The technical matters: young children debugging (with) tangible coding toys

Deborah Silvis
Department of Instructional Technology and Learning Sciences, Utah State University, Logan, Utah, USA

Victor R. Lee
Graduate School of Education, Stanford University, Palo Alto, California, USA

Jody Clarke-Midura
Department of Instructional Technology and Learning Sciences, Utah State University, Logan, Utah, USA, and

Jessica F. Shumway
Department of Teacher Education and Leadership, Utah State University, Logan, Utah, USA

Received 10 December 2021
Revised 7 April 2022
24 June 2022
Accepted 27 June 2022

Abstract

Purpose – Much remains unknown about how young children orient to computational objects and how we as learning scientists can orient to young children as computational thinkers. While some research exists on how children learn programming, very little has been written about how they learn the technical skills needed to operate technologies or to fix breakdowns that occur in the code or the machine. The purpose of this study is to explore how children perform technical knowledge in tangible programming environments.

Design/methodology/approach – The current study examines the organization of young children's technical knowledge in the context of a design-based study of Kindergarteners learning to code using robot coding toys, where groups of children collaboratively debugged programs. The authors conducted iterative rounds of qualitative coding of video recordings in kindergarten classrooms and interaction analysis of children using coding robots.

Findings – The authors found that as children repaired bugs at the level of the program and at the level of the physical apparatus, they were performing essential technical knowledge; the authors focus on how demonstrating technical knowledge was organized pedagogically and collectively achieved.

Originality/value – Drawing broadly from studies of the social organization of technical work in professional settings, we argue that technical knowledge is easy to overlook but essential for learning to repair programs. The authors suggest how tangible programming environments represent pedagogically important contexts for dis-embedding young children's essential technical knowledge from the more abstract knowledge of programming.

Keywords Early childhood, Interaction analysis, Computer literacy, Technical knowledge, Debugging, Sociotechnical organization of learning, Tangible programming

Paper type Research paper

The authors thanks to the principals, Kindergarten teachers, and students who welcomed authors into their classrooms.

Funding: This project has been funded by National Science Foundation grant #1842116; and a Utah State University Research Catalyst grant.

Information and Learning Sciences

2398-5348
DOI 10.1108/ILS-12-2021-0109

Introduction

When we conjure up an image of a computer scientist, the young child is not the default image that comes to mind. Yet early childhood education is fast becoming a site for teaching and learning valued skills like computational thinking (CT), programming and computational literacy. Children as young as preschool-age are engaging in coding with apps like Scratch Junior or with robot coding toys like Kibo (Bers, 2018). Researchers are now actively working to understand how children orient to computational objects and how we ourselves orient to young children as computational thinkers. Thus far, the existing research has tended to focus on how young children learn early computer science (CS) skills (Clarke-Midura *et al.*, 2021; Bers, 2018; Wang *et al.*, 2020). Less attention has been paid to how they learn basic *technical* skills, for example, how to operate a robot's controls or how to manipulate the buttons in an app. The tendency to privilege the work of *programming* over *operating* technologies is incomplete for learning to use tangible coding toys. In this paper, we develop a perspective on learning to debug code with tangible tools that accounts for how programming is also a technical matter and explore how children were performing essential technical knowledge as they learned to program.

While we focus on young children, we note that a programmer-technician hierarchy is common in computing fields. Work in software engineering and computer programming (e.g. writing code) is treated as having higher status than information technology support (e.g. assembling, connecting or operating components; installing software). However, the increased prevalence of tangible and physical computing devices has the potential to disrupt this order Horn (2018). Nonetheless, in research on learning, longstanding distinctions between *computer literacy* (e.g. inserting a CD, plugging in and turning on a machine) and *computational literacy* (e.g. building code and debugging programs) perpetuate the belief that technical knowledge is not on par with its programming counterpart (diSessa, 2000). From this perspective, technical know-how and hands-on interactions with the computer interface have not been valued in the same way as learning to program. This may be particularly true for novice coders like young children, for who operating computers' components is no simple matter or when tangible interfaces entangle hardware with software.

The computational object has traditionally been represented by the code, the screen and the algorithm; with tangible coding toys, these are now physically instantiated in blocks and 3D "sprites" you can hold in the palm of your hand Horn (2018). Once prohibitively expensive and almost exclusively used in lab settings (McNerney, 2004), commercially available programming toys now multiply the number and types of objects to manipulate and "think with" (Papert, 1980) as children learn to code. A wide variety of tangible coding toys are being used in early learning environments, and most kits include a robot (i.e. the agent), a set of directional arrows (i.e. the codes) and a controller or programming interface (i.e. remote-control mechanism) (Clarke-Midura *et al.*, 2019; Yu and Roque, 2019). Whereas learning to think computationally once involved arranging commands on a screen, where the agent then moved in 2D, tangible programming environments incorporate multiple materials and the mobile agent moves through the classroom. These emerging environments complicate coding in a number of ways that bear on this study.

First, preliterate children are still learning conventions for mapping correspondence between actions and symbols (Silvis *et al.*, 2020), resulting in frequent bugs. Second, with robot coding toys, bugs are not "contained" in a screen or software; they are distributed across tangible materials and hardware. Whereas in screen-based programming, broken code crashes the program, with tangible materials, breakdowns are also mechanical or physical and machines can literally crash into things in the physical environment. Children learning to manipulate code are also learning to wrangle coding *materials*, with consequences for debugging their programs. For young children who are novice

programmers, the relationship between breakdowns in code and in tangible coding machinery may not be transparent.

We examined this related set of issues in Kindergarten classrooms where groups of children engaged with tangible coding toys as they learned to build and debug programs. We began by asking broadly what was involved in children's debugging with robot coding toys and found that treating bugs as located in the *program* was inadequate (Silvis *et al.*, 2021). For example, having to enter the code into the toy via remote-control leaves room for human error, such as hitting the wrong button or forgetting to delete the previous program. In this analysis, we explore a perspective that debugging is more than a matter of identifying, locating and resolving errors in programs; it also involves reconciling coding errors with concurrent material mishaps, which we call "physical bugs." This understanding of debugging as also physical and mechanical prompted the questions that we address in the current analysis:

Q1. How is technical work involved in resolving programming breakdowns at the level of code and mechanical breakdowns in tangible toys?

Q2. How do children learn about technical aspects of operating coding robots while debugging (with) them (i.e. how is technical knowledge organized pedagogically)?

Our analysis positions children's technical work as central to debugging and accounts for how distributed technical knowledge mattered for repairing both types of breakdowns. We found that the technical skills children developed were more than a matter of fixing simple mistakes children made using robot coding toys; we also observed that their physical errors were independent of the design and engineering of the toys themselves. In other words, the technical skills required to resolve physical bugs are neither simply reducible to user error nor the result of products' design flaws. Since the program is embedded in and distributed across these tangible programming materials, in addition to bugs in the actual code, physical bugs interacted with programming bugs via distributed materials. The forms of physical debugging we analyzed were intertwined with debugging code and reflected a distinct form of knowledge characteristic of technical work more broadly. To get there, we present a series of cases of children learning to reconcile the physical and programming domains of this particular computational environment and the pedagogical moves that supported their technical knowledge.

In what follows, we first situate the present analysis of the technical work of debugging within related literature on early childhood tangible programming and educational robotics. Then we describe the larger study, a design-based research project designed to develop curriculum and assessment materials for early childhood CT. Next, we detail our analytic process that led to an understanding of the essential technical knowledge of debugging. We then present our findings narratively through microanalysis of three cases where children simultaneously grappled with bugs in the program and bugs in the physical apparatus. These cases position children's technical knowledge as instrumental for successful programming. We discuss how essential technical knowledge often occurs alongside programming but tends to be overlooked or subsumed within it. We argue that making the technical work of programming a distinct pedagogical focus is important in a society where there are ever more technical jobs for today's learners.

Framing perspectives

Tangible computing in early childhood

While the proliferation of coding robots in early childhood settings may suggest they are a recent invention, a consequence of the rise of early childhood computing standards (Bers and Sullivan, 2019; K-12 Computer Science Standards), tangible computing and robotics actually have a long history in early childhood. Research on tangible programming languages traces back to Papert's (1980) original Logo floor turtles, "objects to think with" that merged programming on-screen with a moving 3D robotic agent who drew shapes using a stylus (McNerney, 2004). A veritable menagerie of tangible designs and working prototypes emerged from this long line of research, including Lego bricks (Souza *et al.*, 2018), tangible crickets (Resnick *et al.*, 2000) and a robot named Kibo (Bers, 2018). For at least 20 years, studies of tangible programming in early childhood settings have examined how coding toys and robots support preliterate children to think computationally as they build and tinker, create and play and write and re-write programs (Bers, 2018; Bers *et al.*, 2014; Bers and Horn, 2010; Wyeth, 2008; Wang and Choi, 2021).

