long to play to — play it for
[3:28] **Nathan** I know, that's why we have to look it up though
[3:29] **Chris** No, we already know how long to play it
[3:35] **Nathan** We don't know exactly one sixth or one fourth of a note. Like, do you know that? Do you know how long — like do you see right here? how long to play it for. Like one sixteenth note of note C5, we don't know how long to play it for. We know the notes, yeah
[3:50] **Chris** We just play one note. We don't have to play part of the note. We can play the whole thing.

Chris still disbelieves that they need to know how long to play each note, but Nathan insists that they do. We interpret Chris's remark at [*3:50*] as a proposal that they not worry about duration of the notes and that they should just set them all to whole notes. Nathan does not acknowledge this proposal and continues searching on the computer for sheet music with Jake's help. He finds guitar tablature.

[4:24] **Jake** Alright, so that's a tab. That's for guitar. Do you want to play Smoke on the Water?
[4:33] **Nathan** Yeah
[4:35] **Jake** Umm, this one probably isn't going to help you much because it's like — it's guitar music so it doesn't really — this one is probably your best bet. So, it looks like — so they're chords
[4:51] **Nathan** This is — this is supposed to start at zero
[4:53] **Jake** What?
[4:55] **Nathan** It's supposed to start at zero. That's how you play it. I know how you play it on guitar but this is not how you play it.
[5:02] **Jake** Interesting
[5:03] **Nathan** You're supposed to start at zero.
[5:08] **Jake** It might just be in a different key.
[5:10] **Nathan** Like this one would help us. Oh, this is a trumpet's. This one would help us the most. Yeah, this one, 0033553.
[5:24] **Jake** So looks like it's going to be a D and an F and a G. Is this actually Smoke on the Water?
[5:37] **Nathan** Yeah. I can play it for you on the guitar.
[5:40] **Jake** But I mean — the rhythm of it
[5:43] **Nathan** Do you want me to play it for you?

This exchange began with Jake pointing out that what Nathan found is tablature, which specifies sequences of chords and so will not help much with the boys' quest for timing information. In response Nathan provides a piece of knowledge from his experience playing the song on the guitar, that it's supposed to start at zero (an open string), as a proposed criterion for judging their search results. When Nathan seems to find a result he is happy with, he points it out, saying and repeating "this one would help us." To make his point that it fits his criterion, he says a string of numbers starting with zero. Jake looks at the pitches indicated by the actual notes and wonders out loud if these notes actually correspond to the song they are trying to play. Nathan makes his first ([5:37]) and second ([5:43]) offer to play *Smoke on the Water* for Jake on one of the acoustic guitars in the music room. Jake seems skeptical this sheet music is accurate as he continues analyzing it, this time noting that the rhythm seems wrong. Jake then gets pulled away from this problem by a question from Chris.

[5:44] **Chris** Like, when it's putting the numbers next to the letter on the program, does that mean that it's putting the like what the note it would be on and the position it would be on?
[5:56] **Nathan** I could actually show you
[5:57] **Jake** What do you mean? Looking at this?
[5:59] **Chris** Yeah
[6:01] **Jake** Okay, so what was your question again?
[6:05] **Chris** so if — when it has the note and the number together
[6:13] **Jake** yes
[6:10] **Chris** Is that like the umm the note and the position it's going to be played in?
[6:17] **Jake** So it's going to be — it plays a .... it plays a
[6:26] **Nathan** I can actually play it if you want me to
[6:26] **Jake** Alright, one sec, I need to answer this question first
[6:28] **Nathan** Want to play Chris?
[6:31] **Jake** Umm, it plays the note B4 one sixteenth of a beat. And it's the first note that's played. And the next note that's played is a B flat 4 for one sixteenth of a beat. Does that make sense?
[6:57] **Nathan** [inaudible]
[6:58] **Chris** Yeah

Jake clarifies for Chris that one enters notes into BlockyTalky by specifying a pitch (denoted by a letter between A and G) and octave. Jake also reiterates his earlier description of the system's execution of musical programs as "it basically just plays all these notes in order" by walking through the example of a couple of notes in the pre-made code. Right after Chris says "Yeah" at the end of the episode ([6:58]), Nathan plays the intro to *Smoke on the Water* on the guitar he has retrieved from the wall.

