TechTrends
https://doi.org/10.1007/s11528-021-00618-4

ORIGINAL PAPER

Culturally Responsive Debugging: a Method to Support Cultural Experts' Early Engagement with Code

Michael Lachney[1] · Aman Yadav[1] · Matt Drazin[1] · Madison C. Allen[1] · William Babbitt[2]

Accepted: 2 June 2021

Abstract
Despite the value that cultural experts bring to efforts to broaden the participation of racially minoritized youth in US computer science, there has been little research on supporting their knowledge of computing. This is a missed opportunity to explore the diffusion of computing knowledge across local community contexts where underrepresented youth of color spend time. To address this gap, we present one strategy for promoting cultural experts' early engagement with code, *culturally responsive debugging:* using culturally situated expertise and knowledge to debug code. We analyzed qualitative data from a professional development workshop for cultural experts to evaluate this strategy. Our findings have implications for broadening participation efforts and supporting non-programmers' knowledge of code.

Keywords Debugging · Culturally responsive computing · Computer science education · Adult education

Introduction

Among computer science (CS) education practitioners and researchers there is growing recognition that the economic discrimination and the political disenfranchisement of Black, Brown, and Indigenous communities in the United States have contributed to racialized barriers and gatekeeping in computing education and professions (Margolis et al., 2008; Eglash et al., 2017; McGee, 2020). For example, of the degree-granting institutions that responded to the Taulbee Survey in 2019, African Americans made up approximately 1% of master's degree recipients in CS (Zweben & Bizot, 2020), despite being approximately 13% of the U.S. population (United State Census Bureau, 2010). The underrepresentation of these communities of color at all levels of CS has prompted a range of broadening participation strategies to make CS more diverse and inclusive. A number of these strategies have been organized under the label of *culturally responsive computing* (CRC), an approach to technology education that builds on the programs of culturally relevant pedagogy (Ladson-Billings, 1995), culturally responsive teaching (Gay, 2018), and culturally sustaining pedagogies (Paris, 2012).

Like these programs, CRC begins by critiquing *deficit* thinking in education, which tends to frame students' families, heritages, communities, identities, and backgrounds as barriers to teaching and learning, best kept outside of academics. Alternatively, CRC seeks to make these parts of children's lives central to learning about and with technology (Eglash et al., 2013a; Eglash et al., 2013b; Scott & White, 2013; Ashcraft et al. 2017). In the context of CRC, culture can be understood in dynamic and relational terms, where technologies, material practices, and epistemic systems are co-constituted through interplays between individuals, communities, and institutions (Lachney et al., 2021a). Highlighting how technologies, practices, and knowledges are always changing helps to resist static and deterministic definitions of culture that can reproduce racist and primitivist myths by rooting Black and Indigenous peoples in the past (Vithal and Skovsmose 1997). CRC uses technologies to make connections between the past, present, and future, with explicit anti-racist and anticolonial politics that are grounded in social justice and community self-determination (Lachney et al., 2021b).

Some CRC projects build on intersectional approaches to social justice, where, for example, girls of color draw on their

✉ Michael Lachney
lachneym@msu.edu

[1] Department of Counseling, Educational Psychology, and Special Education, Michigan State University, College of Education, 620 Farm Ln. Room 513A, East Lansing, MI 48824, USA

[2] Science and Technology Studies Department, Rensselaer Polytechnic Institute, Troy, NY, USA

Published online: 20 June 2021

own experiences and backgrounds while programming educational robots to help inform local community members about political and economic issues that impact their lives (Scott et al., 2015). Other CRC projects focus on how technologies can be designed to translate between Indigenous knowledge and science, technology, engineering, and mathematics (STEM) curricula, such as making contact points between 3D visual programming software and *anishinaabe-gikendaasowin*[1] (Eglash et al., 2020a). As these cases exemplify, for the majority of CRC projects there is some element of drawing on local cultural and community knowledges (of adults and children) as part of technology design and implementation. For the purposes of this paper, we call people who have these knowledges *cultural experts*.

Collaborating with cultural experts can be a powerful way to foster deep connections between computing, culture, and community (Lachney, 2017a). What is more, there is some evidence to suggest that these collaborations might be mutually beneficial, in that the local person or group gets something of value out of learning more about technology and computing during CRC projects with teachers and researchers (Lachney et al., 2020). This presents an interesting opportunity to create multi-directional broadening participation strategies where computing, culture, and community do not just appear connected in schools or libraries but also other locally meaningful settings (Lachney & Yadav, 2020). Despite the value that local cultural experts bring to CRC projects, there has been little research or pedagogical work on supporting their knowledge of computing, though there have been some calls to conduct more research on the topic (e.g., Lachney et al., 2021a). This lack of focus is a missed opportunity to explore the diffusion of computing knowledge and technology literacies across local community contexts (beyond schools) where youth who are underrepresented in CS might spend time. This paper seeks to address this gap by presenting one strategy for supporting local cultural experts' early or introductory engagement with code. Focusing on early experiences is purposeful, as many—though not all—of the cultural experts we have collaborated with (whether youth sports coaches, urban farmers, or hair braiders) tend to be non-programmers. We call our strategy for supporting cultural experts' early introduction to code, *culturally responsive debugging* (CRD): using culturally situated expertise and knowledge to debug code.

By analyzing qualitative data from a CRC workshop with a group of cultural experts ($n = 16$), we detail the design and implementation of one CRD activity as a proof of concept. The activity was implemented with a group of cosmetologists, urban farmers, and librarians or educators who were collaborating with university researchers at a public library in Southeast Michigan. After detailing our findings we discuss answers to three questions: 1) How do we support cultural experts' knowledge of programming in ways that are motivated by and affirming of their cultural expertise? 2) How do cultural experts and educators who work with them (e.g., teachers and librarians) solve computational problems that make explicit connections between computing and cultural expertise? 3) How do cultural experts and educators who work with them talk about technology after experiencing a CRC workshop that includes a culturally responsive debugging activity? We end by explaining how our findings have implications for developing multi-directional broadening participation strategies and supporting non-programmers early engagement with code.

Literature Review

A major part of the CRC research program is identifying and building up localized assets as part of technology-oriented teaching and learning (Scott et al., 2015; Lachney, 2017a). This provides unique opportunities to not only use computing and computational thinking in the service of community-based goals and educational projects but also to diffuse computing knowledge across local contexts beyond compulsory schooling. Indeed, research suggests that computational thinking is relevant to improving young people's programming skills and everyday reasoning (Shen et al., 2020), as well as more general problem solving skills (Yadav et al., 2016; Caeli & Yadav, 2020). With this in mind, it becomes meaningful to think about computation (thinking and doing) as a means to bridge students' academic lives with their out-of-school, community, and familial lives; thus, building foundations for multi-directional strategies where broadening participation efforts are not confined to just traditional academic locations.

