Pair Debugging of Electronic Textiles Projects: Analyzing Think-Aloud Protocols for High School Students' Strategies and Practices While Problem Solving

Gayithri Jayathirtha, University of Pennsylvania, gayithri@upenn.edu
Deborah Fields, Utah State University, deborah.fields@usu.edu
Yasmin Kafai, University of Pennsylvania, kafai@upenn.edu

Abstract: Much attention has focused on student learning while making physical computational artifacts such as robots or electronic textiles, but little is known about how students engage with the hardware and software debugging issues that often arise. In order to better understand students' debugging strategies and practices, we conducted and video-recorded eight think-aloud sessions (~45 minutes each) of high school student pairs debugging electronic textiles projects with researcher-designed programming and circuitry/crafting bugs. We analyzed each video to understand pairs' debugging strategies and practices in navigating the multi-representational problem space. Our findings reveal the importance of employing system-level strategies while debugging physical computing systems, and of coordinating between various components of physical computing systems, for instance between the physical artifact, representations on paper, and the onscreen programming environment. We discuss the implications of our findings for future research and designing instruction and tools for learning with and debugging physical computing systems.

Keywords: Debugging, think-aloud protocol, electronic textiles, high school computing learning, physical computing.

Introduction

The push to bring computing courses into K-12 education (Passey, 2017) introduces students to key computational thinking concepts and practices. Much of student learning takes place in the context of designing screen-based applications such as games, animations or stories (Kafai & Burke, 2014), but now also extends to designing physical computing artifacts such as robots (Blikstein, 2013) or electronic textiles (Buechley, Peppler, Eisenberg & Kafai, 2013). However, most of the attention has been paid to how students learn the necessary computational skills to complete their artifact creation with relatively little attention paid to the equally important debugging skills needed to address the inevitable problems or bugs that arise. Debugging can be defined as the process of "finding out exactly where the error is and how to fix it" within computing programs (McCauley et al., 2008, p. 68). Learning debugging is a key computational practice which involves several specific steps such as isolating the problem, hypothesizing causes, generating and implementing a solution, and running verifications (Gugerty & Olson, 1986; Katz & Anderson, 1987).

Because most of the studies examining debugging were conducted in the 1980s with undergraduate and graduate students (McCauley et al., 2008), very little is known about how their findings will extend to K-12 students and their struggles, especially with the more recently developed physical computing systems (Booth, Stumpf, Bird & Jones, 2016). Electronic textiles (e-textiles) is one such physical construction kit designed for learners to sew microcontrollers and other electronic components onto fabric substrates using conductive thread and program them (Buechley et al., 2013). Debugging e-textile artifacts requires not only fixing problems related to syntax and semantics of the programming language but also addressing circuitry issues, for instance with polarity or short circuits, on multi-surfaced physical artifacts (Jayathirtha, Fields & Kafai, 2018). While a few recent efforts have begun examining adult learners debugging physical computing systems (Booth et al., 2016; DesPortes & DiSalvo, 2019), less is known about how high school students' approach and deal with challenges in debugging their artifacts—critical knowledge for both teachers and students as more of these type of programming activities are now entering K-12 computer science classrooms (Kafai et al., 2019).

In this paper, we move from busy classroom settings into more constrained think-aloud sessions (Ericsson & Simon, 1998) to better understand students' debugging of electronic textiles. While think-aloud protocols often focus on individual learners, we decided to engage pairs of students to facilitate students' verbalization of strategies in a more authentic fashion but also to leverage students' prior collaborative classroom experiences (Fields, Jayathirtha & Kafai, 2019). Furthermore, the multi-modal nature of physical computing systems which require learners to interact with an array of physical and electronic materials, computer screen and

artifacts was video-recorded. We conducted eight think-aloud sessions, each approximately 45-minutes long, as pairs of high school students debugged e-textiles projects with multiple (researcher-designed) problems involving crafting, circuit and programming. We then undertook systematic video analysis (Derry et al., 2010) to answer the following questions: (1) What debugging strategies do novice high school students adopt to identify and address the problems in the e-textiles projects? And, (2) How do they navigate the multi-representational problem spaces as they debugged these projects? In the discussion, we address what we learned about students' successes and struggles in finding and fixing problems, and outline recommendations for further research and design efforts, and pedagogical interventions.