Notably, from this point forward, the question of whether the boys need to know precise timings is dropped. BlockyTalky supplies default timing of a $^1/_4$ note. However, because each note has two formal parameters (timing and pitch), Nathan may have felt a premature need (relative to guitar-learning) to supply this information. This in turn led to a back and forth exchange between the boys and a search for information online. Ultimately, this formal, but optional, parameter does not impact the rest of the boys' interactions in the episode. This is probably because the exchange just described ends with a guitar in Nathan's hands and Chris in control of the laptop. Chris, who earlier ([3:50]) suggested assigning a uniform note duration to all notes, is in a position to use the system's default uniform timing while Nathan produces a sequence of pitches on the guitar.

3.5 Phase 2: Representational Transformation

Our narrative continues with Jake asking if Nathan knows the notes he just played, referring to the standard musical notation they need to input the notes into BlockyTalky. This event triggers a shift in cognitive system's function from memory recall to information transformation. The transformation process will eventually map each note of *Smoke on* *the Water* from Nathan's knowledge of how to play the song to a symbolic, formal representation that BlockyTalky can execute.

Crucially, Nathan's knowledge of the notes in the song is only externally visible in the moments that he is putting his fingers into position on the fretboard and plucking the strings. Outside of those moments, the knowledge is internal to him, but it is an embodied understanding that he might have a very hard time mentally reasoning about without use of his body and an instrument. So the moments that it is most accessible to him are also the moments that it is most accessible to others. That is, he does not know the names for these fingers positions and can only communicate his partial and non-symbolic knowledge of the notes by playing them with his hands around a guitar. In contrast, BlockyTalky requires a symbolic (note letter) description of what notes to play (e.g., `C4`); its interface does not allow input in the form of finger positions (i.e. there is no virtual instrument or way to hook in a real guitar as input into the program). Thus, as we shall see, there is a breakdown between the representation of knowledge that the boys can produce with a guitar and the representation that BlockyTalky needs; the boys and tools alone cannot create mappings between these two representational systems (see Figure 1).

[7:36] **Jake** Alright so, do you know what notes those are?
[7:45] **Nathan** [unclear] This is an E
[7:47] **Jake** An E? So this would be what — a D#?
[7:56] **Nathan** I don't know. I know —
[7:59] **Jake** This is an E and that's an F and that's an F# and that's a G. So it's an E, then a G and then an A.
[8:13] **Nathan** I don't know. Do you know someone that knows guitar?
[8:14] **Jake** I mean I — so each of these is a half note. So this would be — the lowest string is an E so then this would be a G so then this would be an A.
[8:27] **Nathan** Awesome. Okay.
[8:32] **Jake** So, we're going to want to do —

As we see above, even though Nathan can play the song and has an embodied knowledge of the notes of *Smoke on the Water*, and even though Chris knows how to put notes into the BlockyTalky system, they cannot proceed. Jake steps in to fill the gap between these two representational forms. This is a notable shift in the structure (see Figure 2) of the system; it was only directly preceding this phase of the episode that Chris started interacting with Jake at all (first interaction at [5:44]). Prior to this, Nathan had been talking to any adults that addressed the group, and no adults had been a part of the boys' problem solving.

As we now show, this new distributed cognition structure (see Figure 3) allows the system to accomplish its function of information transformation in the following way: Each note of the song is retrieved from Nathan's memory, and is externalized using a finger position on the guitar that Nathan holds. Jake then employs his musical knowledge to transform the finger position supplied by Nathan (expressed using his body and the guitar) into a letter note. Jake also fills in a duration and octave he deems an appropriate approximation of the song based on his musical background. Chris hears Jake say these details aloud, then enters 3 pieces of information for each note — duration, letter and octave — as formal parameters in his program code. This emergent distribution is due to the individuals' background knowledge (from prior to our camp), as well as the requirements of the BlockyTalky language. Of the three, only Nathan and Chris know how to play the song on the guitar, meaning that the students supplied the starting representation. The instructor, Jake, then had the task of helping them transform this representation to a more standard one, because of the three he was the only one with enough musical background to figure out the letter notes, octaves and approximate note duration. The new arrangement enables the distributed cognition work of recall from memory, information processing (transformation), and memory storage (as program code) for later playback by the program that Chris and Nathan are writing.

