Computer Science Education

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/ncse20

Distributed debugging with electronic textiles: understanding high school student pairs' problem-solving strategies, practices, and perspectives on repairing physical computing projects

Gayithri Jayathirtha, Deborah Fields & Yasmin Kafai

To cite this article: Gayithri Jayathirtha, Deborah Fields & Yasmin Kafai (21 Jan 2024): Distributed debugging with electronic textiles: understanding high school student pairs' problem-solving strategies, practices, and perspectives on repairing physical computing projects, Computer Science Education, DOI: 10.1080/08993408.2023.2297738

To link to this article: https://doi.org/10.1080/08993408.2023.2297738

COMPUTER SCIENCE EDUCATION
https://doi.org/10.1080/08993408.2023.2297738

Routledge
Taylor & Francis Group

Distributed debugging with electronic textiles: understanding high school student pairs' problem-solving strategies, practices, and perspectives on repairing physical computing projects

Gayithri Jayathirtha[a], Deborah Fields[b] and Yasmin Kafai[c]

[a]Education Studies, College of Education, University of Oregon, Eugene, OR, USA; [b]Instructional Technology & Learning Sciences, College of Education and Human Services, Utah State University, Logan, UT, USA; [c]Teaching, Learning, and Literacy, Graduate School of Education, University of Pennsylvania, Philadelphia, PA, USA

ABSTRACT

Background and Context: Debugging is a challenging yet understudied practice within recent collaborative K-12 physical computing contexts. We examined think-aloud interviews and reflections of seven high school student pairs who debugged researcher-designed buggy electronic textile projects.

Objective: We asked: (1) What strategies did student pairs employ as they debugged e-textile projects? (2) How did interactions between people, tools, and representations shape debugging approaches? (3) How did students reflect on this debugging experience in relation to their learning in the e-textile unit?

Method: We qualitatively analyzed eight think-aloud videos (~45 min each) and student reflections (~15 min twice).

Findings: Debugging required iteratively problem-solving across multiple modes, employing system-level , and a mix of collaborative and individual strategies. All students attributed various gains to this activity, including problem-solving-related practices and positive stances toward collaboration.

Implications: Our analysis expands debugging research to include K-12 physical computing contextand highlights the potential of similar debugging activities as intentional learning experiences.

ARTICLE HISTORY
Received 1 September 2022
Accepted 18 December 2023

KEYWORDS
Debugging; physical computing systems; electronic textiles; high school computing; collaboration

1. Introduction

As computing education is moving into K-12 classrooms, much attention has focused on teaching students programming and developing appropriate tools and programming environments (e.g. Kelleher and Pausch (2005); Maloney et al. (2010)). Most of the work on students' learning has focused on supporting students to plan and write programs paying much less attention to debugging, an equally important issue that nearly all programmers face – from novices to experts (Robins et al., 2003). Debugging, defined as the process of identifying and fixing errors in computing systems, is one of the central computing

CONTACT Gayithri Jayathirtha gayithri@uoregon.edu Education Studies, College of Education, University of Oregon, 1215, Eugene, OR 97403, USA

thinking practices (Brennan & Resnick, 2012). Debugging requires careful attention to different aspects of the computing system and consists of several phases such as isolating bugs or problems, hypothesizing causes, fixing bugs, and verifying solutions (McCauley et al., 2008). The high demand it places on novices can keep them from further engagement within computing (Murphy et al., 2008; Perkins et al., 1986). And yet, only a few studies shed light specifically on how novices debug or what pedagogical practices and activities can better support them (e.g. Lowe (2019)).