One version of the history of tangible computing reflects the field's developing interest in teaching CT skills like abstraction, sequencing, decomposition or debugging (Bers *et al.*, 2014; Brennan and Resnick, 2012; Sullivan and Bers, 2016; Wang *et al.*, 2020). Like block programming, tangible programming languages simplify and structure code so that young children can access essential principles of computational systems. But unlike in programming environments like Scratch or 3D virtual robotic simulations (Witherspoon and Schunn, 2019), learning to build and debug programs using tangible robot coding toys is a hands-on affair. We tend to colloquially construe code as "immaterial," a series of symbols operating "inside" the machine (Kitchin and Dodge, 2011); however, tangible programs actually take material form in a set of moveable, manipulable cards or blocks that represent a series of movements a robot makes. In other words, the robot is tangible, and so are the codes (Schweikardt and Gross, 2008). With a few exceptions (Sullivan and Bers, 2016), studies of coding robots and early childhood CT have tended to favor teaching *programming* over the mechanics of physical components of the computational system.

The second version of coding robot history emphasizes how they serve as the basis for early engineering education. Some of the same toys that are used to teach programming are used in educational robotics (Sullivan and Bers, 2016; Sullivan *et al.*, 2015), as young children learn how to engineer robots from a collection of functional components (Hamner *et al.*, 2010; Kim *et al.*, 2018; Socratous and Ioannou, 2020). While the coding robots we use in this study are pre-fab, commercially available and equipped with ready-made kits, the physical nature of the robots' components – and the consequences on the program of mechanical breakdowns – reflects the technical emphasis of educational robotics. Still, extant studies of young children's debugging in early engineering environments like LEGO robotics often treat problems of a *physical* nature (e.g. connecting a sensor to the wrong port) and *programming* breakdowns (e.g. omitting a command) as if they rely on the same sort of knowledge (Socratous and Ioannou, 2020). Rather than subsuming knowledge of the former within the latter, the current study attempts to understand the technical work of resolving physical errors on its own terms.

Debugging as technical knowledge

We build on a vibrant strand of learning research focused on debugging as a key dimension of programming and important for CT and CS education (DeLiema *et al.*, 2019; Fields *et al.*, 2021; Kafai *et al.*, 2019; McCauley *et al.*, 2008; Pea *et al.*, 1987). Research in this area is just beginning to identify the knowledge, skills and abilities that *young children* use when

debugging programs (Bers, 2018; Heikkilä and Mannila, 2018; Rich *et al.*, 2019; Wang *et al.*, 2020). For example, Bers and colleagues' (2014) model for young children's debugging included skills or steps for:

- recognizing errors;
- modifying goals;
- hypothesizing sources of error; and
- solving the problem in the context of using robot coding toys.

We adopt a perspective that debugging – which we take as the iterative process of creating, recognizing, locating and fixing errors – pertains to errors in programs *or machinery*. By examining breakdowns in the machinery as part and parcel of breakdowns in the program, we focus on how debugging tangible programs is largely a technical matter.

Our focus on not only code but also machinery or materials as a source of bugs is not entirely new to research on learning. Fields *et al.* (2016, 2021) designed for debugging through an e-textiles project in which fixing errors in the code (i.e. missing semi-colons) and fixing faulty circuitry (i.e. crossed wires) were both necessary to build a working product. This intricate relationship between errors in a program and problems with a physical part of a designed object is a widely recognized tension in engineering domains. For example, for machine operators responsible for successful IT demonstrations, the disastrous, ever-to-be-avoided "crash" refers both to broken code crashing the software *and* misguided machines literally crashing into something or physical parts breaking (Smith, 2009). What we analyzed as *technical knowledge* taps the latter set of problems of a material nature but explores them in a novel context where young children learned to code and operate tangible coding toys.

In previous work exploring how learning to debug involves understanding a relationship between physical mistakes and programming errors, the technical work is typically performed by individual children or children working in pairs. These learning designs have examined how older children resolve these two types of bugs (Searle *et al.*, 2018). In the debugging by design (DbD) projects (Fields, 2016, 2021), students embedded bugs in their e-textiles design projects for each other to solve. Of relevance to our study, students were required to create bugs in the program *and* bugs in physical materials, for example, creating buggy projects that included both a misspelled variable name and a light with reversed polarity. This work challenges learning designers to consider what DbD involves in other tangible computing contexts, for example, in a computational environment where debugging (with) coding robots involved learning both the programming language and the technical skills to operate the equipment.

In Kindergarten coding as well as professional practice, technical knowledge is regularly interleaved with programming knowledge across an arc of work. Technicians, *people who do practical work to make things or make things work* (Whalley and Barley, 1997), are involved in practically every occupation. In recent decades, as technologies are digitized and automated, society has organized more and more technical work, whereas the nature of technical work has shifted (Barley and Orr, 1997). Many machine operators no longer work on the factory floor, and their work is increasingly analytical. At the same time, technical functions like repair, maintenance, machine operating, troubleshooting, software support and routine data entry have been "bundled" into jobs that ostensibly require less training and skill. Despite this hiving-off process that has accompanied the specialization of all sorts of scientific and technical roles, practical work takes intellectual skills.

While technologies have changed, the tendency to subsume technical knowledge within more abstract programming knowledge is familiar. The history of scientific and technological breakthroughs is often heralded as one of the white men in white coats, singular discovery and sudden sparks of genius. This has been especially true in CS and engineering. Technicians, though highly skilled and critical for the operation of complex sociotechnical systems, have gone largely unrecognized, overshadowed by the towering figure of the research scientist or inventor in historical accounts. Positioned behind the curtain or in the "backrooms of science" (Barley and Bechky, 1994), people with technical knowledge were "hidden figures" (Shetterly, 2016) who were the mechanical backbone performing essential technical work (Shapin, 1989). In what follows, we recognize children's essential technical knowledge as consequential for debugging when programs are tangible.

Study design and analytic methods

Studying debugging as part of early childhood computational thinking

This study was part of a larger design-based research project focused on operationalizing and then developing classroom assessments and curriculum for early childhood CT (Clarke-Midura *et al.*, 2019, 2021b; Shumway *et al.*, 2021; Silvis *et al.*, 2020) and contributes to a growing body of work on the topic (Bers *et al.*, 2014; Bers, 2018; Bers and Sullivan, 2019; Tang *et al.*, 2020). While there is a general lack of agreement about the CT construct, how to teach or assess it and its value(s), debugging is one of a number of consensus skills widely thought to be relevant to solving problems computationally (Rich *et al.*, 2019; Wang and Choi, 2020). In the broader study, we focus on debugging as one of a suite of subcomponents involved in CT, iteratively:

- operationalizing it;
- teaching it; and
- assessing it.

The design of the curriculum tasks we describe in this analysis was part of the second objective of the project, informed by the other two across curriculum implementations.

Participants and groups

The data analyzed were collected over the course of a school year in two Kindergarten classrooms in two rural schools in the intermountain west of the USA. Thirty-two students participated in the project (female = 11). The two classrooms were part of a grant-funded program that provided full-day kindergarten to support English Language Learners, students who had an individualized education plan and students identified by their entrance assessment. All other students in the district attended half-day kindergarten.

Based on what we had learned about collaborative coding from the implementation of robot coding tasks in a previous pilot phase, we organized students into small groups of three to five children. Participating kindergarten teachers developed the groupings based on our recommendations that they put students together who "work well together." A teaching team of four researchers with elementary or early childhood teaching experience led these activities. Students participated in six coding lessons over a three- to four-week period. Each lesson lasted 30 min, and on any given day, two separate groups were coding robots in different areas of the classroom.

Coding robots and task roles

After studying the affordances and constraints of a number of commercially available options, we invested in tangible, screen-free coding toys (Clarke-Midura *et al.*, 2019; see also

Yu and Roque, 2019). Screen-free toys were the overriding preference of the teachers who were our research partners. We used three different robots (Figure 1), though the majority of lessons were taught using Botley and Cubetto. Without going into great detail about the design of specific curricular tasks, we provide a brief sketch of coding tasks and associated roles to contextualize the nature of bugs and analysis of debugging that follows. Most of the tasks with these toys were centered on the problem of getting the robot from one location to another. To do this, children used physical tiles or physical buttons to sequence a series of directional codes that instructs the robot to move in a specified way on a large floor grid.

To successfully program and debug with robots, children need to coordinate their self-referenced spatial orientation and movement, the robot's orientation and movement and directional symbols that represent the robot's movements in space (Silvis *et al.*, 2021; Clarke-Midura *et al.*, 2020). They must also coordinate the use of multiple coding materials, which are distributed and rotated around the group, multiplying occasions for mishaps. The ongoing redistribution of materials lent to a system of roles, as the nature of materials can determine how roles are designed and labor distributed (Stevens, 2000).

These roles established duties as well as undefined functions that emerged during activities. Two teacher-researchers regularly assigned these roles to students and rotated the role among the students, while two teacher-researchers typically allowed these roles to emerge more organically. In groups where roles were more emergent, rather than naming and designating roles and rotating all students through them, teachers allowed students to navigate in and out of roles as the task progressed. In all groups, whether officially named or de facto, the *Programmer* would design the program, specifying directional arrow codes one-by-one to an *Assistant Programmer*, who placed directional arrows in a specified sequence. Once a program or code segment was designed, the *Technician* or robot controller would press the corresponding buttons on the robot or operate the robot's programming board or remote control. If a group had more than three students, there was also an *Evaluator* who assessed the viability of the program prior to executing it by giving a thumbs up or down and justifying their assessment.