For example, students in one CRC after school program used computing and computational thinking to help design 3D-printed cornrow braids (Lachney, 2017b), which were then placed in a local hair salon. Lachney et al. (2019) report that the 3D-prints prompted conversations in the salon about the "algorithms" of braiding, adding to the already existing repertoire of STEM knowledge that is found in Black hair salons. In this case, the language and ideas of computational thinking became more than academic content, they also became relevant and meaningful to people's socially and culturally situated experiences. One exciting aspect of this multi-directional strategy is supporting cultural experts' knowledge of programming languages, since coding is how many young people engage with CS education. Indeed, it has been suggested that cultural experts' knowledge of programming can

[1] In *Our Knowledge is Not Primitive: Decolonizing Botanical Anishinaabe Teachings,* Wendy Makoons Geniusz (2009) uses the Anishinaabemowin term *anishinaabe-gikendaasowin* to mean the "knowledge, information, and the synthesis" of the teachings of Anishinaabeg communities (p.11). We use the term here to acknowledge Eglash et al.'s (2020) focus on Anishinaabeg architecture and design.

support CS education that is both culturally and computationally rich (Lachney & Yadav, 2020; Lachney et al., 2021a). What is more, having a multi-directional strategy for broadening participation in CS education is not only important because teachers should not be expected to do this work alone, but also because primary and secondary US schools have long histories of alienating youth of color through Eurocentric curricula and structures of assimilation (Spring, 2016; Emdin, 2016). Therefore, opportunities to represent computing knowledge in meaningful ways beyond the school walls is important if we are serious about broadening participation. Yet, many of the cultural experts we have worked with are non-programmers, with a minority being novice programmers. Therefore, it is worth looking into culturally responsive methods for supporting their knowledge of programming specifically and computing more generally.

Learning to program is a complicated task that involves various cognitive skills and conceptual representations (Rogalski & Samurçay, 1990). It is unlikely that cultural experts' involvement in a CRC project is going to result in non-programmers becoming novices, let alone experts. But CRC projects can offer an introduction to the language and artifacts of programming. Indeed, as code proliferates throughout our social and cultural lives—not always in positive ways (e.g., Noble, 2018; Benjamin, 2019; Zuboff, 2019)—there is a growing need for end-users or non-programmers to understand and comprehend code (Gross & Kelleher, 2010a). While traditional CS education tends to prioritize writing code (Lister et al., 2004; Chmiel & Loui, 2004), there is theoretical and empirical support for helping novices and non-programmers with code comprehension through reading, tracing, and debugging (e.g., Griffin, 2016).

One task, in particular, that can help people to "analyze"—i.e., break up content into smaller parts to see how they relate to one another—code is debugging (Shargabi et al., 2015, p. 34). Generally, when programmers debug something it "involves observing abnormalities in a program's behavior, finding bugs (errors), and fixing them" (Griffin, 2016, p. 148). This requires people to read the program, tracing it alongside the output, and iteratively manipulate the code for a desired result. What is more, Ahmadzadeh et al. (2005) found that while novices who were good at debugging were generally good at programming, being good at programming did not automatically make someone good at debugging. It may be that the skills needed to write and program code are not the same as those that are needed to find and fix errors (Katz & Anderson, 1987).

The trial-and-error problem solving of debugging is often frustrating (Fitzgerald et al., 2008) but it can also be an important part of the learning process (Papert, 1980). While the act of fixing the bug is the intended goal, research shows that people tend to take more time locating errors than fixing them (Fitzgerald et al., 2008). Two general methods have been identified for locating errors: forward reasoning where debuggers start with the code (this is often done when a person is debugging someone else's code) and backward reasoning where debuggers start with the output (this is often done when a person debugs their own code) (Katz & Anderson, 1987). Debugging is an important skill for anyone interacting with code. While experts and novices will obviously benefit from learning to debug, end-users and non-programmers also benefit if they work with code from someone else that they manipulate and change, often creating bugs of their own (Gross et al., 2011a).

Recognizing the affordances of debugging for learning has prompted researchers and educators to experiment with purposefully placing bugs in educational programming activities (Richards, 2000; Kafai et al., 2014a; Searle et al., 2018). The educational value of debugging is not new to the CS education community. Drawing on ethnographies of scientific laboratories (e.g., Latour & Woolgar, 1979; Traweek, 1988), Turkle and Papert (1990) likened the trial-and-error learning processes of debugging to some scientific laboratory work. Papert's (1993) theory of *constructionism* largely centers around this type of *learning by doing*. Building on the work of Papert (1980), Griffin (2016) theorizes that learning to take code apart through reading, tracing, and debugging—what Griffin calls "deconstructing code"—is a key part of understanding program behavior. These types of activities show how people can learn to locate and fix bugs through reverse engineering (Griffin et al., 2012).

While there is substantial research on novice and expert debugging (McCauley et al., 2008) there is less on non-programmers. Prompted by the proliferation of end-user code, Gross and colleagues have been exploring how to support end-users' and non-programmers' evaluation and use of code through studying their strategies and designing supportive tools (Gross & Kelleher, 2010a; Gross & Kelleher, 2010b; Gross et al., 2011b; Gross, Yang, & Kelleher 2011a). When working with unfamiliar programs, Gross and Kelleher (2010a) explain that finding code that is responsible for a specific behavioral output can be a major challenge for end-users and non-programmers. They suggest including meta-information and supports for users to connect code to observable outputs can help to overcome this challenge. They also found that non-programmers use similar navigation and search strategies as experts and novices when reading, tracing, and debugging code: forward reasoning, backward reasoning, and working through the code line-by-line (Gross & Kelleher, 2010a).

Building on the claims that debugging can support users' familiarity with code (Richards, 2000) but that non-programmers need support in connecting code to output behavior (Gross & Kelleher, 2010a), this paper proposes the method of *culturally responsive debugging* (CRD) to support cultural experts' familiarity and early engagement with code.

The process of supporting and designing CRD activities begins by first identifying localized knowledge that is embedded in cultural and community practices. Second, computational hardware and/or software are used to represent that knowledge in ways that are respectful, meaningful, and authentic to cultural experts. Next, a bug is placed in the code of the software that changes the output of the program in a way that is counterintuitive to what the cultural experts know to be correct. Finally, they are then prompted to use their own culturally situated knowledge to debug the program. Below we describe the design, implementation, and evaluation of one CRD activity with cultural experts at a CRC professional development workshop.

Context and Methods

Background

This research on CRD is situated within a two-year CRC project that included highlighting the localized and cultural uses of pH knowledge by cosmetologists, natural hair experts, and urban farmers via do-it-yourself computing technologies. While mainstream broadening participation discourses often assume (explicitly or implicitly) that underrepresented communities lack STEM knowledge or expertise and therefore—in standard deficit fashion—it must be brought in from the outside, CRC works with the existing knowledge systems that are already embedded within community contexts. Indeed, these existing knowledge systems are assets for broadening participation efforts. Our choice to focus on pH emerged from a collaboration with a high school cosmetology teacher and a group of high school girls who were hired by our research team to help develop computationally and culturally rich STEM lessons for high school cosmetology courses. People connected to the high school cosmetology program (including the teacher) and the girls identified pH as a key knowledge-base for anyone working with hair and/or chemicals in the salon. A summer professional development workshop in 2017—which brought together teachers, cultural experts, school staff, researchers, students, and technologists—is where the first iteration of the pH Empowered lesson was formally designed.