Furthermore, there is a need to understand debugging within physical computing programming environments such as electronic textiles (hereafter e-textiles), where LEDs, switches, and sensors are sewn with conductive thread on fabric and programmed via a microcontroller in the Arduino programming environment often with other people (Buechley et al., 2013). Past debugging studies have provided substantial knowledge about classic bugs and strategies of novice students (e.g. McCauley et al. (2008)), although mostly limited to post-secondary contexts. The review conducted by McCauley et al. (2008) sheds light on debugging within computing education research: summarizing the findings across approximately 40 papers to answer questions about the different kinds of bugs learners encountered, differences between novices and experts in approaching them, and how educators can support learners pedagogically. With computing education research has historically been housed in post-secondary settings, most studies exploring learners' debugging focus on college-age students – either from post-secondary learning contexts such as undergraduate and graduate classrooms – or expert programmers from the software industry (McCauley et al., 2008). Further, studies were predominantly (∼65%) conducted in the 1980s-one of the most productive decades in computing education research – within the then popular programming languages such as Basic, Pascal, Fortran, and so on. This implies the need for work to understand debugging within the more recent languages and environments such as Java, Python, and Arduino. Further, environments such as physical computing add complexities that may reshape what productive debugging strategies may look like, with tangible microcontroller-based circuits in addition to on-screen programming and multiple tools and representations of the problem space (e.g. circuit diagram, physical interactive circuit, sometimes static text of the code, and interactive code on the compiler), with implications for shaping debugging experiences within these settings.

Finally, debugging in both professional and educational contexts often happens collaboratively, but little research has attended to this issue. Reflecting this trend, recent studies report on the educational productivity of engaging pairs and groups of students in collaborative coding experiences, in contrast to the focus on individual student debugging as reported in earlier work (e.g. Denner et al. (2014); Jayathirtha et al. (2020); Lui et al. (2019)). However, the limited exploration of social aspects of learning within computing education (Grover & Pea, 2013) means there is little understanding of how collaboration and interaction between students and the problem-space affect debugging. While a few recent studies have shed light on the socioemotional learning and identity development that accompany debugging experiences (Dahn et al., 2020; Whalley et al., 2021), these have not included physical computing. Thus, there is a distinct need to jointly examine conceptual, social, and perspectival aspects of the debugging experiences of K-12 students to present a holistic picture of student debugging experiences, especially in physical computing (Grover & Pea, 2013).

In order to understand novices' approaches to debugging physical computing projects, we studied novice, secondary school student pairs debugging strategically designed physical computing projects with several overlapping, pre-designed bugs stretching across hardware and software. We conducted qualitative analysis on eight video-taped, think-aloud debugging sessions where seven student pairs collaboratively debugged electronic textile (e-textiles) projects and interviewed students later about their experiences. We analyzed the observational video and interview data to answer the following research questions: (1) What strategies did student pairs employ as they debugged e-textile projects? (2) How did interactions between the people, tools, and representations shape their debugging? (3) How did students reflect on this debugging experience in relation to their learning in the e-textile unit overall?

2. Background

As computing education within K-12 contexts has expanded globally over the past decade (Anwar et al., 2020; Hubweiser et al., 2011), the use of new programming environments such as block-based programming and physical computing have changed learners' debugging practices, setting the stage for diversifying research on debugging. While programming environments and tools adopted in introductory K-12 classrooms have broadened in the recent decade (Hodges et al., 2020; Horn, 2018), there is a need to update our understanding of different strategies, practices, and perspectives that students develop while debugging within these newer contexts. For example, block-based programming environments are designed to make computing more accessible to young programmers by minimizing syntactic errors, a previously common area of debugging (Kummerfeld & Kay, 2003). Related, programming environments such as Logo or Scratch provide on-screen anthropomorphic agents to better visualize program execution (Lewis, 2010), influencing learners' debugging practices in specific ways. For instance, Lewis (2012) examined middle school students debugging their programs in Scratch, noting that students' attention to the program state – visible by the orientation of the agent and variables in the environment – significantly shaped their debugging trajectories and outcomes. More recently, Martinez et al. (2020) pointed out how block-based and hybrid programming environments elicited different debugging strategies among novices and shaped their programming trajectories, further highlighting the role of programming environments in shaping debugging practices among learners.