Background

Debugging is key to realizing any computational artifact and requires learners to reason and exercise specific problem-solving strategies (McCauley et al., 2008). Prior studies examining on-screen debugging which have focused predominantly on college and graduate students highlight the role of learners' prior knowledge to work through different problem-solving stages such as fault isolation, hypothesis and solution generation, testing, and verification (Gugerty & Olson, 1986; Katz & Anderson, 1987). While debugging problems, these students were observed pursuing specific strategies: forward reasoning (i.e., bug search starting from the given representations such as code while comprehending an unfamiliar problem space), and backward reasoning (i.e., bug search starting from the observation of incorrect behavior based on compiler feedback once sufficiently familiar with the problem) (Katz & Anderson, 1987). Furthermore, novice and expert adult debuggers pursued qualitatively different approaches to generating hypotheses: experts adopted breadth-first approaches as they hypothesized more exhaustively and tested before generating solutions, while novices took depth-first approaches as they hypothesized about fewer causes before implementing a fix (Vessey, 1985). Gugerty and Olson (1986) also found that expert graduate students tended to test their hypotheses and solutions more often and earlier during debugging compared to their novice counterparts. In sum, expert and novice adults tend to use different strategies and reasoning processes while debugging traditional on-screen computer programs.

Nevertheless, only a few studies have closely examined debugging within physical computing systems. These have revealed undergraduate novices' difficulties in programming Arduino-based artifacts: a majority of debugging issues related to programming while the remaining were distributed between circuitry and the intersection of circuit and programming (Booth et al., 2016; DesPortes & DiSalvo, 2019). A classroom study examining high school students debugging Arduino-based e-textile artifacts noted an almost equal distribution of challenges across circuitry, programming, crafting and design (Jayathirtha et al., 2018), a difference that can be attributed to e-textiles circuitry components such as uninsulated conductive thread, which is more prone to short circuits. While these studies examined debugging while making artifacts, one study presented high school students with pre-designed buggy projects and examined approaches and success rates in solving them (Fields, Searle & Kafai, 2016). This approach of presenting e-textile projects embedded with problems became the foundation for our proposed think-aloud study to examine in a more systematic fashion how student pairs iterate through different phases of debugging—fault isolation, hypotheses and solution generation, and testing and verifications—-with the purpose of better understanding high school students' debugging trajectories and challenges.

Moreover, we wanted to take a more careful look at how novices navigated the complex system of tools and representations while debugging, yet another key aspect of working with physical computing systems (e.g., Kafai, Fields & Searle, 2014; Searle, Litts & Kafai, 2018). From circuit drawings to code within a programming environment to the programmed physical artifacts, students need to draw on different kinds of information across these representations as they debug. These various representations and tools, along with the learners' prior experiences, can be seen as a distributed system in which learners "recognize, recall, pattern match, check for consistency across modality, construct and reconstruct" representations (Hutchins, 1995, p. 284) as they debug. Prior research on related physical computing activities indicates that adult makers often navigate multiple representations during the process of planning, designing and making artifacts (Tucker-Raymond, Gravel, Kohberger & Browne, 2017). Further, with collaborative work, coordination with others is another aspect of the distributed system of e-textiles where prior research has shown that students often distribute work in e-textiles based on perceived expertise in circuitry and coding (Buchholz et al., 2014; Searle et al., 2018). Thus, our research focused on understanding students' strategies and practices as they collaboratively debugged e-textiles projects in order to inform efforts to design supportive tools and pedagogical scaffolds.

Methods

Participants

Our study took place in an introductory computer science high school classroom within a U.S. charter school in a large west coast school district that accepts all geographically local students (55% of students from ethnic groups underrepresented in computing and 54% from economically disadvantaged families). The class was implementing the e-textiles unit (Kafai et al., 2019) of the Exploring Computer Science curriculum in which students make four projects, exploring textile crafts, circuitry concepts such as electric polarity, and computational concepts such as sequence, conditionals, digital and analog inputs and outputs. Students were between their third and fourth projects in the curriculum when each pair, as established during the collaborative third project, were invited to think-aloud and debug a researcher-designed buggy project (what we call a DebugIt) within a 55-minute class period. We created four pairs of DebugIts (8 in total) and had a total of 14 high school students participate, eight males and six females with one of the pairs debugging two projects, as planned by the teacher.