The lesson used an Atlas Scientific pH sensor that could be programmed with an Arduino micro-controller to first test the pH levels of off-the-shelf cosmetic products and then of natural products that students made (Fig. 1). Students would build the sensor, upload a program to the Arduino, and calibrate the sensor before generating hypotheses about the pH of cosmetic products and measuring the pH of the products to test their hypotheses. While this lesson was first used in a high school cosmetology program, it was later used in out-of-school STEM programs (e.g., in libraries) and as part of professional development for cosmetologists interested in supporting CRC programs and collaborating with STEM teachers (Fig. 2).

Fig. 1 To an audience of peers and teachers, a student explains her own natural cosmetic product, with a pH sensor and computer in the foreground

However, one of the major limitations of the lesson was the relatively shallow engagement with the Arduino code for the pH sensor itself. The lesson called for uploading the code and calibrating the sensor, but this only required inputting numerical values into the serial monitor, it did not call for reading or manipulating the code. When confronted with these challenges while preparing for a CRC professional development workshop with the pH sensor for cosmetologists, urban farmers, and librarians, the first three authors on this paper

Fig. 2 A group of cosmetologists build pH sensors at a 2018 "Cos-computing" (cosmetology + computing) professional development workshop

explored ways to motivate workshop participants'—who we assumed would be mostly non-programmers—engagement with the code. The challenge was how to motivate engagement in a way that authentically connected to the workshop participants' own knowledge of pH. We came up with the idea of purposefully placing a logic bug into the program that would make the pH reading appear obviously off—and therefore recognizable to people with knowledge of pH—after the code was uploaded. We wanted to make sure that the bug could be fixed in multiple ways, but with little or no prior programming knowledge. We present data from the design and implementation of the coding activity below.

Workshop and Participants

The workshop where we first introduced this CRD activity took place at a public library in Southeast Michigan and was organized in collaboration with a group of three librarians. The workshop took place in a small city of about 20,000 people. People of color make up approximately 63% of the population, and those under 18 years old make up approximately 13.5% of the population (United State Census Bureau, 2010). The present study took place in the city's downtown library, specifically in a teen space where a youth version of the workshop would be held the following summer. This area is generally designed to be a safe space for teens to study, play, socialize, learn, and create. Indeed, part of the library system's mission is to create youth-driven opportunities to connect to the broader community, develop leadership skills, and have a voice at the library.

In collaboration with the librarians, we decided to focus on recruiting local cosmetologists and urban farmers since both use pH as part of their professional practices. The workshop had three purposes: 1) to receive feedback from cultural experts for future iterations of the pH lesson; 2) begin forming relationships with cultural experts who might help implement the pH lesson at the library that following summer; and 3) implement and evaluate the CRD activity. The three librarians who helped to organize the workshop used their connections to the local community—including connections to prominent urban farms, beauty salons, and braiding shops—to recruit participants.

The workshop was divided up into four different parts. Workshop participants largely worked in pairs, but were also given the opportunity to work independently. The first part of the workshop was an introduction to the pH Empowered lesson and how cosmetologists and urban farmers can leverage their knowledge to support its implementation. In the second part of the workshop participants built, calibrated, and debugged the Arduino-based pH sensor. During the third part of the workshop participants made pH "alarms" with LEDs or speakers, which would light up or play music, respectively, when the pH sensor reached a certain threshold. Finally, there was a share-out section where everyone reflected on the workshop, offering feedback on what went well and what could have been improved.

Because this paper is primarily focused on the debugging activity the following is a more detailed explanation of the pH sensor code and the bug. We placed a logic error within the code for participants to find and correct that intentionally drew on what we assumed were cosmetologists' and urban farmers' knowledge of pH. Logic errors differ from syntax errors in that the error allows for "compilation and running but lead[s] to incorrect results" (McCauley et al., 2008, p. 68). Correcting the error would support backward reasoning, where users start with the output behavior of the program first and then move into the code from there to debug. While non-programmers can be intimidated by code and have trouble connecting the behavioral output to sections of code (Gross & Kelleher, 2010a), we hypothesized that making the logic error relevant to the workshop participants' prior knowledge of pH would provide sufficient scaffolding for completing the task.

The section of code with the error contained nested "if" statements for displaying whether the input from the sensor registered a high pH or a low pH. Without the error, if pH was greater or equal to 7.0, "high" printed to the serial monitor. If pH was less than 6.999, "low" printed. The bug we created caused readings below a pH of 6.999 to print the word "high" and readings above or equal to a pH of 7.0 to print the word "low" (Fig. 3). This code, like all Arduino code, has two basic functions. Functions are areas of code that are organized by action in such a way that they might be repeated. The two functions in all Arduino programs are the "setup" function and the "loop" function. The setup function runs whenever the Arduino hardware is turned on or restarted—it creates the structure and initializes values for the rest of the program. The loop function runs immediately after the setup function and repeats itself until the hardware is disconnected from a power source. The code used for this project utilized these two functions. The setup function initialized variables and set output conditions for the serial monitor (the readable display). The loop function continuously read information from the

```
if (sensor_string_complete == true) {
  Serial.println(sensorstring);

  if (isdigit(sensorstring[0])) {
    pH = sensorstring.toFloat();

    if (pH >= 7.0) {
      Serial.println("low");
    }
    if (pH < 6.999) {
      Serial.println("high");
    }
  }
```

Fig. 3 The "logic error" in the pH sensor code

probe and for each reading it conditionally displays information from the probe to the serial monitor.

Sixteen adult participants signed up for and attended the workshop. This included seven cosmetologists, all who identified as Black/African American; six librarians or teachers, three who identified as White, one as Black/African American, and two as multiracial; and three urban farmers, two White and one Black/African American. Of the sixteen adults, thirteen were women and three were men (one man in the cosmetology group and two men in the urban farming group). The ages of the workshop participants ranged from seventeen to sixty-four, but most of the participants were over the age of thirty-five. In addition to collecting demographic information, we also asked participants about their programming experiences. Of the fifteen who responded, seven of the participants indicated that they had some form of programming experience, while eight indicated that they had no prior programming experience. Of the seven who did have experience, we followed up with an open-ended survey question to discover that two had brief exposure to an introductory programming language, one misinterpreted what we meant by programming, and four had more in-depth experiences through the likes of formalized coursework. We categorized the four participants with more in-depth experience as novices, while the rest of the workshop participants were categorized as non-programmers.

Research Questions

To evaluate and explore the pH sensor CRD activity we pose three questions to guide our analysis and findings. We list each question below along with it rationale.

(a) *Given that many cultural experts who work on CRC projects are non-programmers, how do we support their knowledge of programming in ways that are motivated by and affirming of their cultural expertise?* Here we aim to put our hypothesis to the test: connecting the logic error to workshop participants' knowledge of pH will provide sufficient scaffolding for completing the CRD task. But, this is not just about completing a task. CRD does not treat cultural experts' knowledge as sugar coating or surface gloss for learning to code, but, instead, is about collaborative relationships that aim to respectfully and meaningfully represent cultural knowledge with computing. The goal is for culture to motivate computational engagement in ways that are respectful and affirming of the depth of cultural and intercultural knowledge systems.