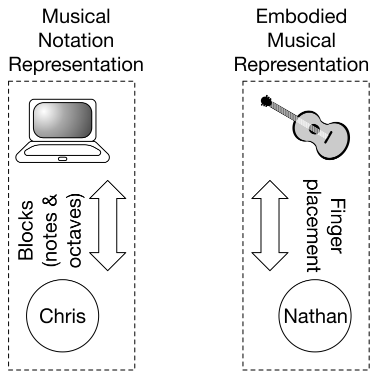
Musical Notation Representation
Embodied Musical Representation
Blocks (notes & octaves)
Finger placement
Chris
Nathan

Figure 1: Representational Breakdown

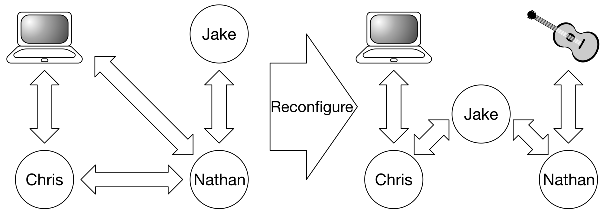

Figure 2: System Reconfiguration

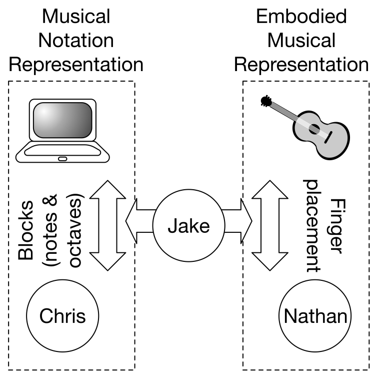
Musical Notation Representation
Embodied Musical Representation
Blocks (notes & octaves)
Jake
Finger placement
Chris
Nathan

Figure 3: Representational Bridge

[8:43] **Nathan** It's E, it starts out with an E. Chris do you want to start writing it down? It starts out with an E as a - and these are half notes, right?
[9:07] **Jake** Yeah
[9:07] **Nathan** Half notes.

[9:07] **Jake** They're probably half notes.
[9:10] **Nathan** And it starts out with an E
[9:11] **Jake** So do an E4. And then we're going to have another half note.
[9:19] **Nathan** Oh, I could ask him, he probably knows. Oh- okay okay okay.
[10:00] **Chris** So what would ummm, this one be? Would it be G#?
[10:13] **Jake** So that's going to be A#
[10:14] **Chris** A#
[10:17] **Jake** B flat
[10:21] **Jake** Yeah, the important thing to remember on a guitar is that every fret is a half note.
[10:27] **Nathan** Okay
[10:29] **Jake** So if you know that the bottom string you can just count up

Nathan, despite his prior claims that they should know the duration of each note ([3:35]), proposes that they set all the notes to have the duration of a half note ([8:43]). Chris starts trying to figure out how the transformation works by asking for confirmation on the next note ([10:00]). In response, Jake supplies the correct answer and a key piece of knowledge he is using for the transformation process ([10:21]). This could have been an opening for another reconfiguration, Jake stepping out of the system and Chris taking over his role. However, as we show, Chris does not attempt the transformation process on his own. The conversation begins to shift to Chris's part of the process — programming the notes into the BlockyTalky. Jake continues to help Nathan transform notes from his embodied representation to a more standard representation so Chris can program them, as well as intermittently providing support when Chris asks for it.

[10:31] **Chris** Would it be A#4 or A#3?
[10:35] **Jake** A#4. So the numbers start on C so like CDEFGAB are all the same number then the number goes up.
[11:07] **Jake** So right here *pointing to screen* what do you want this to sound like? So do you want to play Smoke on the Water for us again?
[11:13] **Nathan** Okay. Wait. Wait...
[11:22] **Jake** You're missing one
[11:24] **Nathan** Oh yeah
[11:29] **Jake** So we go an E, we go a G. We go an E, we go an G -
[11:38] **Nathan** Then we go up: zero, this one, this one and then we go here. Yeah, 0, 3 5, 0, 3 6 5
[11:41] **Jake** E G A and then .....
[11:46] **Jake** Right, okay, so right now, there we go.
[11:53] **Chris** Ummm, where do you enter [inaudible]
[11:54] **Jake** Where do you- oh, so you want to add more notes you have to drag new notes in there. So you drag a note slot and you just drag it into the bottom there and then that adds a new note slot. And if you want to delete a note slot, you just go there and you pull it out.
[12:46] **Chris** What happened?
[13:10] **Jake** Alright, so where is our new thing?
[13:12] **Chris** I don't know where the other ones went. I put the notes in there but they disappeared.
[13:19] **Jake** What happened?
[13:19] **Chris** They disappeared
[13:23] **Jake** Hmmmm. So what were you.. Oh oh oh, so okay. Were you trying to- are you trying to add more these into those?
[13:35] **Chris** Yeah
[13:37] **Jake** Alright, so the easiest way to do it is to just - you can just go into music and the thing is right here
[13:45] **Chris** Oh, okay
[13:48] **Jake** So it's just right there in the music
[13:48] **Chris** Wait, umm, how do I take away one?
[13:51] **Jake** Take away one?
[13:53] **Chris** Can I just pull out one?
[13:54] **Jake** Yes but you have to pull out the correct one. So you want to go down to the bottom and pull out the last one.
[14:00] **Nathan** Oh, did you get it? Do you need help?
[15:57] **Nathan** We gotta stop, we gotta stop for a minute. Wait wait. Put it on blue now
[16:06] **Jake** So you know that putting it on blue color will make your new song play.
[16:24] **Nathan** Shouldn't it go a little faster?
[16:24] **Jake** Alright, so we have the right notes but they aren't all the right lengths, you know.