Data collection

We videotaped all eight think-aloud debugging sessions, each roughly 45 minutes long. Each pair was given a researcher-designed DebugIt, printed code, an intention statement (how the project should work after debugging), a "teacher approved" circuit drawing (i.e., with no errors), a computer with programming environment, sewing supplies, and paper (see Figure 1, left). The four DebugIt designs were aesthetically different but had similar bugs, allowing for comparisons of practices and strategies across student pairs. Two of the four DebugIt designs were based on the third (mural) project in the e-textiles unit with two digital switches as inputs and LED lights as digital outputs; the other two DebugIt designs were based on the fourth (human sensor) project, with a hand-crafted analog "squeeze" sensor and LED lights as digital outputs. Each DebugIt had exactly three circuitry and three programming bugs, with different physical layouts but the students did not know the number of bugs. The crafted circuit on the DebugIt had bugs such as loose connections causing short circuits and reverse polarity (see Figure 1, right) while the "teacher approved" circuit drawing had the circuit layout, polarity of the LEDs and microcontroller pin numbers, all as intended in a functional artifact. The code, approximately 30 lines long, had errors such as missing initialization, incorrect logical expressions and mismatched variables, based on challenges novices face while making Arduino-based projects (Booth et al., 2016; Jayathirtha et al., 2018).

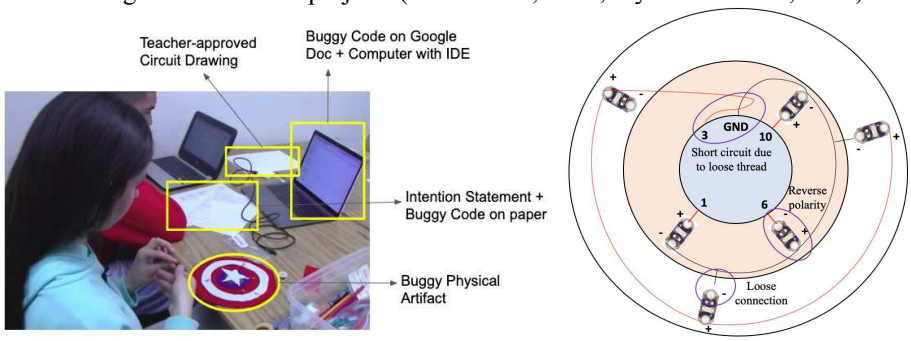

Figure 1. A snapshot of a pair in action depicting different tools and representations that were available to debug (left); a visualization of the given buggy artifact with three circuitry mistakes as labeled (right).

We asked all pairs to think aloud as they debugged the DebugIts and video-recorded them. The researcher regularly probed students to verbalize their thoughts as per the think-aloud protocol (Ericsson & Simon, 1998).

Data analysis

All eight videos were systematically and iteratively analyzed in a deductive fashion (Derry et al., 2010), guided by our two research questions. We adopted a distributed cognition lens throughout our analysis and treated the whole system—learners, tools and representations—as the unit of analysis (Hutchins, 1995) instead of focusing on individuals. As a first step, each video was indexed and sliced into 5-minute segments and details regarding the different interactions were noted. We synthesized all the notes across segments to create descriptive narratives and intermediate visualizations for each pair, as described by Derry and colleagues (2010). These intermediate representations comprising of annotated screen captures and transcripts of the videos enabled discussions among researchers to observe patterns and generate codes. Initially, one of the videos was watched together by all three authors to iteratively generate codes, narrative descriptions and intermediate representations until a consensus about different codes was reached. In the end, each 5-minute segment was coded to capture phases of problem-solving, different strategies and reasoning, interactions with tools and representations, and nature of collaboration within each segment. These were compared with the coding and descriptive results produced by another researcher colleague to ensure consistency. Upon confirming a match of overall descriptions, the remaining seven videos

were watched and analyzed by the first author, in weekly consultation with the other two authors for ten weeks. We elaborate on the details of each of these areas below, further considering how these interactions related to the students' problem solving as a whole.