For example, the group of four students in Figure 2 worked together with their teacher Mr K to program Botley to travel along a short path according to the requisite commands

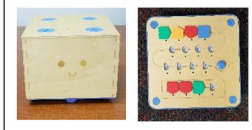
	BeeBot	Cubetto	Botley
Robot and controller			
Controlled by	Pressing arrow buttons on top of robot	Inserting code tiles into program board	pressing buttons on remote controller
Sample Program (FORWARD, LEFT, BACKWARD, RIGHT)			
Website	https://www.terrapinlogo.com/products/robots.html	https://www.primotoys.com/cubetto/	https://www.learningresources.com/shop/collections/botley

Figure 1.
Robots, controllers and directional codes for three kits used in the study

Figure 2. Four students programming Botley to travel forward, right, forword

Notes: Left to right: Benjamin, Franny, Kenneth, Caylie, Mr. K

FORWARD, RIGHT, FORWARD. In this task, Caylie was the *Programmer*, Kenneth was the *Assistant Programmer*, Franny was the *Technician* and Benjamin was the *Evaluator*. Caylie instructed Kenneth to place three arrows on the program organizer (the long white strip of paper, divided into segments for individual codes) indicating FORWARD, LEFT, FORWARD. Before running the program, Benjamin assessed that the program was buggy, giving it a thumbs down. Franny subsequently entered these three codes using the buttons on Botley's remote controller but inadvertently added a spurious FORWARD to the end of the program; thus, there was both a programming error *and* an error in operating the remote. Sure enough, when enacted, students could see this program was incorrect. How groups of students like this ultimately debugged such programs – and the consequentiality of Franny's technical work – is the subject of the following analysis (Table 1).

Table 1. Group member task roles, defined duties and other functions

	Programmer	Assistant Programmer	Technician	Evaluator
Defined duties	*Determine path Design program Point to code cards/tiles*	*Verifies selected codes Place codes/tiles in sequence*	*Enters codes into controller Presses Go button Clears program*	*Assesses program viability Gives thumbs up/down*
Other functions		*Suggests alternative programs/codes*	*Handles robot*	*Suggests alternative programs/codes*

It is worth noting that we designed for coding task roles in a context where students were very familiar with such an organizational structure (DeLiema *et al.*, 2020). In kindergarten classrooms, students often vie for particular roles and responsibilities like "door holder," "line leader," "snack person" or "plant and pet caretaker" (Wolfe, 2006). Designating roles is not only useful for organizing classroom routines and distributing responsibility. Roles also serve a function for learning, especially when activities are collaborative and materials must be shared. Designing "procedural" roles around materials and their uses entail implicit "intellectual" roles tied to procedures (Herrenkohl, 2006). For example, in her study of elementary student roles, Herrenkohl (2006) found that if you are the designated "reporter" in a group of students engaged in learning science, this procedure calls on certain ways of thinking particular to what reporters do. The ways in which intellectual labor was embedded in the technician's scope of work (i.e. their technical knowledge) was partly what was at stake in our designation of the "robot controller," or technician role.

Data and analytic approach

All group lessons were video recorded by designated members of the research team, who also played a role in on-site design memos and teaching memos during the curriculum implementations. The main data source for the current analysis is approximately 25 h of video recordings. In broad sweeps, the analysis process included:

- writing design memos;
- content logging video and open coding for aspects of debugging;
- iteratively refining the analytic lens on the data through analytic memos; and
- conducting microanalysis of key instances of analytic categories that emerged (Figure 3).

At some moments, these were contemporaneous steps, but for the purpose of clarifying our approach, we briefly outline each as separate steps before presenting the empirical cases.

Design memo-ing. Our research team produced 48 design memos, one for each group lesson. This step began during the first curriculum implementations, so the analysis was already underway before data collection had ended. This meant that we were noticing patterns in teaching and learning debugging, *which then changed how we were teaching children to debug*. For example, during implementation in the first classroom site, we noticed that, while they recognized a program had not executed as expected, some students

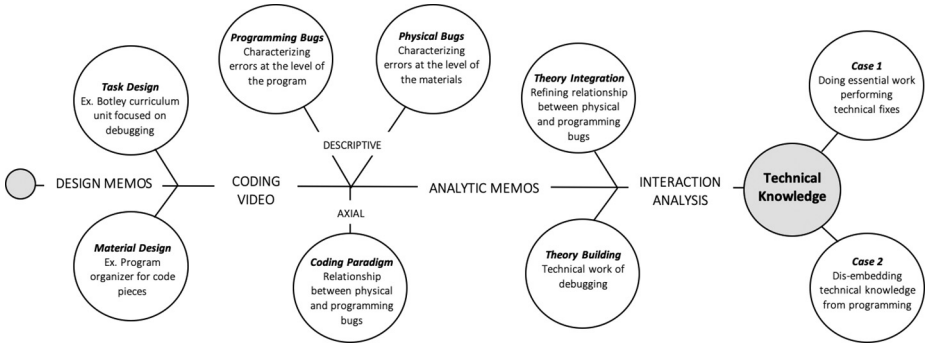


Figure 3. Schematic of the analytic process

struggled to locate the source of the error or to apply suitable debugging strategies. Sometimes they defaulted to adding a backward arrow to the end as if to "undo" an error somewhere in the program. For instance, following an especially lengthy debugging attempt that involved ten separate rounds of debugging of what should have been a seven-code program, one researcher documented in her memo:

> The program is run again and Botley goes too far before turning. [The student] tries to add backwards to the end of the code, but [the teacher] helps her to identify what went wrong and where in the program the bug is. [The student] does not quite see it [.. .] While the lesson was originally focused on algorithmic thinking, it elicited a lot of discussion and practice with debugging [.. .] Programming with physical obstacles and goals, there may be multiple opportunities to debug. [10-22-19, RP]

Seeing that this debugging step or skill could benefit from a more explicit pedagogical focus, we decided to make debugging the focus of an entire curriculum unit, using one robot called Botley. However, when we implemented debugging tasks in subsequent groups, it was not the case that learning to debug was necessarily restricted to this particular design nor exclusively to Botley. Debugging continued to occur pervasively in all tasks and robots, leading us back to the data to try to account for patterns in debugging beyond our designs.

Content logging and coding. Between the first and second classroom implementations, we began a months-long process of content logging the video data (Jordan and Henderson, 1995). As we logged, we developed codes to describe the data. Initially, these were highly descriptive and focused on easily observable aspects of debugging such as bug types, the position of bugs in programs, numbers of attempts or trials and debugging strategies, such as adding codes onto the end of the program or "wiping" the entire program and starting over (Silvis *et al.*, 2021). Early on, at this very grounded and descriptive level, we were still regarding the *program* as the primary source of errors. As implementations continued through the school year, and our memos and design debrief discussions became more analytical, we started to lift up from the descriptive categories and look beyond our designs to what else was happening in interactions that might be consequential for debugging. It was at this time that we began to foreground something that had been latent in our debriefs, discussions and designs: *the use (and misuse) of coding materials*.

On the one hand, we knew that the ways in which children handled the robot and accessories were critical, so much so that we designed additional hand-made aids to supplement the commercially available kits that came included with the toys. For example, we had designed an item we called a "program organizer," which served to order and sequence codes that would sometimes become jumbled or get strewn about in the excitement of children's coding. It was this sort of problem that drew our analytical attention to the ways in which physical materials were causing what looked like bugs in the program but were actually "bugs" in the apparatus. We coded for the various forms of breakdown we observed and identified a number of categories for bugs in the physical apparatus, including material mishaps, re-initialization errors, operator errors, mechanical issues and intentional user errors that we called "building in a bug." Furthermore, these types of errors occurred as frequently as programming errors, with operator or controller errors accounting for a substantial proportion of all bugs (Silvis *et al.*, 2021). Material mishaps like the robot getting hung up on the mat required technical interventions, and so did operator errors like pressing the incorrect buttons or mechanical errors like Bluetooth pairing problems. We observed that what we termed *physical bugs* often exacerbated and were laminated onto programming bugs in ways that complicated successful debugging (ibid). Treating *physical bugs* as a category of problems that co-occurred in consequential ways with *programming bugs*, we shifted our analytic lens to the relationship between these two high-level categories of bugs.

Refining analytic questions and memos. From the very start of our coding, we produced approximately 200 analytic memos, and theoretical notes that helped us differentiate between categories and concepts (Glaser and Strauss, 1967). These memos range in length from a few words (i.e. analytic notes) to several pages. The majority of the memos are linked to coded segments of tape; however, longer memos integrated emerging concepts and cut across segments, lessons, groups and school implementations. For us, analytic memos were most useful for extending the theoretical integration of the data and developing the relationship between physical and programming errors (Glaser and Strauss, 1967). For example, Memo 114, corresponding to a segment of video sub-coded for a type of physical bug *wrong-button press* and cross-coded with a pedagogical approach *modeling a debugging strategy*, reads:

> Whereas before, Eli had needed scaffolding for building the program through one-by-one coding, now Cory receives the same supports but *for the physical toy itself*. Both the program and the physical material at times require decomposing into single units in order to reduce errors and focus on the symbol-movement-button correspondence *emphasis added* [6/8/20, ET].