(b) *How do cultural experts and educators who work with them (*e.g.*, teachers and librarians) solve computational problems that make explicit connections between computing and cultural expertise?* The purpose of this question is to explore the problem-solving processes that cultural experts and educators use when confronted with the CRD activity. We hope that this provides insight into improving both the theory and practices of CRD.

(c) *How do cultural experts and educators who work with them talk about technology after experiencing a CRC workshop that includes a culturally responsive debugging activity?* Given that programming was new to many of the workshop participants, we are interested in exploring if and how they think their ideas about technology changed. We are interested in reflections that might be about coding specifically, but also about technology generally since opening up the black box of computing may shape how people think about their own relationships to technological devices and systems.

To answer these three questions, we analyzed four types of data: video data of the debugging activity, observational notes from workshop facilitators, pictures from the workshop, and focus group interview data from after the workshop. To answer questions one and two we relied on video data, pictures, and observational notes that were collected during the workshop. For the third question we analyzed data from focus group interviews that took place after the workshop.

Data Collection and Analysis

Video data of the debugging activity were collected in two ways. First, video was taken of participants building, calibrating, and debugging their pH sensors as they worked in groups or individually. Second, video was captured from participants' computer screens as they worked in the Arduino integrated development environment to calibrate and debug the pH sensor. Drawing on video analysis methods from qualitative CS education research (Kafai et al., 2014b; Searle & Kafai, 2015; Tenenberg, 2019), we created a video log of both the people and screen captures side-by-side, showing a minute-by-minute breakdown of individuals' and groups' activities. This resulted in an analysis of approximately 220 minutes of video of 10/16 participants. One participant did not show up in time for the debugging activity and another five workshop participants had either malfunctions with their computer cameras or their cameras were turned off.

A two-step qualitative analysis was conducted of the video log. During the first step, notes were made by the first and last authors of this paper about when, where, and how participants first identified the existence of the bug, how long it took them to search the code, and how long it took them to fix the bug. During the second step, findings were compared and any disparities in interpretations were debated and discussed. Pictures, observational notes, and video from the workshop were triangulated—combining different data sources from different people and/or different methods during analysis—with the video log to construct trustworthiness and validity.

After the workshop, three focus-group interviews were conducted to collect workshop participants' ($n = 16$) self-reports about any changes in their perceptions or attitudes toward technology. The groups for each of these interviews were demarcated by professional expertise: a group of cosmetologists, a group of urban farmers, and a group of librarians or educators. They were semi-structured, with each lasting approximately 35–50 minutes. Reflecting on their uses of technology during the workshop was only part of the focus group interview, other topics, reported elsewhere, included their STEM expertise and the role of the concept of *race* in broadening participation efforts (Lachney et al., 2021b).

Analysis of the focus group interviews was completed by the first and last authors as part of a larger "provisional coding" process, which consists of an a priori list of codes that were determined based on previous research (i.e., the work with cosmetologists leading up to the creation of the debugging activity) (Saldaña, 2016, 297). Content dealing with technology, computing, and debugging were aggregated under the code of "technology" for each of the three focus groups. The first and last authors then engaged in intersubjective dialogue—"agreement through a rational discourse and reciprocal criticism between those interpreting a phenomenon" (Brinkmann & Kvale, 2015, p. 279)—to come to an agreed interpretation of participants' self-reports.

Findings

Culturally Responsive Debugging in Action

As a way to explore computational problem solving among workshop participants, Table 1 is a breakdown of the expertise, programming experience, and time that it took to complete the debugging activity for the six groups or individuals whose videos were analyzed. The "approx. time to find bug" indicates the time from when they entered the Arduino programming environment to when they found the bug in the code. "Approx. time to fix bug once found" indicates the time from when they identified the bug in the code to when they uploaded a correctly fixed program. For some additional context, all paired groups and individuals were able to identify and fix the logic error that we placed in the code. When prompted to look for a problem within their programs, all groups and individuals, except the solo librarian (#6), used the serial monitor to identify the mix-up between "high" and "low." The librarian did not find the problem until she was prompted that there was a bug, and she was already looking through the code. Of those who identified the problem within the serial monitor, everyone independently went from the serial monitor to the programming environment to debug, except for the group of cosmetologists (#5). Group #5 at first thought that the bug could be fixed by recalibrating the pH probe and did not go to the code until after a facilitator explained that the bug would be found there.

In line with some prior literature on debugging (e.g., Fitzgerald et al., 2008), group or individual #1, #4, and #6 took more time to locate the error than to fix it. Group #5 took a notable amount of time figuring out where they could fix the error but once they were in the programming environment they quickly skimmed it for signs of pH and fixed it by correcting the print text. Groups #2 and #3 found the bugs quicker than most of the other groups or individuals but took longer to fix it. While taking longer to fix the error might not look good from a traditional standard of code efficiency, from a learning perspective taking longer to fix the bug represented deeper and more active engagement with the pH sensor code.

To further understand workshop participants' engagement with code, Table 2 represents the different strategies that groups and individuals used to find and fix the bug. We demarcated the groups and individuals by paired or not paired and then by programming background. We then only included shared strategies. This was appropriate for the most part, but it also meant that we did not represent data on group #2, who added to the code in their attempt to fix the bug. Every group or individual talked aloud to some extent while locating and

Table 1 Breakdown of each individual's or group's expertise, programming experience, and approximate time to find and fix the error ($n = 10$)

Group or individual	Number of Participants	Expertise	Non-programmer -or- Novice	Approx. time to find bug	Approx. time to fix bug once found
1	2	a. Teacher b. Librarian	a. Non-programmer b. Non-programmer	3 min	2 min
2	2	a. Librarian b. Cosmetologist	a. Novice b. Non-programmer	2 min	15 min
3	2	a. Farmer b. Farmer	a. Novice b. Non-programmer	2 min	5 min
4	1	a. Cosmetologist	a. Non-programmer	4 min	2 min
5	2	a. Cosmetologist b. Cosmetologist	a. Non-programmer b. Non-programmer	1 min	1 min
6	1	a. Librarian	a. Non-programmer	4 min	2 min

Table 2 Workshop participants' strategies for locating and fixing the bug (n = 10)

Paired Groups or Individuals	Bug Location Strategies	Debugging Strategies
Paired Novice Programmers and Non-programmers (#2 & #3)	- Work through code from top-down - Read parts of the code aloud - Talk aloud to each other - Talk about computer specific terms and ideas	- Talk aloud to each other - Change syntax - Change print text - Discuss and/or try to create a "mid" case
Paired Non-programmers (#1 & #5)	- Work through code from top-down - Read parts of the code aloud - Talk aloud to each other	- Talk aloud to each other - Change print text
Individual Non-programmers (#4 & #6)	- Works through code from top-down - Talks aloud to self and/or others	- Talks aloud to self and/or others - Changes print text

fixing the bug and every group or individual started from the top and worked their way down when trying to locate the bug. But the groups with novices were the only ones to discuss computer specific terms while locating the error, including asking questions about strings and explaining syntax (e.g., brackets). This makes sense given their prior knowledge and experiences. In addition, the time these groups spent fixing the bug provided them with more opportunities to explore different strategies than the non-programmers.