Findings

Students' use of debugging strategies

Our first research question asked what strategies students adopted to debug e-textiles. To this end, we identified when students used various aspects of strategies discussed in debugging literature, including: isolating a problem by either forward reasoning or backward reasoning, creating hypotheses, generating solutions, and verifying and/or testing hypotheses or solutions (e.g., Gugerty & Olson, 1986; Katz & Anderson,1987; Vessey, 1985). We further looked at the usage of these strategies in relation to relative success in identifying and solving bugs. Pairs who identified and solved half or more (3+) of the given bugs are in bold lettering in Figure 2. We found that the type of problems mattered less for overall success than the strategies used in solving them, in line with Katz and Anderson' (1987) observation (i.e., no type of DebugIt led to automatic success or failure).

While all pairs used each of these strategies at some points, certain frequency and timing of the strategies seemed to be associated with better performing groups. For instance, less successful pairs cycled through the debugging strategies less frequently overall, spending significant time either isolating a single problem (i.e., a faulty light) and/or hypothesizing causes within a single system (i.e., circuitry or code). Looking at the groups' narratives, it was clear that these groups did not consider the entire system of the DebugIt (code, circuitry and other physical properties) through their debugging strategies. In contrast, more successful groups tended to use many of the listed strategies with greater frequency and in patterns of iteration (cycling through a set of strategies repeatedly), visible in the checkered patterns of blocks along the rows with bolded names in Figure 2. We consider specific debugging strategies in more detail below.

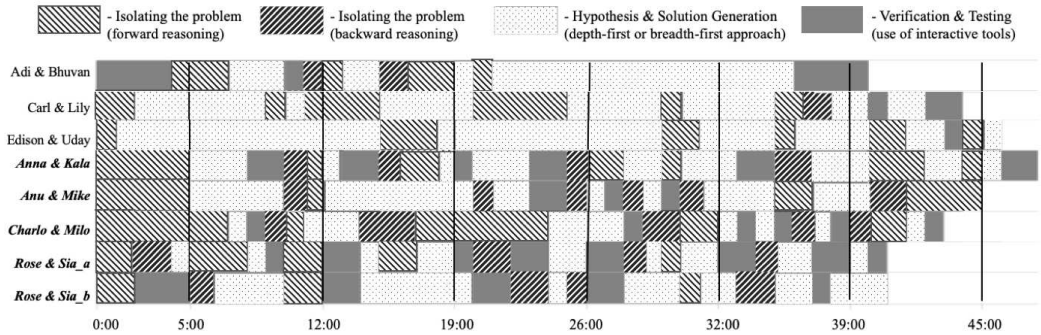

Figure 2. Visualization of debugging trajectories across eight student pairs (horizontal axis is minutes). The different patterns illustrate phases in debugging that pairs iterated through as given by the legend. Student pairs with bold lettered names debugged half or more problems in their DebugIts.

Combined usage of forward and backward reasoning played a strong role in isolating errors in DebugIts, setting pairs up for opportunities to test and verify their ideas. This pattern is visible in the forward and backward reasoning blocks in Figure 2. All pairs spent a significant proportion of time, especially the first few minutes (except Carl/Lily and Edison/Uday), undertaking forward reasoning (see Figure 2). Forward reasoning involves search for faults "that stems from actual written code" or other given representations (Katz & Anderson, 1987, p. 375). With respect to DebugIts, this generally involved visually scanning the physical artifact for "obvious mistakes" (a phase commonly used by students) such as short circuits, running consistency checks across the given code and circuit, and comparing the given and their previous project code. For instance, Anna and Kala spent a significant proportion of their initial ten minutes running exhaustive checks of the artifact and the given code in pursuit of errors. This started with Anna asking Kala to "check if the wires are connected correctly," following inspections of the polarities of LED lights and microcontroller pins. Kala further continued to visually scan the code, matching the connections in the physical circuit with certain code chunks which led them to identify the missing function call. However, pairs that further employed backward reasoning by generating causal explanations for an observed unexpected behavior fixed more issues (boldened pairs in Figure 2).