The interrelation between programming and physical bugs ultimately formed the "axis" around which further coding was conducted, eventually establishing the core category of the emerging theory (Kelle, 2007). Integrating this emerging theoretical construct with related literature on debugging and technical knowledge, we were now operating under what Strauss referred to as a "coding paradigm," where the meaningfulness of a piece of data was viewed in terms of physical materials and their mis/use on debugging. In other words, we were developing a theory that positioned the technical knowledge of the person in the role of the robot controller who was responsible for how the materials were used, as consequential for successful debugging and programming.

Selection and microanalysis of cases. When it came time to identify comparative cases for closer microanalysis, we drew on a number of selection criteria. First, we knew that physical bugs occurred across a wide variety of materials and objects associated with the coding robots. We, therefore, selected cases that illustrate different robots or parts of the physical apparatus that can cause mechanical trouble. Second, we identified cases where teachers played a key role in children developing technical knowledge because, in our task designs, the teacher was instrumental for supporting or thwarting technical work. Third, we selected cases where technical knowledge emerged from the group's collaborative work rather than being siloed. While essential technical knowledge sometimes gets hidden away in labs or embedded in the work of the programmer, (Barley and Bechkey, 1994; Shapin, 1989; Smith, 2009), we chose cases where technical knowledge was foregrounded as an essential function of the group. Fourth, we chose cases where programming and physical bugs co-occurred because we had seen how resolving both orders of bugs highlighted the essential role of the person performing the technical work (Silvis *et al.*, 2021).

We conducted interaction analysis (IA) of focal cases that represented these key criteria (Jordan and Henderson, 1995). While the outputs of IA vary, most analysts hold to a set of shared principles (Hall and Stevens, 2016), a number of which guided our analysis. First is the importance of learning as a "member's phenomenon," which focuses analysis not on *a priori* theories of or adults' goals for a particular learning phenomenon (in our case, debugging) but on what learners collectively believe they must accomplish (Keifert and Stevens, 2019; Stevens, 2010); while conducting analysis, we operated under the assumption that learners' collective accomplishments always take place in a negotiated and powered space. Second, and relatedly, knowledge is achieved through purposeful interaction, not as an individual accomplishment, incidentally acquired; knowledge is not something a learner

has (or lacks), but rather knowledge is observed in-use (Hall and Stevens, 2016). Building from this, a third principle we drew on was that these "uses" are always "re-uses," built upon the substrates of prior interactions, which become sedimented in materials and ideas, and are carried into their subsequent uses (Goodwin, 2018). Guided by these and other guiding principles of IA, we examined multimodal ways technicians and their collaborators accomplished tasks, paying particular attention to children's multimodal use, sharing and discussion of materials. In what follows, we present three instances in which children demonstrated and developed their technical knowledge in the context of debugging tangible programs.

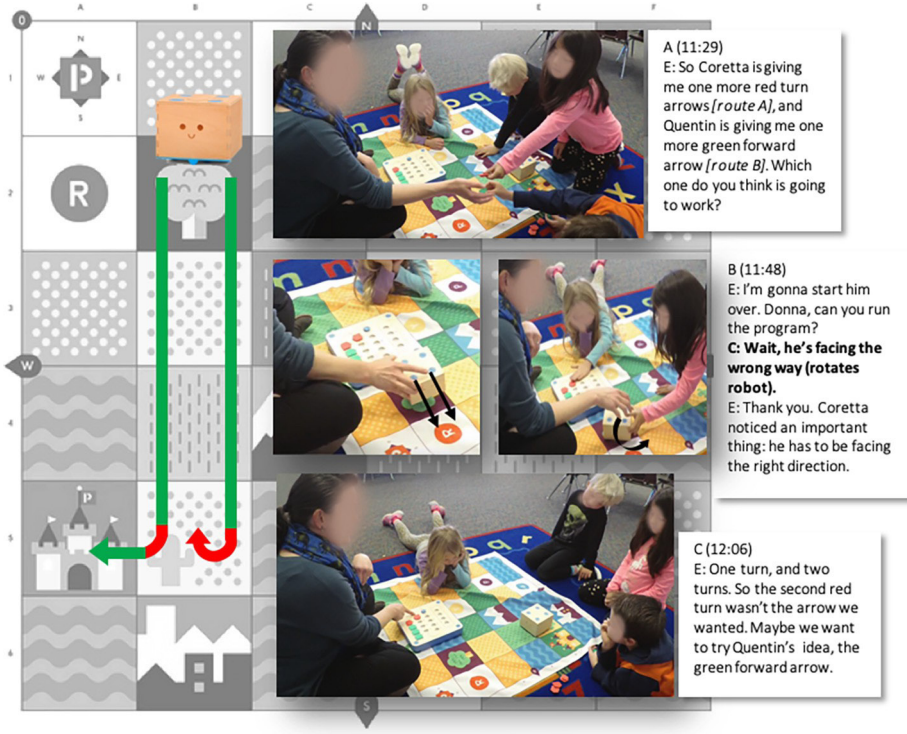
Analytic findings

Case 1: doing essential work of performing technical fixes

Working in groups and taking on different roles multiplied the number of possible problems to solve. When physical bugs like user errors (i.e. pushing the wrong buttons) and programming bugs like miscoding the turn unit (i.e. ROTATE, ROTATE, rather than ROTATE, FORWARD) *co*-occurred, it was often difficult for novice coders to identify the source of the error to debug it (Silvis *et al.*, 2021). Children had to reason whether their program yielded an unexpected error because the machinery was misused or because the program was faulty. Ideally, user error could be mitigated so that children only needed to focus on the *program* to debug. One way this mitigation of user error came about was when children demonstrated their technical knowledge "just-in-time," sometimes as the program was running but before the physical error occurred. Despite our designs for loosely structured designation of roles, students (and teachers) moved fluidly in and out of different roles in-task. Spontaneously stepping into the role of the technician was sometimes necessary to neutralize the effect of a physical bug on the successful execution of a program (Barley and Bechky, 1994).

Coretta and her group were learning how to program Cubetto to travel on a path that required the program FORWARD, FORWARD, FORWARD, ROTATE RIGHT, FORWARD. Coretta and Quinten had different understandings of the semantics of the turn unit at the end of the program and disagreed about whether ROTATE RIGHT, *FORWARD* (Quentin's) or ROTATE RIGHT, *ROTATE RIGHT* (Coretta's) was the correct end-sequence [Figure 4(A)]. Their teacher Ms D suggested that they first try Coretta's idea. However, after Coretta placed her (incorrect) code, Ms D inadvertently oriented the robot facing the *wrong* way on the path [Figure 4(B)]. As Donna, who was nominally assigned to the technician role, pressed the Go button, Coretta noticed the robot's improper orientation and quickly turned it in the proper direction [Figure 4(B)]. Ms D thanked Coretta for "noticing an important thing", and the robot proceeded to run Coretta's planned (buggy) program. The children subsequently successfully debugged the program, adopting Quinten's original idea [Figure 4(C)].

The children were able to focus on what was wrong *within the code* because Coretta, demonstrating her technical knowledge of the machinery, ensured that the robot operated correctly. Coretta repositioned Cubetto just as the program started, ensuring that the program ran as expected, although it was buggy. *Their debugging was subsequently successful, and children were able to locate the source of the programming error because Coretta prevented technical trouble before it occurred.* During scientific experiments or technical demonstrations, machine operators and technicians are often called upon to recognize trouble before it arises or prevent breakdowns before they occur. Barley and Bechky (1994) explain how "unless a lab worker called attention to the matter, a casual observer might not notice that an error had been neutralized" (p. 110). The essential work



Notes: Clockwise from Emma: Donna, Isaiah,Coretta, Quentin

Figure 4.
Correta corrects the robot's starting orientationjust-in-time

Coretta performed may have gone unnoticed because her fix occurred just-in-time; however, Ms D's recognition of the "important thing" Coretta did gestures to the vital role of technical knowledge involved in programming.