Indeed, each of the groups or individuals with all non-programmers simply found the error and changed the printed text, swapping "high" and "low." In groups with novice programmers (#3 and #2), there was a strong focus on the lack of the middle or "mid" case in the code for when the pH sensor would read exactly 7.0, which, according to any cosmetology or chemistry textbook, would be "neutral." In addition, calibrating the sensor required inputting a value for "mid" into the serial monitor to account for a neutral level, but because the sensor rarely ever reads exactly 7.0 it was not part of the code. The additional time it took to fix the code in both of these groups was due to discussing if they should add the additional code for a "mid" case, how to go about doing so, and making adjustments in the code to test out their ideas. The librarian and cosmetologist group (#2) did add the case, altering the existing code for the high reading to be suitable for exactly equal to 7.0, and then adding a third case for "high" (greater than 7.0). While doing this they created more errors in the process, which prompted them to explore more of the code. The farmer group (#3) decided to forego the addition of the case, deleted the equal sign in the greater than if statement, preferring that it not read anything for 7.0.

After the debugging activity, we found that some groups and individuals continued to modify and explore the code by drawing on their knowledge of pH to inform their decisions and inquiries. The two urban farmers (#3) continued to think about what they would need to change in the code to add a "mid" case. While manipulating some of the code, they wondered if such a case would ever be meaningful in the context of an urban farm. The group of cosmetologists (#5) drew on their general and profession-specific knowledge of pH, changing "low" and "high" to "battery acid" and "lye." Lye is high on the alkaline side of the pH scale and can be found in products such as relaxers. These workshop participants' continued explorations of pH through computing helped to reinforce the goal of the CRD activity: making programming meaningful and relevant to cultural experts' own knowledge and contexts.

Post-Workshop Reflections on Technology

In general, participants' attitudes toward the workshop were affirming and positive. In the library group, there was a sense that the workshop made programming, as one participant put it, "accessible" and "low stakes." The librarian who worked alone on the debugging activity (#6) was particularly vocal about what this exposure to code meant for her personally. She has a brother who often works with programming languages but expressed that it always seemed too "intimidating" for her personally. While she said that this feeling lingered at the beginning of the workshop, once she got started working with the sensor and code she recalled a shift in her attitude: "That's what I learned new is how to actually, like, read through code and figure out, like, what some of the terms mean now, as opposed to be[ing] like, 'okay that's cool, yeah sure, all right'." Another librarian expressed that it was helpful to debug in pairs:

> *I liked it, how it was kind of like a puzzle that you needed to figure out and it was very nice that I had a partner to help me figure it out because when I was reading it as "high" when it was like 3.1, I thought that meant high acidity and I was like 'Oh that's good. It's reading as high acidity" and then she was like no, that means like high on the scale. I was like "Oh yeah then that is wrong."*

Here we can see that prior knowledge of pH motivated recognizing the error and prompted this group to try and solve the debugging "puzzle." In addition, the workshop prompted

librarians to consider not only supporting young people's knowledge of code but also adults who might not have been exposed to code when in school: "...with the coding, how to incorporate it also into the library system, it's not just, I mean we always focus on the young people, but it's not just young people, it's a lot of our, you know, older adults who would like to do something like that." Thinking about the library as a space for adult programming education was an exciting suggestion given that one of the goals of the workshop was to explore methods and strategies for supporting cultural experts' programming knowledge.

The urban farmers also expressed enthusiasm for the debugging activity and workshop in general. The non-programmer in group #3 explained her experience:

> *I think maybe one thing that's different now than earlier is, before, I knew there was all this open-source code and plans... and all this stuff available for us to use. But I've never physically done anything before. So, while I was like, really, I'm really happy that it exists among the farming community, now I feel like, "Oh actually this is something that maybe I could do."*

This attitude is different from her more general stance that technology does not often work in favor of small farming operations:

> *I feel like using computer programming and technology… we're at a crossroads in the farming world where there is a shit ton of money being put towards high tech agriculture. That completely takes the soul out of agriculture and produces food that doesn't have micronutrients in it and puts people out of work...*

The urban farmers tended to express general skepticism of any large-scale farming or technology operation. But it did seem that their convivial experiences with the pH probe and Arduino code helped to connect the "soul" of farming to technology in a meaningful way. The novice urban farmer recalled their discussions and explorations of the "mid" case, which they continued to explore after they had fixed the bug:

> *So, we altered it [the code] to do acidic, alkaline, and neutral, which was a learning curve just to remember the code and everything. So, I think understand[ing], making sure that I know why I'm putting things where they are, and understanding why I'm putting that code in there is the key for me teaching somebody.*

Here we can see that the workshop helped this farmer think about what it would mean to teach programming in his own context, which included youth outreach programs.

After the workshop, the cosmetologists focused less on the specifics of the CRD activity and more so on the potential of integrating technologies that they already knew about into their practices, particularly to support their clients' health. One cosmetologists discussed how technology could help them understand pH in more "precise" ways. Another cosmetologist discussed other applications of technology that she had seen and would like to use in her practices, such as for a "scalp analysis": "You put the data in, you get [a] printout for the client to see and you to see. You all huddle together and come up with a resolution to cure whatever." The cosmetologists articulated that technology applications had the potential to support their practices but also expressed uncertainty in terms of their own knowledge and access. As one cosmetologist who had helped to facilitate the workshop put it, "It's our job to try to start educating our clients more… as far as the possibilities of having healthy hair. For me… that's basically what it is. The technology is out there, it's just trying to find where it is." While there are certainly benefits to cultural experts having computing knowledge to help support young people's interest in CS, this quote opens up the door for thinking about how CRC might help cultural experts access and use technology to innovate their own practices.

Discussion

Despite the growing importance of cultural and community expertise in constructing equitable CS education, there is a dearth of literature on how to support cultural experts' knowledge of CS generally and programming specifically. This is unfortunate because there is evidence to suggest that cultural experts can help to make deep culture-computing connections that challenge traditional assumptions about whose knowledge is relevant to CS and programming (Lachney & Yadav, 2020; Lachney et al., 2021a). This also provides opportunities for cultural experts to bring knowledge of culture-computing connections back into their own locally meaningful contexts (Lachney et al., 2019), thus helping to build a multi-directional foundation for broadening participation. In response, this paper has sought to introduce the idea of *culturally responsive debugging* and provide a proof of concept through one CRC professional development workshop with cosmetologists, librarians, and urban farmers.

How Do we Support Cultural Experts' Knowledge of Programming in Ways that Are Motivated by and Affirming of their Cultural Expertise?

We have proposed one method for supporting cultural experts' engagement with code called CRD: using culturally situated expertise to debug code. In our case, groups and individuals used their culturally situated knowledge—

from urban farming or cosmetology—of pH to identify a problem with the output of the program and then, mostly, engaged in backward reasoning to connect the behavior to specific lines of code. We were intentional in our choice to make the error explicitly relevant to the content of pH because non-programmers tend to have trouble identifying connections between behavioral output and code (Gross & Kelleher, 2010a).