Backward reasoning involves searches that "start with the incorrect behavior of the program" or the artifact after testing the code and/or the artifact (Katz & Anderson, 1987, p. 375). This involved pairs taking clues from compiling the program or examining the runtime artifact behavior to isolate problems. As an example, Sia

and Rose's fault isolation, around 30 minutes into debugging, stemmed from their observation of unexpected lighting patterns after uploading the compiled code onto the artifact and comparing the artifact behavior to the one listed in the intention statement. Their observation and eventual isolation of the issue with the conditional statements came from the broad set of hypotheses—spreading across the code and the physical circuit—that they generated based on their observation. Overall, pairs that used both forward and backward reasoning repeatedly, tended to identify more problems, leading them to opportunities for verification and testing.

We also observed two different approaches to generating hypotheses and solutions: depth-first and breadth-first. A depth-first approach involved generating solutions based only on limited hypotheses (Vessey, 1985). With DebugIts, this often led to solutions not rooted in actual problems, taking up substantial time for nonessential work. Edison and Uday hypothesized that the cause for dysfunctional lights was a positively charged line that connected multiple lights in parallel (see the lights on the outer ring in Figure 1, right). This was not a malfunction in the project based on the intention for all the lights to blink at once, but without testing the pair decided to redesign the circuit to connect each light to a separate microcontroller pin. Their rapid solution generation from a depth-first approach precluded generation and testing of other probable causes for dysfunctional lights. In contrast, Sia and Rose adopted a breadth-first approach, generating six different hypotheses across circuitry and code as probable causes for the malfunctioning lights. They began by questioning the connections between the lights and the microcontroller pin, then looked at the function calls in the code, cycling through a series of probable reasons and testing some before implementing their solution. This breadth-first approach provided many more opportunities to consider the working of the whole system, eliminate incorrect hypotheses, and generate a viable solution.

A final key to understanding debugging strategies was the presence and frequency of verification cycles. Verification means examining the outcome in comparison to the intended one, either while isolating sites of errors or after testing a probable solution (solid blocks in Figure 2). A simple type of verification involved using the compiler to identify syntactical bugs in code or logical bugs as animated by the artifact behavior. Pairs that used verification and testing to prune their hypotheses or dig for more information as part of their backward reasoning solved more issues compared to pairs that verified later and less often. For example, Charlo and Milo, one of the more successful pairs, connected the artifact to the computer and uploaded the code to "see if [the artifact] works" as described in the intention statement within the initial 10 minutes. Upon observing incorrect behavior, the pair not only continued to hypothesize causes by both backward and forward reasoning but also devised specific strategies to identify the issue by further testing the lights in isolation. This early verification also led the pair to run cycles of verification repeatedly throughout their debugging session (see Charlo and Milo's row in Figure 2). In sum, finding ways to check hypotheses about problem causes and test solutions was a key strategy that more successful pairs used. Notably these tests often had to be conducted across representational spaces in the e-textile artifacts (i.e., across the whole system of the artifact) in order to narrow down the probable solution space.

Students' navigation across representations and people

Our second research question asked how students navigated the distributed problem spaces (both objects and people) while fixing DebugIts. Despite students' limited experience with e-textiles, their fluid interaction with tools and representations and their flexible collaborative styles demonstrated their developing abilities to recognize relationships between different parts of the e-textiles system. Below we describe a few instances of how pairs ran consistency checks across modalities, matched patterns, and reconstructed representations similar to adult experts (Hutchins, 1995; Tucker-Raymond et al., 2017). Then we further consider how they adjusted their collaborative arrangements to match the demands of the task at hand.

One aspect of the distributed system of e-textiles is the interrelatedness of the subsystems of circuitry, code, and craft (Kafai et al., 2014). With respect to handling multiple representations in the problem space, students demonstrated their understanding of the connections between these areas as they ran consistency checks between circuit diagrams, printed and onscreen code, the intention statement, and the various surfaces (top, bottom, inside) of the physical artifact. Initially, many pairs ran consistency checks across a subset of these representational spaces in order to comprehend the problem space during the first few minutes. All the pairs attempted to understand the circuit layout by visually scanning the physical artifact (without connecting it to power source) and comparing it to the circuit drawing (Figure 3, left), often tracing the circuitry lines with fingers. In a few cases (4 out of 8), after students connected the artifact to the computer as a power source (to observe how the artifact functioned), they ran active visual scans comparing the artifact's behavior with the intention statement and the given code (Figure 3, right). In this way the students mapped the physical connections between microcontroller pins and lights to the corresponding variables defined and setup as INPUTs in the code, focusing on the appropriate code chunks. This demonstrates students' recognition of the relationship between the code and circuitry, a key aspect of debugging physical computing systems (DiSalvo & DesPortes, 2019; Fields et al., 2016).