Case 2: disembedding technical knowledge from programming

The technical knowledge Coretta demonstrated was an essential part of collective debugging. But how is such knowledge organized pedagogically in the normal course of debugging? One day, while teaching the correspondence between codes and the robot's movements, Mr K shifted the focus from codes' abstract symbolic correspondence to call explicit attention to the significance of technical details for the success of the program. Cory (the *Programmer*), Eli (*Assistant Programmer*) and Stanley (*Technician*) were working together on a task with Botley called "Crack-the-code," a task designed to decode a program by watching the robot execute a hidden sequence (Shumway *et al.*, 2021). On their fourth debugging attempt decoding the hidden program FORWARD, ROTATE RIGHT, FORWARD, Cory had designed the faulty program FORWARD, ROTATE RIGHT, ROTATE RIGHT. Before running this program, Stanley cleared the codes already entered in the remote by pressing the trash can button. Failures to "trash" or reset the program resulted in a type of what we termed *initialization errors* (running the old program and then the new program as a single procedure), a pervasive type of user error across all tasks and groups and a necessary step in many programming environments (Silvis *et al.*, 2021) [1]. Compounded by the fact that children had not yet established a stable understanding for the

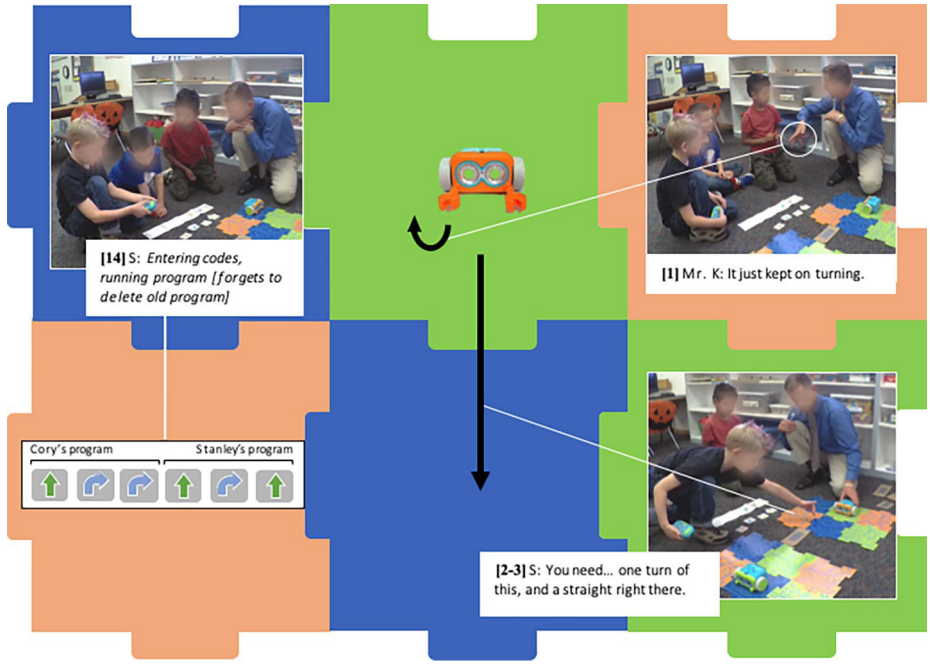


Figure 5.
Stanley debugs Cory's code, but creates a physical bug

Notes: Left to right: Stanley, Eli, Cory, Mr. K

arrow-movement correspondence or expectations for robot movement conventions (Silvis *et al.*, 2020), forgetting to trash the previous program was a source of significant confusion for children trying to diagnose the error source.

When Cory's buggy program "just kept on turning" [Figure 6, line 1], which Mr K. demonstrated by rotating his hand in the air over the grid, Stanley stepped in to help with programming. Alternately gesturing from the program organizer to the grid space, Stanley provided a rationale for the unit of the turn, and Mr K moved the robot on the grid, simulating these commands [lines 2–4]. Mr K checked Cory's agreement, asking what he thought and Cory shrugged [line 5–6]. Their designated programmer was at a loss for what to do next. The group collectively decided to run Stanley's program, and Eli placed a FORWARD code on the program organizer at the end of the sequence [lines 7–13]. However, Stanley had become preoccupied by the programming while performing technical work; *he forgot to press trash* and merely entered the correct codes FORWARD, ROTATE RIGHT, FORWARD into the remote before executing the program [line 14]. Because Stanley had not cleared the previous codes with the remote, the robot failed to reinitialize and moved FORWARD, ROTATE RIGHT, *ROTATE RIGHT,* FORWARD, before Mr K interrupted the run-through and picked up the robot, halting the demonstration.

K: It just kept on ((*rotates hand in air*)) turning ((*extended eye contact with Cory*))
S: Maybe you need a straight ((*gestures in front of robot*)), and one turn ((*points at codes on program organizer*)) of this, and a straight right there ((*gestures in front of robot*))
[Mr K simulates these movements].

K: ((*extended eye contact with Cory*)) what do you think?
C: ((*shrugs shoulders*))
K: You don't know? ((*repositions robot at start position*)) Okay, well let's try Stanley's ((*removes end code on program organizer*)) So you said not that, Stanley, do what?
S: Put a straight ((*points at F code*))
K: Okay, do it
S: ((*points to the code again*)) put this [inaudible]
K: Eli [inaudible]
E: ((*places forward arrow on end of program*))
S: ((*enters codes on remote, runs program*))
K: ((*picks up robot*)) Oh, you didn't click trash. [feigning frustration] Darn it, Stanley. If you don't push trash, it just keeps putting new, new, new ones on, buddy.

Feigning frustration, Mr K explained that Stanley "didn't push trash" [line 15]. He then seized the opportunity to explicitly connect the physical bug with the programming bug. He explained that if they forget to delete the faulty codes, the robot will rerun the old program first, adding on "new, new, new ones" [line 16]. To emphasize how this *technical* error would work, Mr K produced a series of circular hand movements, punctuating each "new" code

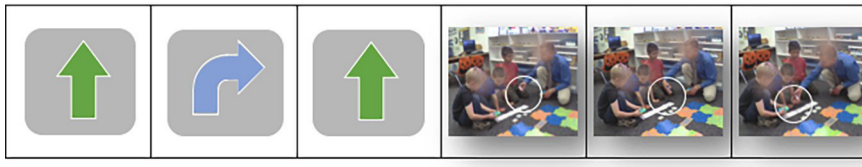


Figure 6.
Mr. K's gesture over the programe organizer incorporates physical and programming bugs

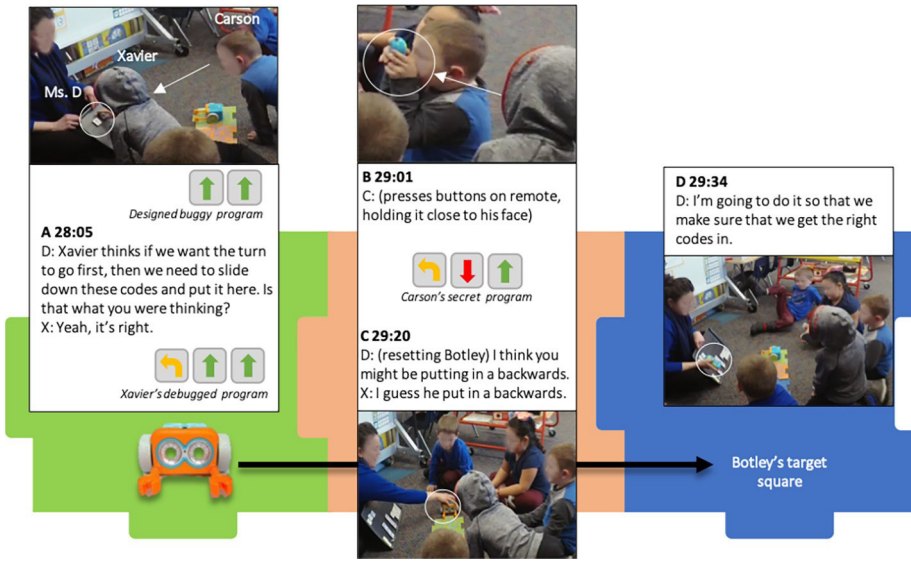


Figure 7.
Carson secretely add a backwards when entering Xavier's debugged program into the remote

with another circle of this hand progressing further down the program (Figure 7). This gesture closely resembled the motion he had used when explaining to Cory how his earlier *programming* bug had made the robot continue to just "keep on turning" [line 10]. More importantly, Mr K performed this circular gesture in the air over the program organizer to draw their attention to the consequence for the program of an error in the physical domain. His "environmentally-coupled gesture" performed multiple semiotic functions, both referencing the current material field of action and recalling or recycling a previously meaningful gesture as a novel resource for sense-making (Goodwin, 2018).

Through his words and gestures, Mr K explicitly drew connections between physical and programming breakdowns, disembedding the essential technical function from the programming domain and showing how they were related. In doing so, he pulled back the curtain that obscures how robots run programs, revealing a mechanism that relies on the technical knowledge of the person controlling the robot. For novice programmers, learning just how much information you have to give the computer is part of overcoming what Pea (1986) called "intentionality bugs," where the student endows the program with the capability to "know" or "see" what to do without being told. In fact, Botley cannot "go beyond the information given" (p. 29), and must be told *by the technician* to delete the previous commands, technical knowledge that, for Stanley, was still being organized. Calling attention to technical details is precisely what is needed when learning to program in kindergarten. When the frequent co-occurrence of physical and programming bugs challenges novice programmers, calling attention to the consequences of technical work can mean the difference between knowing *what* codes to use and knowing *why* a program works (or fails).

Case 3: the technical undoing of a programmer's redoing

Coretta's demonstration of technical knowledge and Mr K's disembedding of technical breakdowns from programming errors were examples of how technical knowledge-building supported collaborative debugging. At other times, resolving programming errors or teasing them out of physical errors involved a technician's clever trouble-making rather than their troubleshooting. While essential technical knowledge can mitigate errors and save a program from catastrophic failure, technicians possess specialized skills and are in a unique position to undermine an experiment's integrity. It is precisely because of their know-how that technicians hold the disruptive power to undo the experiment or break the mechanism. Commenting on the importance of the machine operators in Robert Boyle's 17th-century experiments, Shapin (1989) wrote that:

> Technicians' doings then become an important source of opaqueness … in fact, situations in which experiments miscarried provide by far the richest source in Boyle's texts for establishing the nature and scope of his assistants' laboratory work (p. 558).