While we spent a notable amount of space examining one case of this method, the process of coming up with the pH activity also required sustained temporal engagement with cultural experts themselves. Before the first, second, and third authors came up with the debugging activity, the first and last authors spent multiple years collaborating with cosmetologists to understand how their expertise intersected with STEM topics and how to represent that expertise in computing education. Elsewhere we have reported on how to design technologies and technology experiences that bridge culture and computing, as well as our approaches to collaboration and trust-building (e.g., Lachney et al., 2019; Lachney et al., 2020; Lachney et al., 2021b), but we have not studied our collaborators' own experiences or engagement with code.

Our findings indicate that when designing paired programming activities that are meant to foster early engagement with code, we should not only focus on disciplinary expertise but also pairing people up with different levels of programming experience. While all of the groups or individuals in our study fixed the bug, groups that had novice programmers working alongside non-programmers took more time to engage with the code, exploring creative solutions for fixing the bug. Indeed, both groups with novices focused on fixing the bug by discussing and/or programming a "mid" case to account for a neutral pH level. While this did not fit the traditional standards of coding efficiency—indeed, group #2 created more bugs during their debugging process—it did support trial-and-error learning, which is a major educational affordance of computing (Papert, 1980). Therefore, when developing CRC programs that support cultural experts' computing knowledge it is worth considering how to bring people together with not only cultural and community expertise but also different programming backgrounds. It may be that the novice programmers, having worked with code before, felt more comfortable with treating debugging as an open problem, whereas the non-programmers were still figuring out how to engage with code in more open-ended ways.

For groups with novices and groups with just non-programmers, framing the debugging activity around pH helped them connect computing to their professional and cultural expertise. After fixing the bug, the group of farmers and the group of cosmetologists drew on context-specific knowledge of pH to motivate discussing and/or manipulating the code. Indeed, discussing how the code might be changed to fit the urban farm context or representing more alkaline numbers along the pH scale as "lye" is, in a sense, providing insights into how to deepen the cultural responsiveness of the pH lesson by offering more bridges between community, cultural expertise, and computing.

How Do Cultural Experts and Educators Who Work with them (e.g., Teachers and Librarians) Solve Computational Problems that Make Explicit Connections between Computing and Cultural Expertise?

As we saw from the workshop participants, the error that we created could easily be fixed without much knowledge of syntax or programming more generally. As we have explained, we find that the majority of cultural experts who we work with are non-programmers but, nonetheless, introducing computing and programming to cultural experts is an important part of our CRC work because it might help to foster a more dynamic community of stakeholders for broadening participation efforts. Therefore, the bug was designed to support introductory engagement with code that was motivated and contextualized by cosmetologists' and farmers' knowledge of pH. This helped to resist the idea of engaging with code for coding's sake, which, arguably, just reproduces the status quo (Lachney & Yadav, 2020).

We felt that the connection was well made, as post-workshop discussions indicated that the participants were able to use the activities to think about computing and technology in their own professional and cultural contexts. More specifically, we found that to motivate making this connection we needed for workshop participants to identify a bug by seeing a mismatch between what they know about pH and the behavior of the program. They were then prompted to use backward reasoning to locate and fix the bug so that the output matched their existing knowledge. This is similar to learning processes that many educational programming environments (e.g., Logo, Scratch, Snap!, etc.) help to facilitate: trial-and-error problem solving based on a desired visual output, which is immediately provided when the code is run. We found that the majority of individuals or groups were able to engage with this backward reasoning, though it was difficult for the group of cosmetologists (#5) and, also, the solo librarian (#6) was already in the programming environment when she learned about the bug.

Once the workshop participants were in the code, regardless of if they were working individually or in a group, they all found the bug by going through the code from the top-down and talked through the code aloud, actually reading it and/or talking about it to people around them (paired partners, other individuals or groups, and/or facilitators). In addition, the groups with novices uniquely discussed programming specific terms and ideas with their partners while trying to locate the bug. Similar to previous research on the debugging practices

of novices (e.g., Fitzgerald et al., 2008), the non-programmers took more time finding the bug than fixing it. Yet, we found that groups with novices did not match this prior research, as they took more time fixing the bug than finding it. It may be that since they were in groups with non-programmers they took on a more mentoring role, but it is hard to say with our limited data, which does not include groups of just novices. To fix the bugs the non-programming groups and individuals switched around the printed text, a path of least resistance to obtaining the desired output. The groups with novices changed the syntax and considered adding or actually added additional lines of code to try and fix the bug.

How Do Cultural Experts and Educators Who Work with them Talk about Technology after Experiencing a CRC Workshop that Includes a Culturally Responsive Debugging Activity?

After the workshop, there were generally positive attitudes among the participants toward the technology used in the workshop and about broadening participation efforts. Many expressed increased confidence in engaging with computing. In addition, each of the post-workshop discussion groups spoke about how they might integrate technology, if not programming specifically, into their professional practices. The librarians considered extending their computer coding programs to adults, the farmers thought about what it would be like to teach coding to youth at their outreach programs on the farm, and the cosmetologists considered what technologies could fit into their salon practices. Therefore, another way to support cultural experts' programming knowledge is to not only introduce computing through relevant content (e.g., pH) but also then support explorations of how that knowledge can be used in their own practices. There has been increased interest among CRC researchers to not only support education but also wealth generation in the local economy (e.g., Eglash et al., 2020b; Robinson et al., 2020). This type of mutually beneficial multi-directionality—where technologies and computing education work in service of both traditional academic knowledge and local sources of wealth generation (e.g., a braiding shop, hair salon, urban farm, artisan textile studio)—is worthy of more research.

Limitations

The logic bug that we purposefully placed within the pH sensor code was relatively simple and could be fixed without changing any syntax or having more general knowledge of programming. This was largely because we had assumed that most of the workshop participants would be non-programmers—many of the cultural experts we work with are—and we wanted them to be able to fix the bug with their knowledge of pH. While our research provides a proof of concept, we are limited in our ability to draw stronger conclusions about if our CRD method would work with more advanced problems. For example, maybe the output behavior only includes a print of "high" for the whole pH scale; indicating the need to add another case for "low," if not "mid" as well. We were also limited in our sample size and group make-ups. We could have made stronger conclusions about the length of time it took groups with novices to fix code if we could have had groups consisting of only novice programmers.

Future Work

While our study provides a proof of concept for CRD, whether this method could be used for sustained engagement with code is hard to tell. What would a CRD activity look like when designed for novice programmers or expert programmers? Furthermore, what does CRD look like in professional settings? For example, members of the Culturally Situated Design Tool research team—who design programming environments to explore the mathematics embedded in Indigenous and vernacular designs (Eglash et al., 2006; Babbitt et al., 2012; Bennett, 2016)—must often make negotiations between the fidelity of cultural designs and the constraints of the software or hardware that they are using to simulate and represent the designs (Lachney, 2017a). What types of CRD practices might these technologists engage in with cultural experts? Future work might, therefore, not only consider CRD as an educational method for supporting broadening participation efforts, but also a method for working with artisans and other cultural experts in the design of technologies that represent their knowledges; in other words, CRD might be a domain-specific form of computational thinking and problem solving (Lachney et al., in press).