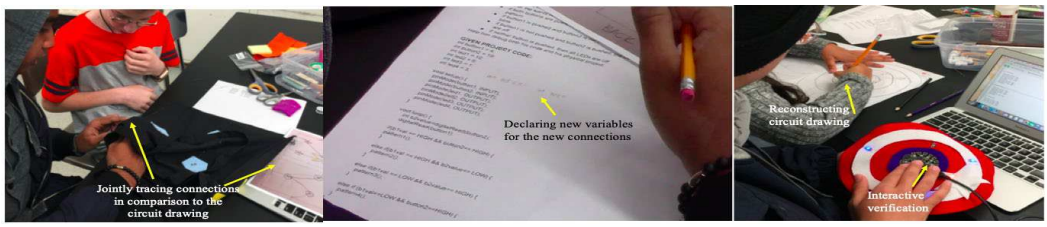

Figure 3. Edison and Uday jointly compared the physical circuit with the circuit diagram (left); Uday appended new lines of code to the given code (middle); Anu redrew the circuit while Mike verified the outcome (right).

Students' fluid navigation across the different tools and spaces continued even when students introduced their own representations into the space, using either prior knowledge or developing new representations altogether. For instance, in order to check the provided code for "obvious mistakes," more than half of the pairs (6 out of 8) brought in their previous (working) project code for comparison, looking for structural differences between code chunks. In addition, all the pairs compiled the onscreen code, watching for the "red dot" signaling a syntactic error in the Arduino programming environment, demonstrating their familiarity with the compiler as an interactive tool. However, though most of the pairs continued to examine the code after syntactic fixes, one pair (Adi & Bhuvan) concluded that syntax-error-free code was bug-free, which led them to ignore the logical issues in the code and instead spend their time hypothesizing issues only with circuitry. In addition to bringing in prior project codes, students introduced their own representations in the form of new notations and diagrams. For example, Edison and Uday first made changes to the code printed on paper (see Figure 3, middle). They chose to write their changes on paper before putting them into code as a way to coordinate their understanding (i.e., for visibility) as well as a way to keep track of the changes they made in case they needed to undo them.

Further, students drew on prior knowledge while evaluating circuitry and generating new diagrams. For example, Anu and Mike used a personal checklist based on rules they had learned from their prior e-textile projects to evaluate the circuitry, both of the physical artifact and the circuit diagram. They scanned the physical artifact for "cross overs," frayed or loose threads that cause short circuits when crossing polarity lines. They also carefully looked at the circuitry in the diagram and the physical artifact for functional positive and negative connections as well as connections between LEDs and pin numbers on the microcontroller. As a result of these inquiries they chose to redraw the circuit diagram, changing to a version they thought would work better. Even though their new diagram included unnecessary changes (since they worked with a naïve prior conception that parallel circuits were buggy), the redrawing of the circuit diagram demonstrated their understanding of the importance of these paper-based representations for coordinating their individual understandings and the enactment of their crafted solutions.

In addition, students also developed flexible collaborative arrangements, shifting how they collaborated during their class period of making artifacts. We observed that many pairs often divided the labor between circuit and coding, using a divide-and-conquer collaborative strategy to analyze subsets of the problem space (Figure 3, right). This reflects some of the ways students have traditionally divided e-textiles tasks into circuitry and coding during e-textiles designs (Lui et al., 2019). On the other hand, sometimes pairs worked together by coordinating activity across related spaces within the whole, jointly attending to specific aspects of the problem space, such as co-investigating the code or having one person call out the circuitry connections from the diagram while the other mapped the corresponding connections on the physical artifact or the code (Figure 3, left). Frequently, pairs moved between these two modes depending on the task at hand. For example, Sia and Rose divided the coding and sewing tasks between themselves while comprehending the problem space. However, they jointly conducted tasks that benefited from extra hands and eyes when running consistency checks and testing the whole system. In yet another case, Edison and Uday worked together for most of the time except when one of them had to sew to implement one of their solutions. In that case, the other partner inspected the code, shifting to a divide-and-conquer strategy. In sum, joint debugging within a distributed system not only revealed expert-like navigation across multiple representational spaces of the system but also displayed students' strategic and adaptive collaborative arrangements, pointing to the complexity of debugging with physical computing systems like e-textiles.