In the next example from a Kindergarten classroom, Carson demonstrated the nature and scope of *his* technical knowledge and took it as an opportunity for the creative undoing of what would have been a successful redoing of the program.

Ms D was assisting Xavier as he performed a challenging debugging operation that required inserting a missing left turn code at the beginning of their program, which was buggy by design (Fields *et al.*, 2021) (Figure 5, Frame A). Ms D then instructed Carson to enter Xavier's program into the remote: ROTATE LEFT, FORWARD, FORWARD. Carson, who had been eagerly awaiting his turn to operate Botley's remote control, had carefully watched over his shoulder as Xavier debugged the program (Frame A). Contrary to our pedagogical focus on *programming*, the vast majority of children desired to perform

technical work and vied for opportunities to take control of the robot. The role of the machine operator can be one of prestige when students' values for particular types of work are centered (Hennessey Elliott, 2020).

Holding the remote close to his face, Carson secretively pressed the arrow buttons and then pressed Go, and the robot moved ROTATE LEFT, *BACKWARD*, FORWARD (Frame B). Confused, Ms D asked him to verify he had pressed the correct buttons (Frame C). Xavier, who had now been looking over *Carson's* shoulder, also surmised that Carson had pressed the backward button, keying into how the robot was not operating as it should, given the debugged program Xavier had specified. Carson insisted he had faithfully entered ROTATE LEFT, *FORWARD*, FORWARD and ran it once more. Botley again moved ROTATE LEFT, *BACKWARD*, FORWARD. Eventually, Ms D took the remote from Carson and entered Xavier's debugged program (Frame D), and Botley reached the target square. The technician wields power and influence in programming tasks, which becomes visible when their knowledge is socially distributed, and they opt to take the task in another direction from their collaborators. When Carson obscured his own actions by holding the remote where no one could see what he was really up to, he demonstrated both his desire to playfully subvert the experiment and the potential scope of his growing technical and programming knowledge. His crafty use of the remote control was not a mistake; rather, it signaled his creative understanding of how to complicate the task.

Calling attention to Carson's subversion *as skill* requires a pedagogical reorientation to technical knowledge. While we embraced the approach of Fields *et al.* (2021) to design for debugging and regularly embedded programming errors in tasks as part of the learning design, Carson's technical demonstration pushes our tangible programming designs further in two ways. First, Carson, not us, devised the form he wanted the bug to take. Second, by intentionally embedding *a physical bug* in a program debugging task, Carson challenged Xavier, his other group members and his teacher to disentangle the two types of bugs. Without Carson's technical subversion, Xavier had only to debug the program by adding the missing forward code, a presumably simple task. By layering a physical bug into the task *by his own design*, Carson created a task with even more complexity. Whereas Ms D placed a higher value on "getting the right codes in" to validate Xavier's redoing of the program, Carson's technical knowledge demonstrated how much can be learned about the relationship between physical and programming errors through technicians' undoings.

Discussion

Technical work as a feature of learning, not a bug

Through these three examples, we elaborated on how children were performing their technical knowledge and how learning to make programs work was contingent on learning how to work tangible coding robots. The cases demonstrate how debugging is an important context for developing technical knowledge, bridging a gap between what little we know about young children's CT with their growing knowledge of the physical components of computational systems. In the case of Coretta, essential knowledge like the ability to perform just-in-time fixes at the level of the physical apparatus was instrumental in ultimately resolving a programming error. In the case of Stanley and Mr K., resolving a programming error presented a pedagogical opportunity to dis-embed technical and programming domains. In Carson's case, intentionally embedding a physical bug and subverting successful program debugging represented another form technical work can take. Taken together, these examples of the technical knowledge involved in tangible programming raise a number of questions about the nature of computing and the organization of technical work in learning environments.

How is technical work being organized in CS and adjacent domains? As hived off, in the hands of particular technicians? As distributed among teams of operators? As something that programmers, too, must learn and do? While engaged in tangible programming, technical work and program planning overlapped as children assumed fluid roles; being a programmer involved performing some technical work and being a technician involved some knowledge of programming. For example, when Coretta spontaneously reoriented Cubetto as she was programming it, she engaged in a familiar form of technical work, where abstract reasoning was both conceptual and physically instantiated (Buccharelli and Kuhn, 1994). Or, when Stanley volunteered to help Cory solve the programming problem, his unstable technical knowledge was nonetheless essential for operating the robot. This sort of fluid ingenuity reflects ongoing restructuring of engineering and design fields, where the organization of technical work shifts as emerging occupations bundle technical tasks in new ways.

We encourage learning designers to examine the organization of technical work more closely in a range of educational settings. In computer labs, are we planning for how people learn to operate tools and keep them in working order, including powering up and down, installing apps and software, running updates, managing passwords, connecting compatible components and accessories and maximizing the lifespans of devices? In makerspaces, are we providing instruction for disentangling software and hardware problems by *teaching* troubleshooting as opposed to assuming people learn to fix computers through trial and error? In everyday learning situations, do we take for granted that technical knowledge is somehow a default setting of growing up surrounded by screens and devices or is there something special technicians must know and do?

We need not limit ourselves to patently technical endeavors like programming or engineering to witness the pervasive organizational hierarchy between "mental" and "manual" labor and the back-staging of essential work (Rose, 2004). All around us are examples of the ways in which work is organized to downplay technical knowledge. The Covid-19 pandemic and resulting shut-downs placed essential workers in sharp relief, drawing widespread attention to the many ways that technical knowledge is involved in nearly every facet of social life, from lab techs who monitor and recalibrate medical instruments that test for disease to prep cooks who prepare a perfect dice for your meal at a restaurant (Wu, 2020). To the degree that the physician's or the chef's work appears to the patient or the diner to be single-handed works of brilliance, essential technical knowledge goes unrecognized. Recognizing how children's programming reflects historical divisions of labor in broader society is part of designing approaches to CS and engineering domains in which technical knowledge is valued as "sociotechnical activity rooted in specific contexts and communities" (Nasir *et al.*, 2021, p. 558), rather than merely a set of rote skills for operating machines. Learning to labor (Willis, 1981) in technology-rich learning environments means accounting for these historical patterns and disrupting technoscientific hierarchies and ideologies (Philip and Sengupta, 2020).

Foregrounding technical work in classrooms, computer labs and other "backrooms of learning" is part of establishing more equitable learning arrangements. To bring technicians out of backrooms, one must first know where the backroom is, visit it, spend time there and attempt to understand its sociotechnical organization. The learning environments of young children are some of education's best-hidden backrooms (to say nothing of the relative invisibility and undervaluing of early childhood educators, essential workers in their own right). But bringing technicians out of their backrooms involves less physical repositioning and more an intellectual one (Star and Strauss, 1999; Suchman, 1995). Rather than simply accommodating technical work(ers) within the scope of engineering and CS – and then

mapping this knowledge onto young children in STEM – we might ask: what are the ways of thinking and technical knowledge that they embody that help us reimagine the means and ends of CS or CT? Conversely, what technical knowledge is always already required of those wearing proverbial white lab coats, and how can we dis-embed this knowledge as a focus of instruction? This focus on technical work is becoming even more important as the field embraces physical computing, even as these learning designs risk perpetuating oppression and extraction of human labor within "ideologies of workforce readiness and a 'skilled technical workforce'" (Philip and Sengupta, 2020, p. 11). We see multiple ways to incorporate technical work as a feature and not a bug in the organization of activity to expand what counts as authentic computing.

Conclusions and implications

Revaluing (how) the technical matters in debugging To make multiple forms of technical knowledge more visible across computational contexts, we conclude with a series of ways to surface and revalue the work of making things work, specifically when making things work means debugging them. The first pertains directly to early childhood debugging and CT. Rather than simplifying coding using toys that are deceptively user-friendly, we believe it is productive to expose children to the inner workings of the computational system that is black-boxed under normal working conditions. Previous studies of tangible programming attempted to "un-black-box" components to teach computing (Resnick *et al.*, 2000), and more recent designs engage children in thinking about the entangled relationship between hardware and software (e.g. Bers *et al.*, 2014) or DbD (Fields *et al.*, 2016, 2021). Approaches like these draw out the relationship between physical and programming domains and prepare children to engage with *both* orders of bugs that co-occur even when we have not designed for them.

For us (and for students) to design for debugging that facilitates learning, we need more accounts of the interactional texture and nuance of children's programming experiences, including how they reason with one another about relationships between programming and physical domains. Technical knowledge of young children like Carson and their creative perspective on physical bugs is instructive for pushing the limits of our pedagogical approaches with young children in CS and expanding who is in control of DbD. The DbD paradigm challenges learning designers to embed bugs as pedagogical tools, involve learners in the bugs' selection and incorporate both orders of bugs: physical/material and programming/symbolic (Fields *et al.*, 2021). Our approach adds to this work by illustrating what DbD looks like in a novel computational environment with tangible programming toys and with young children learning to code. Our designs also contribute to understanding how larger groups of children – assigned different roles and working within a division of labor – collaboratively reconciled planned and unplanned bugs in the program and the physical materials. This is important for preparing learners to work on teams where people have heterogeneous roles and specialized knowledge, in and out of engineering and CS fields.