Physical computing is another genre of programming environment that further expands debugging across space and modes through expanding computation to physical circuits consisting of microcontrollers, sensors, and actuators (Hodges et al., 2020). DesPortes and DiSalvo (2019) highlight this shift in interactions between programmers and the environment where undergraduate novices struggle to program and debug within an Arduino-based physical computing environment where bugs or errors are present in hardware and software. In addition, collaborative work on computing artifacts further changes the debugging context, as learners must coordinate between each other to debug, for instance in structured pair programming contexts onscreen (Denner et al., 2014) and even across visible and tangible modes with physical artifacts (Lui et al., 2019). Previous studies around collaborative problem-solving have highlighted the role of student interactions such as having a shared problem space, paying joint attention, and

having mutuality in shaping outcomes for student pairs (Barron, 2000). Emergent collaborative arrangements can also shift the context of debugging beyond pairs as groups or even whole classrooms share knowledge (Fields et al., 2019). These types of collectively collaborative contexts can support students' development of positive perspectives about learning to program (Morales-Navarro et al., 2021, 2021; Tofel-Grehl et al., 2017).

This expansion of computing environments, tools, and activities gives rise to a need for new analytical frameworks such as distributed cognition that account for collaborative arrangements and interactions between tools and representations while understanding debugging. While there is extensive knowledge about novice debugging strategies and reasoning in the 1980's programming environments, thanks to research guided by cognitive theories (McCauley et al., 2008), there are far fewer accounts of how these problem-solving practices are distributed across people, diverse tools, and representations. Further, recent research has begun to attend to how students' attitudes and perspectives on "failure" and debugging influence students' debugging practices and experiences, for instance growth mindset, perseverance through failure, and perspectives about debugging and programming (e.g. Dahn et al. (2020); Morales-Navarro et al. (2021)). Expanded theoretical and analytical lenses must consider the social, tool-based, and affective components of debugging which are the focus of this study. Below we summarize relevant debugging literature that this study draws on and contributes to (a) studies about individual students' debugging reasoning and strategies; (b) debugging as a distributed activity between people, tools, and representations; and, (c) development of students' perspectives of debugging.

2.1. Individual student debugging reasoning and strategies

Debugging is a key computing practice that demands learners reason and exercise-specific problem-solving strategies in order to realize functional computing artifacts (McCauley et al., 2008). Previous research (Katz & Anderson, 1987; Vessey, 1985) highlights five specific phases of problem solving in debugging: (a) Initial exploration of the computing system, (b) Testing the system – either to locate errors or to verify solutions, (c) Strategic location of errors, (d) Hypothesis generation to identify the cause of the error, and (e) Implementing the solution. The exploratory phase, i.e. gaining familiarity with the function and structure of the system can involve reasoning in one of the two directions; students can either draw from their prior knowledge or rely on given representations such as code to generate hypothesis and solutions, or test the buggy system, compare its current behavior with the expected one, and let that observation drive the rest of their debugging process (Katz & Anderson, 1987). Students in the next phase—i.e. testing the system actively – compare the observed with the intended output to isolate the error, and this could require students to explore the program execution or control flow. However, on some occasions, testing can follow the solution implementation phase with an intention to ensure the correctness of the implemented solution.

A mix of initial exploration and testing is followed by the adoption of specific reasoning techniques to locate errors. Following the exploratory phase, continued *forward reasoning* while isolating errors can take a few different forms: simple exploration of the program in the serial order (i.e. scanning through line-by-line as printed or typed program) or exploration along the program execution flow, which accounts for conditional and

looping structures that may derail the sequential execution of programs. In contrast, adopting *backward reasoning* to isolate errors may involve identifying differences between the intended and observed outcomes. In either case, locating errors can involve shallow or deep reasoning (Tubaishat, 2001) – heuristic reasoning without detailed attention to the programming constructs in the case of shallow reasoning approach or paying particular attention to the program semantics, syntactic rules, and the algorithm at play in the case of deep reasoning. Further, one can examine the given program through a fine-grained analysis of the code, by looking more closely at the individual lines or even the variables within the program or by adopting a more coarse-grained approach of chunking programs and exploring them to locate errors.