Discussion

In this paper we investigated students' debugging strategies and practices in e-textiles, expanding our understanding of debugging in the understudied area of physical computing systems. While prior research in e-textiles has illustrated some of novices' conceptual struggles while making (and debugging) e-textiles, our analyses of time-constrained, researcher-designed DebugIts in think-aloud settings allowed for more nuanced examination and comparison of debugging strategies and practices across pairs of students. Some debugging strategies we identified closely matched those previously adopted by programmers in onscreen debugging (e.g.,

McCauley et al., 2008). Yet, we also observed distinct problem-solving practices that were specific to the distributed nature of the physical computing task, where problems were spread across different modalities and afforded rich opportunities to iteratively locate faults, generate hypotheses and solutions, and run system-wide tests. The ways in which learners had to aggregate information, run consistency checks, and match patterns across code and circuitry called for more elaborate practices than just debugging onscreen programs. Further, students exhibited the ability to move between different collaborative arrangements as suited to various tasks of debugging the distributed physical system. These findings demonstrate some of the unique challenges and opportunities available in debugging physical computing systems, calling for deeper research on strategies to support learners through pedagogical and tool designs, and further research in debugging in physical computing systems.

One particular challenge this study revealed is the difficulty some students had in employing system-level strategies both while isolating issues and hypothesizing probable causes within physical computing systems. Students who were more successful followed approaches of hypothesizing multiple causes for a problem and pruning these hypotheses with testing and verifying across the whole system (Gugerty & Olson, 1986). How can we provide more support for listing multiple possible causes of errors across the system and systematically reasoning through these hypotheses before jumping to a solution that may be time-consuming and/or ineffective to implement? One solution may be to create debugging activities like those in this paper, as a means of providing students more opportunities for debugging without the pressure of successfully realizing a personally relevant project. Prior research on less challenging DebugIts in e-textiles suggests that students may find these activities encouraging and helpful for their learning, pulling them outside of their normal design space to debug a project they are less emotionally connected to (Fields et al., 2016). In addition, opportunities like time-limited DebugIts may provide an explicit opportunity for students to reflect on and share their debugging strategies, including their own invented collaborative arrangements. Sharing design strategies is already a documented productive practice in classrooms focused on physical computing (Fields et al., 2018). Sharing debugging strategies more explicitly may be a further means to support student learning and agency. Another way to support students is to share collaborative arrangements that help to coordinate across these areas, helping learners jointly problem-solve in ways that move beyond divide-and-conquer strategies that seem to occur more intuitively in areas like e-textiles (Buchholz et al., 2014; Lui et al, 2019). Further research is needed to understand how DebugIts may support learning, including how and where within a design trajectory they are useful.

A further challenge revealed in this study was students' difficulty of coordinating between various components of physical computing systems, for instance between the physical artifact, representations on paper, and the onscreen programming environment. The struggle to coordinate different aspects of e-textiles surfaced in the ways some pairs failed to account for the whole system during the debugging process. One way to support students in looking at the whole system is to design and develop suitable tools. Current compilers, debuggers and multimeters are mostly designed for individual, localized testing of code or circuitry within a single sub-system of e-textiles but not across the whole system at once. Instead we need new tools that help students navigate across code, circuitry, and multiple surfaces of a physical artifact in relation to one another rather than in isolation. This is in line with the call for such specific tools by DesPortes and DiSalvo (2019) that can support student testing of their naïve conceptions and debugging. Specific tools that make whole system state visible and help students check across both circuitry and code is one of the ideas for new tools, though there may be many more. Learning how to debug physical computing systems like e-textiles involve complex challenges in understanding not only software but also hardware issues. Our analyses of think-aloud sessions provided insights into what students are capable of and where they might need more support in the form of tools and pedagogical scaffolds.

References

Booth, T., Stumpf, S., Bird, J., & Jones, S. (2016). Crossed wires: Investigating the problems of end-user developers in a physical computing task. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 3485-3497. ACM.