A second way in which we can reposition technical knowledge is by organizing learning environments that center on the types of roles that children value. As we orient children to computational objects in changing learning environments, we are also learning to orient children as computational thinkers. When we bring young children into computer labs, tinkering programs and maker spaces, we can take valuable lessons from established organizational principles of early childhood environments: collaborative work, role-taking and special attention to physical manipulatives and tangible tools (Vossoughi *et al.*, 2021). To account for young children's particular forms of CT we need to take seriously the "revaluing of the concrete" by recognizing how children's knowledge and interests are

bound up in concrete action (Montessori, 1949; Turkle and Papert, 1992). This could take the form of statements from a teacher praising students for catching technical errors and stating explicitly that fixing those is just as important as using the right codes to accomplish the immediate goal.

Focusing on the technical, concrete aspects of programming errors is an extension of the guiding principles of epistemological pluralism, which inspired the design of many of the coding toys that appear in early learning environments today (Turkle and Papert, 1992). As early childhood computing becomes more commonplace, it is, therefore, ironic that *learning to build working programs* should drive learning designs, as opposed to *learning how to work the machine*. The tendency to background technical knowledge and foreground programming knowledge is even more curious when we consider how children are drawn to the materials themselves and place a high value on doing technical work. Despite the learning goals and roles that adults value, children collaboratively determine which roles are high-status and collectively position each other within them (Fields and Enyedy, 2013; Hennessey Elliott, 2020). Identifying aspects of computing that bear significance to children is important because computational roles and identities are not always geared toward programming languages per se; students are sometimes more invested in material artifacts (Haduong, 2019).

If we want to approach children's learning as a member's phenomenon (Keifert and Stevens, 2019) and value them as members of societies they are helping to form, then this means learning to see children's technical knowledge as meaningful *to them* and as vital for programming. As we pay attention to children's own perspectives on bugs, programming, robots or technical work, we are keying into the values children like Coretta, Stanley or Carson were developing for types of labor and roles in society, be it essential (yet easy to overlook) or visible and highly esteemed. As society is organizing more and more technical roles for learners, bringing young children's values for technical knowledge into view is part of establishing more equitable and endogenous forms of learning with technology. With children as young as preschool-age now learning to code, we hope this presents an opportunity to appreciate the critical learning that takes place during early childhood, to consider the powered relations within technological practices and reaffirm the role learning sciences can play in designing for young children's thriving. Whether and in which ways computer technologies play into young children thriving is more than a mere technical detail. It is an open question we hope to see addressed in designs to come.

Note

1. Indeed, this is a common error in traditional computer programs where previous values of variables need to be erased or initialized, sometimes through their own special function that is typically at the start of the program. Failure to re-initialize can cause a number of bugs.

References

Barley, S.R. and Bechky, B.A. (1994), "In the backrooms of science: the work of technicians in science labs", *Work and Occupations*, Vol. 21 No. 1, pp. 85-126.

Barley, S.R. and Orr, J.E. (1997), "The neglected workforce", in Barley, S.R. and Orr, J.E. (Eds), *Between Craft and Science: Technical Work in US Settings*, Cornell University Press, Ithaca, New York, NY.

Bers, M. and Sullivan, A. (2019), "Computer science education in early childhood: the case of scratch Jr", *Journal of Information Technology Education: Innovations in Practice*, Vol. 18, pp. 113-138.

Bers, M.U. (2018), *Coding as Playground: Programming and Computational Thinking in the Early Childhood Classroom*, Routledge, New York, NY.

Bers, M. and Horn, M.S. (2010), "Tangible programming in early childhood: revisiting developmental assumptions through new technologies", in Berson, I.R. and Berson, M.J. (Eds), *Childhood in a Digital World*, Information Age Publishing, Greenwich, CT.

Bers, M.U., Flannery, L., Kazakoff, E.R. and Sullivan, A. (2014), "Computational thinking and tinkering: exploration of an early childhood robotics curriculum", *Computers and Education*, Vol. 72, pp. 145-157.

Brennan, K. and 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, Vancouver*, April, Vol. 1, p. 25.

Clarke-Midura, J., Lee, V.R., Shumway, J.F. and Hamilton, M.M. (2019), "The building blocks of coding: a comparison of early childhood coding toys", *Information and Learning Sciences*, Vol. 120 Nos 7/8, pp. 505-518.

Clarke-Midura, J., Kozlowski, J.S., Shumway, J.F. and Lee, V.R. (2021a), "How young children engage in and shift between reference frames when playing with coding toys", *International Journal of Child-Computer Interaction*, Vol. 28, p. 100250.

Clarke-Midura, J., Silvis, D., Shumway, J.F., Lee, V.R. and Kozlowski, J.S. (2021b), "Developing a kindergarten computational thinking assessment using evidence-centered design: the case of algorithmic thinking", *Computer Science Education*, Vol. 31 No. 2, pp. 117-140.

DeLiema, C., Enyedy, N. and Danish, J.A. (2019), "Roles, rules, and keys: how different play configurations shape collaborative science inquiry", *Journal of the Learning Sciences*, Vol. 28 Nos 4/5, pp. 513-555.

DeLiema, D., Dahn, M., Flood, V.J., Abrahamson, D., Enyedy, N. and Steen, F. (2020), "Debugging as a context for collaborative reflection on problem-solving processes", in Manolo, E. (Ed.), *Deeper Learning, Dialogic Learning, and Critical Thinking: Research-Based Strategies for the Classroom*, Routledge, New York, NY, pp. 209-228.

diSessa, A. (2000), *Changing Minds: Computers, Learning, and Literacy*, MIT Press, Cambridge, MA.

Fields, D. and Enyedy, N. (2013), "Picking up the mantle of 'expert': assigned roles, assertion of identity, and peer recognition within a programming class", *Mind, Culture, and Activity*, Vol. 20 No. 2, pp. 113-131.

Fields, D., Searle, K.A. and Kafai, Y.B. (2016), "Deconstruction kits for learning: students' collaborative debugging of electronic textile designs", in Blikstein, P., Berland, M. and Fields, D.A. (Eds), *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education, New York, NY*, ACM, pp. 82-85.

Fields, D.A., Kafai, Y.B., Morales-Navarro, L. and Walker, J.T. (2021), "Debugging by design: a constructionist approach to high school students' crafting and coding of electronic textiles as failure artefacts", *British Journal of Educational Technology*, Vol. 52 No. 3, pp. 1078-1092.

Glaser, B.G. and Strauss, A.L. (1967), *The Discovery of Grounded Theory: Strategies for Qualitative Research*.

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

Haduong, P. (2019), "I like computers: I hate coding': a portrait of two teens' experiences", *Information and Learning Sciences*, Vol. 120 Nos 5/6, pp. 349-365.

Hall, R. and Stevens, R. (2016), "Interaction analysis approaches to knowledge in use", in DiSessa, A.A., Levin, M. and Brown, N.J.S. (Eds), *Knowledge and Interaction: A Synthetic Agenda for the Learning Sciences*, Routledge, New York, NY.

Hamner, E., Lauwers, T. and Bernstein, D. (2010), "The debugging task: evaluating a robotics design workshop", *Association for the Advancement of AI Spring Conference: Educational Robotics and Beyond*.

Heikkilä, M. and Mannila, L. (2018), "Debugging in programming as multimodal practice in early childhood education settings", *Multimodal Technologies and Interaction*, Vol. 2 No. 3, doi: 10.3390/mti2030042.

Hennessey Elliott, C. (2020), "'Run it through me:' positioning, power, and learning on a high school robotics team", *Journal of the Learning Sciences*, Vol. 29 Nos 4/5, pp. 598-641, doi: 10.1080/10508406.2020.1770763.

Herrenkohl, L.R. (2006), "Intellectual role-taking: supporting discussion in heterogeneous elementary science classes", *Theory into Practice*, Vol. 45 No. 1, pp. 47-54.

Horn, M. (2018), "Tangible interaction and cultural forms: supporting learning in informal environments", *Journal of the Learning Sciences*, Vol. 27 No. 4, pp. 632-665.

Jordan, B. and Henderson, A. (1995), "Interaction analysis: foundations and practice", *Journal of the Learning Sciences*, Vol. 4 No. 1, pp. 39-103.

Kafai, Y.B., DeLiema, D., Fields, D.A., Lewandowski, G. and Lewis, C. (2019), "Rethinking debugging as productive failure for CS education", *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, February, pp. 169-170.

Keifert, D. and Stevens, R. (2019), "Inquiry as a members' phenomenon: young children as competent inquirers", *Journal of the Learning Sciences*, Vol. 28 No. 2, pp. 240-278.

Kelle, U. (2007), "The development of categories: different approached in grounded theory", in Bryant, A. and Charmaz, K. (Eds), *The SAGE Handbook of Grounded Theory*, SAGE.