Conclusion

Broadening the participation of underrepresented, minoritized youth in CS should not be done by schools alone. Indeed, multiple teams of researchers and educators have made the case that "it takes a village" to support equitable CS education (Ryoo et al., 2015; Lachney et al., 2021a). To take this call seriously, we have been exploring multi-directional strategies for broadening participation, which do not only include enrolling traditional school employees and staff in CS education but also local people with unique cultural and community expertise who shape young people's lives and experiences out-of-school. Doing so may support local cultural experts' knowledge of CS and diffuse CS ideas across culturally meaningful locations (Lachney et al., 2019).

However, many of the cultural experts who we have worked with are non-programmers. Therefore, we have sought to introduce CRD as a method for supporting non-

programmers' early and meaningful engagement with code. In providing a proof of concept for CRD, we found that it generally supported positive attitudes toward technology among workshop participants and helped them connect technology to their own contexts. We also found that supporting deeper engagement with code for non-programmers may be helped by pairing them up with novice programmers. While this paper provided a relatively simple debugging challenge, future work can explore if CRD might also be appropriate for the education of novices and in supporting technologists' more nuanced engagement with code and culture.

Acknowledgements We would like to thank Science + Society @ State and the John and Lucy Bates-Byers Educational Technology Endowment for their support of this work.

Declarations

Ethical Approval All procedures in this study that involved human subject research were approved by Michigan State University's Institutional Review Board and conducted in accordance with the 1964 Helsinki declaration and its later amendments or comparable standards.

Informed Consent Informed consent was obtained from every individual who participated in this study.

Conflict of Interest The authors declare no conflicts of interest.

References

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005, June). An analysis of patterns of debugging among novice computer science students. In proceedings of the 10th annual SIGCSE conference on innovation and technology in computer science education (pp. 84-88).

Ashcraft, C., Eger, E. K., & Scott, K. A. (2017). Becoming technosocial change agents: Intersectionality and culturally responsive pedagogies as vital resources for increasing girls' participation in computing. *Anthropology & Education Quarterly, 48*(3), 233–251.

Benjamin, R. (2019). *Race after technology: Abolitionist tools for the new Jim code*. John Wiley & Sons.

Babbitt, B., Lyles, D., & Eglash, R. (2012). From Ethnomathematics to Ethnocomputing: Indigenous algorithms in Traditional Context & Contemporary Simulation. In S. Mukhopadhyay & W-M. Roth (Eds)., *Alternative forms of knowing (in) mathematics* (pp. 205–219). Brill Sense.

Bennett, A. G. (2016). Ethnocomputational creativity in STEAM education: A cultural framework for generative justice. *Teknokultura, 13*(2), 587–612. https://doi.org/10.5209/rev_TEKN.2016.v13.n2.52843.

Brinkmann, S., & Kvale, S. (2015). *Interviews: Learning the craft of qualitative research interviewing* (third edition / Svend Brinkmann, Steinar Kvale). Thousand Oaks, CA: Sage.

Caeli, E. N., & Yadav, A. (2020). Unplugged approaches to computational thinking: A historical perspective. *TechTrends, 64*(1), 29–36.

Chmiel, R., & Loui, M. C. (2004). Debugging: From novice to expert. *ACM SIGCSE Bulletin, 36*(1), 17–21.

Eglash, R., Babbitt, W., Bennett, A., Bennett, K., Callahan, B., Davis, J., et al. (2017). Culturally situated design tools: Generative justice as a foundation for STEM diversity. In P. Tripathi & Y. Rankin (Eds.), *Moving students of color from consumers to producers of technology* (pp. 132–151). IGI Global.

Eglash, R., Gilbert, J. E., & Foster, E. (2013a). Toward culturally responsive computing education. *Communications of the ACM, 56*(7), 33. https://doi.org/10.1145/2483852.2483864.

Eglash, R., Gilbert, J. E., Taylor, V., & Geier, S. R. (2013b). Culturally responsive computing in urban, after-school contexts: Two approaches. *Urban Education, 48*(5), 629–656.

Eglash, R., Lachney, M., Babbitt, W., Bennett, A., Reinhardt, M., & Davis, J. (2020a). Decolonizing education with Anishinaabe arcs: Generative STEM as a path to indigenous futurity. *Educational Technology Research and Development, 68*(3), 1569–1593.

Eglash, R., Bennett, A., O'donnell, C., Jennings, S., & Cintorino, M. (2006). Culturally situated design tools: Ethnocomputing from field site to classroom. *American anthropologist, 108*(2), 347–362.

Eglash, R., Robert, L., Bennett, A., Robinson, K. P., Lachney, M., & Babbitt, W. (2020b). Automation for the artisanal economy: Enhancing the economic and environmental sustainability of crafting professions with human-machine collaboration. *AI & SOCIETY*, 1–15.

Emdin, C. (2016). *For White Folks Who Teach in the Hood... and the Rest of Y'all Too: Reality Pedagogy and Urban Education*. Beacon Press.

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: Finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education, 18*(2), 93–116.

Gay, G. (2018). *Culturally responsive teaching: Theory, research, and practice*. Teachers College Press.

Geniusz, W. M. (2009). *Our knowledge is not primitive: Decolonizing botanical Anishinaabe teachings*. Syracuse University Press.

Griffin, J. M. (2016). Learning by taking apart: deconstructing code by reading, tracing, and debugging. In Proceedings of the 17th Annual Conference on Information Technology Education (pp. 148–153).

Griffin, J., Kaplan, E., & Burke, Q. (2012). Debug'ems and other deconstruction kits for STEM learning. In IEEE 2nd integrated STEM education conference (pp. 1-4). IEEE.

Gross, P., & Kelleher, C. (2010a). Non-programmers identifying functionality in unfamiliar code: Strategies and barriers. *Journal of Visual Languages & Computing, 21*(5), 263–276.

Gross, P., & Kelleher, C. (2010b). Toward transforming freely available source code into usable learning materials for end-users. In *Evaluation and usability of programming languages and tools* (pp. 1-6).

Gross, P., Yang, J., & Kelleher, C. (2011a, May). Dinah: An interface to assist non-programmers with selecting program code causing graphical output. In proceedings of the SIGCHI conference on human factors in computing systems (pp. 3397-3400).

Gross, P., Kelleher, C., & Yang, J. (2011b, September). An investigation of non-programmers' performance with tools to support output localization. In 2011 IEEE symposium on visual languages and human-centric computing (VL/HCC) (pp. 55-58). IEEE.

Kafai, Y. B., Lee, E., Searle, K., Fields, D., Kaplan, E., & Lui, D. (2014a). A crafts-oriented approach to computing in high school: Introducing computational concepts, practices, and perspectives with electronic textiles. *ACM Transactions on Computing Education (TOCE), 14*(1), 1–20.

Kafai, Y., Searle, K., Martinez, & Brayboy, B. (2014b). Ethnocomputing with electronic textiles: Culturally responsive open design to broaden participation in computing in American Indian youth and communities. In proceedings of the 45th ACM technical symposium on computer science education (pp. 241–246). ACM. https://doi.org/10.1145/2538862.2538903.