The episode's end is marked by the boys expressing their next goal: fixing the timing. Nathan expresses dissatisfaction ([16:24]) with the speed of the song overall, and then Jake points out that the duration of the notes are not all half notes so they need to be changed. It is important to note at this point that the students' familiarity with the song gives them a resource for constructing a feedback loop with BlockyTalky as they proceed to address this weakness. When running the program or testing their code, the students can hear whether the song sounds correct — in the above case they are noting that the timing is not the same in their code as compared to the actual song that they remember. This feedback loop enables students to critique their own projects, enabling them to capture the sound of the original song by iterating over their code.

4. DISCUSSION AND CONCLUSIONS

Nathan and Chris used their musical and computational knowledge to customize a pre-built computer music project to meet their own goals. Through a DCOG process of recall, representation, and data transformation, they programmed a melody with BlockyTalky. At the beginning of the episode, it was clear that Nathan could play the song and had an embodied, non-symbolic representation of it. Chris demonstrated he was capable of putting standard music notation into computer code. However, they were only successful because the group reconfigured social structure to include the impromptu support provided by Jake and his extensive musical background. Despite the representation transformation roadblock they encountered, the students successfully programmed a sequence of notes that played one after another by programming a sequence of blocks that was based upon a sequence of notes plucked on a guitar.

This episode illustrates how the DCOG offers insights into how thinking and learning are happening that are not possible with a traditional theory of cognition. Cognition in this episode was distributed across the group. Nathan retrieved from his long-term memory a representation of what the notes in the song were. The format of this representation was not usable within the BlockyTalky software, which led Nathan and Chris to detect that they were at an impasse and then to initiate a reconfiguration of the group. They roped in Jake, who assisted with crucial information processing work, which Chris then encoded in the computer's memory (in the form of program code). The work was deeply shaped by cultural resources: the students and Jake all knew what *Smoke on the Water* should sound like because it is a mainstay of classic rock. This provided a common ground in which to problem solve, as well as motivation to the boys. The cultural artifact of the guitar, available because of the setting of the camp in a music room, was instrumental in Nathan's ability to articulate what the notes of the song should be; together with his body (his fingers, in particular) it became the mechanism through which Nathan's knowledge went from an internal (to the head) resource to a shared resource for problem solving. The event of Nathan's retrieval of a guitar sent the group down a particular problem solving path; had they continued searching online they may have found other ways to find the notes of the song, such as by finding ways to transform guitar tablature to standard musical note names or by simply finding a list of notes to directly input into the computer. Finally, the events that took place after the episode detailed here, tweaking the note durations to comport with the boy's memories of the song, reflects the computer's instrumentality to the boys' cognition: they are not skilled enough musicians to reverse-engineer timing from their own intuitive senses of the song, and so the artifact of the computer becomes a cognitive resource that enables problem solving through easy trial-end-error experimentation with note lengths. With more musical practice this system could be rebalanced to not need this technological affordance, but until then, the computer is a vital part of the boys' musical cognition.