Kim, C., Yuan, J., Vasconcelos, L., Shin, M. and Hill, R.B. (2018), "Debugging during block-based programming", *Instructional Science*, Vol. 46 No. 5, pp. 1-21.

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L. and Zander, C. (2008), "Debugging: a review of the literature from an educational perspective", *Computer Science Education*, Vol. 18 No. 2, pp. 67-92.

McNerney, T.S. (2004), "From turtles to tangible programming bricks: explorations in physical language design", *Personal Ubiquitous Computing*, Vol. 8, pp. 326-337.

Montessori, M. (1949), *The Absorbent Mind*, The Theosophical Publishing House.

Nasir, N.S., Lee, C.D., Pea, R. and McKinney de Royston, M. (2021), "Rethinking learning: what the interdisciplinary science tells Us", *Educational Researcher*, Vol. 50 No. 8, pp. 557-565, doi: 10.3102/0013189X211047251

Papert, S. (1980), *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, New York, NY.

Pea, R.D. (1986), "Language-independent conceptual 'bugs' in novice programming", *Journal of Educational Computing Research*, Vol. 2 No. 1, pp. 25-36.

Pea, R.D., Soloway, E. and Spohrer, J.C. (1987), "The buggy path to the development of programming expertise", *Focus on Learning Problems in Mathematics*, Vol. 9 No. 1, pp. 5-30.

Philip, T.M. and Sengupta, P. (2020), "Theories of learning as theories of society: a contrapuntal approach to expanding disciplinary authenticity in computing", *Journal of the Learning Sciences*, pp. 1-20.

Resnick, M., Berg, R. and Eisenberg, M. (2000), "Beyond black boxes: bringing transparency and aesthetics back to scientific investigation", *Journal of the Learning Sciences*, Vol. 9 No. 1, pp. 7-30.

Rich, K., Strickland, C., Binkowski, T.A. and Franklin, D. (2019), "A K-8 debugging learning trajectory derived from research literature", *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE'19)*, Feb. 27-Mar. 2, 2019, *Minneapolis, MN*.

Rose, M. (2004), *The Mind at Work: Valuing the Intelligence of the American Worker*, Viking Press.

Schweikardt, E. and Gross, M.D. (2008), "The robot is the program: interacting with roBlocks", *Tangible Embedded Interaction*, *Bonn*.

Searle, K.A., Litts, B.K. and Kafai, Y.B. (2018), "Debugging open-ended designs: high school students' perceptions of failure and success in an electronic textiles design activity", *Thinking Skills and Creativity*, Vol. 30, pp. 125-134.

Shapin, S. (1989), "The invisible technician", *American Scientist*, Vol. 77 No. 6, pp. 554-563.

Shetterly, M.L. (2016), *Hidden Figures: The American Dream and the Untold Story of the Black Women Mathematicians Who Helped Win the Space Race*, HarperCollins, New York, NY.

Silvis, D., Lee, V., Clarke-Midura, J. and Shumway, J. (2021), "Objects to debug with: how young children resolve errors with tangible coding toys", in de Vries, E., Yod, Y. and Ahn, J. (Eds), *Proceedings of the Annual Meeting of the International Society of Learning Sciences 2021*, *Bochum*.

Silvis, D., Lee, V.R., Clarke-Midura, J., Shumway, J. and Kozlowski, J. (2020), "Blending everyday movement and representational infrastructure: an interaction analysis of kindergarteners coding robot routes", in Gresalfi, M. and Horn, L. (Eds), *Proceedings of the Annual Meeting of the International Conference of the Learning Sciences*, *Nashville, TN*.

Smith, W. (2009), "Theatre of use: a frame analysis of information technology demonstrations", *Social Studies of Science*, Vol. 39 No. 3, pp. 449-480.

Socratous, C. and Ioannou, A. (2020), "Common errors, successful debugging, and engagement during block-based programming using educational robotics in elementary school", in Gresalfi, M. and Horn, L. (Eds), *Proceedings of International Conference of the Learning Sciences (ICLS) 2020*, *Nashville, TN*, International Society of the Learning Sciences.

Souza, I., Andrade, W., Sampaio, L. and Araujo, A. (2018), "A systematic review on the use of LEGO robotics in education", *2018 IEEE Frontiers in Education Conference (FIE)*, *San Jose, CA*, 2018, pp. 1-9. doi: 10.1109/FIE.2018.8658751.

Star, S.L. and Strauss, A. (1999), "Layers of silence, arenas of voice: the ecology of visible and invisible work", *Computer Supported Cooperative Work (CSCW)*, Vol. 8 Nos 1/2, pp. 9-30.

Stevens, R. (2000), "Divisions of labor in school and in the workplace: comparing computer and paper-supported activities across settings", *The Journal of the Learning Sciences*, Vol. 9 No. 4, pp. 373-401.

Stevens, R. (2010), "Learning as a member's phenomenon: toward an ethnographically adequate science of learning", *Yearbook of the National Society for the Study of Education*, Vol. 109 No. 1, pp. 82-97.

Shumway, J.F., Welch, L.E., Kozlowski, J.S., Clarke-Midura, J. and Lee, V.R. (2021), "Kindergarten students: mathematics knowledge at work: the mathematics for programming robot toys", *Mathematical Thinking and Learning*, pp. 1-29, doi: 10.1080/10986065.2021, 1982666.

Suchman, L. (1995), "Making work visible", *Communications of the ACM*, Vol. 38 No. 9, pp. 56-64.

Sullivan, A. and Bers, M.U. (2016), "Robotics in the early childhood classroom: learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade", *International Journal of Technology and Design Education*, Vol. 26 No. 1, pp. 3-20.

Sullivan, A., Elkin, M. and Bers, M.U. (2015), "KIBO robot demo: engaging young children in programming and engineering", *Proceedings of the 14th International Conference on Interaction Design and Children (IDC '15)*, *Boston, MA*, ACM, pp. 418-421, doi: 10.1145/2771839.2771868.

Tang, X., Yin, Y., Lin, Q., Hadad, R. and Zhai, X. (2020), "Assessing computational thinking: a systematic review of empirical studies", *Computers and Education*, Vol. 148, p. 103798.

Turkle, S. and Papert, S. (1992), "Epistemological pluralism and the revaluation of the concrete", *Journal of Mathematical Behavior*, Vol. 11 No. 1, pp. 3-33.

Vossoughi, S., Escude, M., Kitundu, W. and Espinoza, M.L. (2021), "Pedagogical 'hands and eyes': embodied learning and the genesis of ethical perception", *Anthropology and Education Quarterly*, doi: 10.1111/aeq.12382.

Wang, X.C., Choi, Y., Benson, K., Eggleston, C. and Weber, D. (2020), "Teacher's role in fostering preschoolers' computational thinking: an exploratory case study", *Early Education and Development*, Vol. 32, pp. 1, 26-48.

Whalley, P. and Barley, S.R. (1997), "Technical work in the division of labor: stalking the wily anomaly", in Barley, S.R. and Orr, J.E. (Eds), *Between Craft and Science: Technical Work in the United States*, Cornell University Press, pp. 23-52.

Willis, P. (1981), *Learning to Labour: How Working Class Kids Get Working Class Jobs*, Routledge.

Witherspoon, C.D. and Schunn, D. (2019), "Teachers' goals predict computational thinking gains in robotics", *Information and Learning Sciences*, Vol. 120 Nos 5/6, pp. 308-326.

Wolfe, S. (2006), *Your Best Year yet! a Guide to Purposeful Planning and Effective Classroom Organization*, Scholastic, New York, NY.

Wu, K.J. (2020), "Nobody sees us': testing-lab workers strain under demand", The New York Times, available at: www.nytimes.com/2020/12/03/health/coronavirus-testing-labs-workers.html?referringSource=articleShare

Wyeth, P. (2008), "How young children learn to program with sensors, action, and logic blocks", *The Journal of the Learning Sciences*, Vol. 17 No. 4, pp. 517-550.

Yu, J. and Roque, R. (2019), "A review of computational toys and kits for young children", *International Journal of Child-Computer Interaction*, Vol. 21, pp. 17-26.

Further reading

Bucciarelli, L.L. and Kuhn, S. (1997), "Engineering education and engineering practice: improving the fit", in Barley, S.R. and Orr, J.E. (Eds), *Between Craft and Science: Technical Work in US Settings*, Cornell University Press, Ithaca, New York, NY.

Jurow, A.S., Teeters, L., Shea, M.V. and Van Steenis, M. (2016), "Extending the consequentiality of 'invisible work' in the food justice movement", *Cognition and Instruction*, Vol. 34 No. 3, pp. 210-221.

K-12 Computer Science Framework Steering Committee (2016), "K-12 computer science framework", available at: https://k12cs.org/

Smith, C.P., Berland, M. and Martin, T. (2015), "Playing robot: exploring how students alternate perspectives in IPRO", in Lee, V.R. (Ed.), *Learning Technologies and the Body: Integration and Implementation in Formal and Informal Learning Environments*, Routledge, New York, NY.

Corresponding author
Deborah Silvis can be contacted at: deborah.silvis@usu.edu