Isolating errors then makes room for hypothesis generation in which prior research has shown certain qualitative differences between the approaches that novice and expert adult debuggers pursue. Within hypothesis generation, experts adopted *breadth-first approaches* as they hypothesize more exhaustively, mostly informed by both forward and backward reasoning before generating solutions, while novices took *depth-first approaches* as they hypothesize about fewer causes before implementing a fix (Vessey, 1985). This is in line with (Lui et al., 2019) observations that more advanced learners took time to "understand the code, and identify where and why it was wrong" while novices' exploration was limited to "superficial difference in program outcomes". Further, this concurs with Gugerty and Olson (1986)'s finding that expert graduate students tended to test their hypotheses and solutions more often and earlier during debugging compared to their novice counterparts. In sum, while prior research highlights different specific strategies and reasoning processes that programmers adopt while debugging, it also alludes to qualitative differences in strategies that novices and experts employ for debugging traditional on-screen computer programs. However, since most of these studies solely involve on-screen, individual-focused (versus collaborative) task arrangements, we know very little about how these strategies translate or even relate to more distributed computing environments where debugging may be complicated by collaboration and physical tools.

2.2. Distributed debugging across people

In addition to different debugging strategies, collaborative arrangements are consequential to shaping the debugging process and the participants' experiences. Pair programming – a programming practice where two programmers take on collaborative roles as they share tools and jointly produce a computing artifact (Williams et al., 2000) – is one such well-researched collaborative arrangement. Debugging within this arrangement can involve discourse around hypothesizing reasons and probable solutions, and even running tests jointly. Derived from software engineering industry practice, pair programming in learning contexts involves assigning specific roles of "drivers" and "observers" to participants in pairs sharing a computer. "drivers" are expected to be on the keyboard, paying attention to the programming details, while "observers" are required to guide the driver with the big picture understanding of the algorithm or the problem being solved. A structured exchange of roles during specific intervals ensures that students receive similar opportunities to engage with programs – taking turns examining the big-picture

and the line-wise details of programming. The "shoulder-to-shoulder" arrangement aims to enable learners to learn from each other as they jointly design, construct, review, and debug programs.

Like pair programming, pair debugging can positively shape students' debugging processes. Murphy et al. (2008) have noted that transactive discussions were helpful in pair debugging among undergraduate students. A comparative study of undergraduate students found that pair programming led pairs to successfully complete the assignments and to generate more robust code in comparison to the students who worked alone. In another study that compared solo and pair debuggers, Zhao (2014), discussed how pair programmers debugged more efficiently, i.e. taking lesser time per bug and at the same time reported the debugging activity to be more enjoyable than the solo programmers. Zhao also reported differences between solo and pair debuggers' strategies while debugging. While pairs spent some time initially planning and generating a common understanding of the problem-space and a set of action items, solo programmers jump into problem-solving more quickly by reading and even testing the given code. This is similar to Liebenberg et al. (2012)'s study that reported that pairs of high school girls better debugged and reviewed programs due to interactions with their partners. A study closely examining student pairs problem-solving noted the need for pairs to establish shared problem space, joint attention, and mutuality to successfully problem-solve (Barron, 2000).

In addition to creating and sustaining joint problem-solving spaces, learners also reported increased self-confidence, motivation, and satisfaction at the end of the pair programming activity. For instance, high school girls in Liebenberg et al. (2012)'s study reflected on their increased persistence, especially while debugging, in comparing their pair programming experience to when programming alone. Opportunities to "bounce off ideas" and build on them during programming helped learners develop positive attitudes towards programming. Along with learning new strategies to problem solve, the girls also developed new perspectives about collaboration itself – collaborating with friends on projects now meant that they had to trust and support each other to sustain participation, shifting classroom culture from being competitive to being collaborative in nature. Given that students' perceptions about collaborative work significantly shape their participation in pair programming and debugging activities among K-12 students (Campe et al., 2020), deeper analysis of the different perspectives that students develop as they engage in collaborative debugging is key to better supporting students' engagement with computing.