Katz, I. R., & Anderson, J. R. (1987). Debugging: An analysis of bug-location strategies. *Human-Computer Interaction, 3*(4), 351–399.

Lachney, M. (2017a). Culturally responsive computing as brokerage: Toward asset building with education-based social movements. *Learning, Media and Technology, 42*(4), 420–439. https://doi.org/10.1080/17439884.2016.1211679.

Lachney, M. (2017b). Computational communities: African-American cultural capital in computer science education. *Computer Science Education, 27*(3–4), 175–196. https://doi.org/10.1080/08993408.2018.1429062.

Lachney, M., Babbitt, W., Bennett, A., & Eglash, R. (2019). Generative computing: African-American cosmetology as a link between computing education and community wealth. *Interactive Learning Environments*, 1–21. https://doi.org/10.1080/10494820.2019.1636087.

Lachney, M., Babbitt, W., Bennett, A., & Eglash, R. (2020). "A voice to talk about it": Cosmetologists as STEM experts in educational technology design and implementation. *European Journal of Open, Distance and E-Learning, 22*(2), 41–55. https://doi.org/10.2478/eurodl-2019-0009.

Lachney, M., Bennett, A. G., Eglash, R., Yadav, A., & Moudgalya, S. (2021a). Teaching in an open village: A case study on culturally responsive computing in compulsory education. *Computer Science Education*, 1–27. https://doi.org/10.1080/08993408.2021.1874228.

Lachney, M., Eglash, R., Bennett, A., Babbitt, W., Foy, L., Drazin, M., & Rich, K. M. (2021b). *pH empowered: Community participation in culturally responsive computing education* (pp. 1–22). Learning. https://doi.org/10.1080/17439884.2021.1891421.

Lachney, M., Green, B., Allen, M. C., & Foy, L. (in press). Ethnocomputing and computational thinking. In A. Yadav & U. Dalvad Berthelsen (Eds.), *Computational thinking in education: A pedagogical perspective*. Routledge.

Lachney, M., & Yadav, A. (2020). Computing and community in formal education. *Communications of the ACM, 63*(3), 18–21.

Ladson-Billings, G. (1995). Toward a theory of culturally relevant pedagogy. *American Educational Research Journal, 32*(3), 465–491. https://doi.org/10.3102/00028312032003465.

Latour, B., & Woolgar, S. (1979). *Laboratory life: The construction of scientific facts*. Princeton University Press.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., et al. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin, 36*(4), 119–150.

Margolis, J., Estrella, R., Goode, J., Holme, J., & Nao, K. (2008). *Stuck in the shallow end: Race, education, and computing*. MIT Press.

McGee, E. O. (2020). *Black, Brown*. How Racialized STEM Education Stifles.

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education, 18*(2), 67–92. https://doi.org/10.1080/08993400802114581.

Noble, S. U. (2018). *Algorithms of oppression: How search engines reinforce racism*. NYU Press.

Papert, S. A. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic books.

Papert, S. A. (1993). *The children's machine: Rethinking school in the age of the computer*. Basic Books.

Paris, D. (2012). Culturally sustaining pedagogy: A needed change in stance, terminology, and practice. *Educational Researcher, 41*(3), 93–97.

Richards, B. (2000). Bugs as features: Teaching network protocols through debugging. In proceedings of the thirty-first SIGCSE technical symposium on computer science education (pp. 256-259).

Robinson, K. P., Eglash, R., Bennett, A., Nandakumar, S., & Robert, L. (2020). Authente-Kente: enabling authentication for artisanal economies with deep learning. *AI & SOCIETY*, 1–11.

Rogalski, J., & Samurçay, R. (1990). Acquisition of programming knowledge and skills. In *Psychology of programming* (pp. 157-174). Academic press.

Ryoo, J., Goode, J., & Margolis, J. (2015). It takes a village: Supporting inquiry-and equity-oriented computer science pedagogy through a professional learning community. *Computer Science Education, 25*(4), 351–370.

Saldaña, J. (2016). *The coding manual for qualitative researchers* (3rd ed.). Sage.

Scott, K. A., Sheridan, K. M., & Clark, K. (2015). Culturally responsive computing: A theory revisited. *Learning, Media and Technology, 40*(4), 412–436.

Scott, K. A., & White, M. A. (2013). COMPUGIRLS' standpoint: Culturally responsive computing and its effect on girls of color. *Urban Education, 48*(5), 657–681.

Searle, K. A., & Kafai, Y. B. (2015). Boys' needlework: Understanding gendered and indigenous perspectives on computing and crafting with electronic textiles. In *ICER* (pp. 31-39).

Searle, K. A., Litts, B. K., & 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, 30*, 125–134.

Shargabi, A., Aljunid, S. A., Annamalai, M., Shuhidan, S. M., & Zin, A. M. (2015). Tasks that can improve novices' program comprehension. In 2015 IEEE conference on e-learning, e-management and e-services (IC3e) (pp. 32-37). IEEE.

Shen, J., Chen, G., Barth-Cohen, L., Jiang, S., & Eltoukhy, M. (2020). Connecting computational thinking in everyday reasoning and programming for elementary school students. *Journal of Research on Technology in Education,* 1-21.

Spring, J. (2016). *Deculturalization and the struggle for equality: A brief history of the education of dominated cultures in the United States*. Routledge.

Tenenberg, J. (2019). Qualitative methods for computing education. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge handbook of computing education research*. Cambridge University Press.

Traweek, S. (1988). *Beamtimes and lifetimes: The world of high energy physicists*. Harvard University Press.

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

United States Census Bureau (2010). United States Census Bureau. Accessed Dec. 28th 2020, https://www.census.gov/

Vithal, R., & Skovsmose, O. (1997). The end of innocence: a critique of'ethnomathematics'. *Educational Studies in Mathematics, 34*(2), 131–157.

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. *TechTrends, 60*(6), 565–568.

Zuboff, S. (2019). *The age of surveillance capitalism: The fight for a human future at the new frontier of power*. Profile Books.

Zweben, S. B., & Bizot. (2020). 2019 Taulbee survey. *Computing Research News, 32*(5), 3–63.

Michael Lachney is an Assistant Professor at Michigan State University in the Educational Psychology and Educational Technology program. His research is on the cultural politics of educational technology design and implementation in both school and out-of-school settings.

Aman Yadav is a Professor of Educational Psychology and Educational Technology at Michigan State University. His research and teaching focus

on improving student experiences and outcomes in computer science and engineering at K-16 levels.

Matt Drazin is an Educational Psychology and Educational Technology doctoral student at Michigan State University. His research is on how youth use methods of critical investigation within makerspaces.

Madison C. Allen is a Ph.D. student in the Educational Psychology and Educational Technology program at Michigan State University. Madison's doctoral research focuses on the intersections of culture, technology, and education. Her scholarship examines these intersections through theories of culturally responsive computing and critical theories of education.

William Babbitt is a Research Associate at the Rensselaer Polytechnic Institute in the Science and Technology Studies Department. His research focuses on educational technology design and implementation strategies to improve outcomes in computer science education.