The DCOG analysis also enables us to ask nuanced questions about assessment: Was the boys' dependence upon help from Jake problematic? Was their getting stuck a reflection of difficulties with understanding programming, or something else? They could not have achieved their goal without Jake doing a key aspect of the representation transformation with them, but the help he provided was not primarily about computing. This suggests that if we want students to be able to explore computer science and engineering through media like music, then our learning environment designs (including tools, curriculum, and teaching) need to allow students to succeed even if they do not yet possess virtuosity in the media that they are computing with. To wit: Music is a very powerful cultural resource, and one that could be quite useful for supporting learning in computing. But students should not need to know how to read someone else's hands on a guitar in order to learn to program a computer. This suggests a design problem: how should we provide support for computer music that facilitates progress without requiring such extensive music disciplinary knowledge? For example, should we change our system to support plugging a guitar into the computer so that simply playing it creates a program? That would certainly circumvent the need to know standard music notation, but it would also obviate most of the project's programming in the process. In our current design, even with Jake's help, Chris still modified the existing program, first changing existing note blocks and then adding new ones as the sequence of instructions lengthened to match Nathan's guitar plucking. Going directly from a guitar to program code would skip this programming work, a prospect that we find undesirable. In future work we will explore intermediate representations that are within students' zones of proximal development [33].

Jake's preservation of students' agency raises exciting questions for curriculum design: He was able to scaffold the activity for the boys, becoming part of the system while letting the boys do the computer programming entirely themselves. Agency in the group remained with the students. In our experiences, it is often the case that when a teacher assists a group with a hard problem, agency over the group's problem solving shifts to the teacher. That was not so here. This may mean that the interdisciplinary nature of the project, combined with the teacher demonstrating authority in the musical content but not overtly so in the computer programming aspects of the project, enabled a less agency-stripping enactment of the role of expert. This leads us to the question: Can future media computation curricula deliberately include productive participatory roles for teachers where they can play an expert role in a non-computing domain (like music) to afford them close observation of students' emerging computation without interfering with the computing agency and skill that students are developing?

New programmers sometimes treat computers as if they were intelligent partners, as machines that can ask for clarification [24]. This conceptual bug is a source of errors in learning. Yet computer interfaces, including programming languages, frequently demand information from their users. BlockyTalky's prompting for note durations led the boys to argue about the amount of information about each note that they would need to know to be productive. This set of discursive events was spurred by students' differing interpretations of what clarification the system needed in order to be programmed. In a sense, the computer's interface imbued it with agency to guide the group's discussion. Though the timing discussion in this case was ultimately inconsequential, computing education researchers might investigate ways that future tools could use this potential for agency to guide group problem solving discussion.

We have shown how DCOG can be a powerful theoretical framework for analyzing collaborative CS learning. But its utility is not limited to studying multi-student groupings, as most of the properties of cognitive systems that DCOG highlights also inhere in single-student learning (which is also culturally- and historically-situated, involves external and internal representations, tools, etc.). A limitation of this approach is that detailed discourse analysis of the sort we performed here is time consuming and therefore difficult to scale. However, as DCOG opens up possibilities for understanding student learning in CS that other perspectives and methodologies lack, it could become a powerful tool for improving research on computing education.

5. ACKNOWLEDGMENTS

We thank the National Science Foundation for funding this work (Award 1418463), and especially appreciate Jeff Forbes's advocacy within NSF for this EAGER project. We also thank Theresa Perry for her advice and assistance, and Nick Benson, Ben Helm, Case Jemison, Michael Ferdico, & Anthony Ambroso for their help in collecting these data.

6. REFERENCES

[1] D. Abrahamson and U. Wilensky. The stratified learning zone: Examining collaborative-learning

design in demographically-diverse mathematics classrooms. In *Annual Meeting of the American Educational Research Association*, 2005.

[2] J. R. Anderson, L. M. Reder, and H. A. Simon. Situated learning and education. *Educational Researcher*, 25(4):5–11, 1996.

[3] A. Basawapatna, K. H. Koh, A. Repenning, D. C. Webb, and K. S. Marshall. Recognizing computational thinking patterns. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 245–250, 2011.

[4] M. Berland, C. Smith, and D. Davis. Visualizing live collaboration in the classroom with AMOEBA. In *Proceedings of the Tenth International Conference on Computer-Supported Collaborative Learning*, 2013.

[5] P. Blikstein, M. Worsley, C. Piech, M. Sahami, S. Cooper, and D. Koller. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4):561–599, 2014.

[6] J. D. Bransford, A. L. Brown, and R. R. Cocking. *How people learn: Brain, mind, experience, and school.* National Academy Press, 1999.