Buchholz, B., Shively, K., Peppler, K., & Wohlwend, K. (2014). Hands on, hands off: Gendered access in crafting and electronics practices. *Mind, Culture, and Activity*, *21*(4), 278-297.

Buechley, L., Peppler, K., Eisenberg, M., & Yasmin, K. (2013). *Textile Messages: Dispatches from the World of E-Textiles and Education.* New York, NY: Peter Lang Publishing.

Blikstein, P. (2013). Gears of our childhood: constructionist toolkits, robotics, and physical computing, past and future. In *Proceedings of the 12th International Conference on Interaction Design and Children*, 173-182.

Derry, S. J., Pea, R. D., Barron, B., Engle, R. A., Erickson, F., Goldman, R., ... & Sherin, B. L. (2010). Conducting video research in the learning sciences: Guidance on selection, analysis, technology, and ethics. *The Journal of the Learning Sciences*, *19*(1), 3-53.

DesPortes, K., & DiSalvo, B. (2019). Trials and tribulations of Novices working with the Arduino. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, 219-227.

Ericsson, K. A., & Simon, H. A. (1998). How to study thinking in everyday life: Contrasting think-aloud protocols with descriptions and explanations of thinking. *Mind, Culture, and Activity*, *5*(3), 178-186.

Fields, D. A., Jayathirtha, G., & Kafai. Y. B. (2019). Bugs as a nexus of emergent peer collaborations: Contextual and classroom supports for solving problems in electronic textiles. In Lund, K., Niccolai, G., Lavoué, E., Hmelo-Silver, C., Gweon, G., Baker, M. (Ed.) *In the Proceedings of* the *13th International Conference on Computer Supported Collaborative Learning*, 472-479.

Fields, D. A., Kafai, Y., Nakajima, T., Goode, J., & Margolis, J. (2018). Putting making into high school computer science classrooms: Promoting equity in teaching and learning with electronic textiles in Exploring Computer Science. *Equity & Excellence in Education*, *51*(1), 21-35.

Fields, D. A., Searle, K. A., & Kafai, Y. B. (2016). Deconstruction kits for learning: Students' collaborative debugging of electronic textile designs. In *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education*, 82-85. ACM.

Gugerty, L., & Olson, G. (1986). Debugging by skilled and novice programmers. *ACM SIGCHI Bulletin*, *17*(4), 171-174.

Hutchins, E. (1995). How a cockpit remembers its speeds. *Cognitive science*, *19*(3), 265-288.

Jayathirtha, G., Fields, D. & Kafai., Y. (2018) Computational concepts, practices, and collaboration in high school students' debugging electronic textile projects. In *Proceedings of International Conference on Computational Thinking Education,* 27 - 32. The Education University of Hong Kong, China.

Kafai, Y. B., & Burke, Q. (2014). *Connected code: Why children need to learn programming*. Cambridge, MA: MIT Press.

Kafai, Y. B., Fields, D. A., Lui, D. A., Walker, J. T., Shaw, M. S., Jayathirtha, G., ... & Giang, M. T. (2019). Stitching the loop with electronic textiles: Promoting equity in high school students' competencies and perceptions of computer science. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 1176-1182. New York, NY: ACM.

Kafai, Y., Fields, D., & Searle, K. (2014). Electronic textiles as disruptive designs: Supporting and challenging maker activities in schools. *Harvard Educational Review*, *84*(4), 532-556.

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

Lui, D., Kafai, Y., Litts, B., Walker, J., & Widman, S. (2019). Pair physical computing: high school students' practices and perceptions of collaborative coding and crafting with electronic textiles. *Computer Science Education*, 1-30.

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.

Passey, D. (2017). Computer science (CS) in the compulsory education curriculum: Implications for future research. *Education and Information Technologies*, *22*(2), 421-443.

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.

Tucker-Raymond, E., Gravel, B. E., Kohberger, K., & Browne, K. (2017). Source code and a screwdriver: STEM literacy practices in fabricating activities among experienced adult makers. *Journal of Adolescent & Adult Literacy*, *60*(6), 617-627.

Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies*, *23*(5), 459-494.

Acknowledgments

This work was supported by a grant from the National Science Foundation to Yasmin Kafai (#1742140). Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF, the University of Pennsylvania, or Utah State University. We would like to thank Justice Walker and Mia Shaw for their valuable feedback on our writing.