[7] K. Brennan and M. Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, 2012.

[8] A. L. Brown. Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The Journal of the Learning Sciences*, 2(2):141–178, 1992.

[9] N. C. Brown and A. Altadmri. Investigating novice programming mistakes: Educator beliefs vs. student data. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, pages 43–50, 2014.

[10] A. S. Bruckman. *MOOSE Crossing: Construction, community, and learning in a networked virtual world for kids.* PhD thesis, Massachusetts Institute of Technology, 1997.

[11] M. Cole. *Cultural psychology: A once and future discipline.* Harvard University Press, 1998.

[12] A. Collins, J. Brown, and S. Newinan. Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics. In *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, pages 453–494. Lawrence Erlbaum Associates, Inc., 1989.

[13] D. A. Fields, V. Vasudevan, and Y. B. Kafai. The programmers' collective: Connecting collaboration and computation in a high school scratch mashup coding workshop. In *Learning and Becoming in Practice: ICLS 2014 Conference Proceedings*, 2014.

[14] M. Heidegger. *Being and time: A translation of Sein und Zeit.* SUNY Press, 1996.

[15] J. Hollan, E. Hutchins, and D. Kirsh. Distributed cognition: Toward a new foundation for human-computer interaction research. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(2):174–196, 2000.

[16] E. Hutchins. *Cognition in the Wild.* MIT press, 1995.

[17] E. R. Kazakoff, A. Sullivan, and M. U. Bers. The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal*, 41(4):245–255, 2013.

[18] J. Lave. *Cognition in practice: Mind, mathematics and culture in everyday life.* Cambridge University Press, 1988.

[19] S. T. Levy and U. Wilensky. Inventing a "mid level" to make ends meet: Reasoning between the levels of complexity. *Cognition and Instruction*, 26(1):1–47, 2008.

[20] C. M. Lewis. *Applications of Out-of-domain Knowledge in Students' Reasoning About Computer Program State.* PhD thesis, Berkeley, CA, USA, 2012. AAI3555787.

[21] L. Malmi, J. Sheard, R. Bednarik, J. Helminen, P. Kinnunen, A. Korhonen, N. Myller, J. Sorva, A. Taherkhani, et al. Theoretical underpinnings of computing education research: What is the evidence? In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, pages 27–34, 2014.

[22] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald. Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 49(8):90–95, 2006.

[23] M. Packer. The problem of transfer, and the sociocultural critique of schooling. *The Journal of the Learning Sciences*, 10(4):493–514, 2001.

[24] R. D. Pea, E. Soloway, and J. C. Spohrer. The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics*, 9:5–30, 1987.

[25] L. Porter, C. Bailey Lee, and B. Simon. Halving fail rates using peer instruction: A study of four computer science courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pages 177–182, 2013.

[26] L. Porter, C. Bailey Lee, B. Simon, and D. Zingaro. Peer instruction: Do students really learn from peer discussion in computing? In *Proceedings of the Seventh International Workshop on Computing Education Research*, pages 45–52, 2011.

[27] B. Rogoff. *The cultural nature of human development.* Oxford University Press, 2003.

[28] J. Roschelle. Learning by collaborating: Convergent conceptual change. *The Journal of the Learning Sciences*, 2(3):235–276, 1992.

[29] N. Shah, C. M. Lewis, and R. Caires. Analyzing equity in collaborative learning situations: A comparative case study in elementary computer science. In *Proceedings of the 11th International Conference of the Learning Sciences*, 2014.

[30] B. Sherin, A. A. diSessa, and D. Hammer. Dynaturtle revisited: Learning physics through collaborative design of a computer model. *Interactive Learning Environments*, 3(2):91–118, 1993.

[31] J. P. Smith III, A. A. Disessa, and J. Roschelle. Misconceptions reconceived: A constructivist analysis of knowledge in transition. *The Journal of the Learning Sciences*, 3(2):115–163, 1994.

[32] J. Tenenberg and Y. B.-D. Kolikant. Computer programs, dialogicality, and intentionality. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, pages 99–106. ACM, 2014.

[33] L. S. Vygotsky. *Mind in society: The development of higher psychological processes.* Harvard University Press, 1980.

[34] T. White. Code talk: Student discourse and participation with networked handhelds. *International Journal of Computer-Supported Collaborative Learning*, 1(3):359–382, 2006.