2.3. Distributed debugging across tools and representations

With most studies focused on on-screen programming environments, programming and debugging practices within physical computing have been understudied. Although physical computing environments involve on-screen programs, they differ from traditional programming since computation and artifact generation is distributed between the computer and physical artifacts such as robots or textile-based objects (Przybylla & Romeike, 2014). Traditional debugging is mostly confined to on-screen manipulation of

code, while the distributed nature of debugging in physical computing complicates the process of debugging and requires tracing across multimodal space. 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 program and debug in these programming environments. 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). Each of these representations can be treated as "inscriptions" that communicate through a system of symbols and meanings (e.g. polarity information printed on the circuit components or the English language works used as keywords in a programming language).

Prior research on related physical computing activities indicates the benefits of such analytical lenses because it makes visible different processes of programming, outside of the strategies and reasoning adopted. For instance, Wagh et al. (2017) noted how youth's computing practices such as problem decomposition and debugging were distributed across the physical toolkit, group members, and various representations that they generated. A group of four youth, making a physical musical artifact using Makey Makey and Arduino toolkits, simulated the different user interactions with their projects, iteratively manipulated the organization of the different components making up the project, and generated and revised drawings and tables which represented different aspects of the project as they realized a functional artifact. Here, programming and debugging were analyzed beyond the different reasoning and strategies that the group members adopted, and instead the different computation-related practices were captured as the group made the artifact. Similarly, computing practices of expert adult makers making a broad variety of physical computing projects such as ergonomic keyboards, daft punk, and TRON helmets were distributed across multiple representations, interactions between people, and tools during the process of planning, designing, and making artifacts (Tucker-Raymond et al., 2017). Given physical computing has entered introductory computing K-12 school classrooms only recently, there are far fewer studies that have examined programming and debugging in these contexts compared to the completely on-screen environment (Hodges et al., 2020; Y. B. Kafai et al., 2019).

2.3.1. Learning with and debugging e-textiles: a special case of physical computing

E-textiles is a special instance of physical computing that has only recently been integrated within introductory computing high school curricula and has been adopted in a sizable number of classrooms (Y. B. Kafai et al., 2019; Nakajima & Goode, 2019). Creating e-textile projects involves sewing connections between microcontrollers and sensors and actuators using conductive thread. Similar to making other physical computing projects, students are tasked with navigating a broad set of tools and representations across the physical and the digital space as they design and realize circuits and associated computer programs (e.g. Litts et al. (2021)). Making e-textiles entails its own affordances and challenges. Sewing circuits in e-textiles using conductive thread contains challenges different from traditional electronic kits with breadboards and insulated wires: uninsulated conductive thread, when imprecisely sewn, leads to short circuits and loose contacts, while sewing

circuitry onto fabric makes these connections harder to revise and test (Lui et al., 2019). Further, bugs are often distributed across the circuitry (on physical artifacts) and programs on the computer screen (Jayathirtha et al., 2018). Realizing functional artifacts in this context requires students to explore and adopt different strategies and reasoning across the distributed space consisting of a wide variety of tools and representations.

The distributed nature of the task invites and constrains collaboration in ways different from on-screen programming activities. While the demand for the task to involve multiple knowledge bases, resources, and tools holds the potential for collaborative engagement, the individualistic design of the tools such as needles and laptops precludes more than one set of hands from using them. It is generally easier to silo work among collaborative pairs, i.e. where members split up sewing and coding tasks (Lui et al., 2019). Yet unplanned collaboration can emerge while debugging, for instance when projects are easily visible and classroom environments allow students to chat, commiserate, and share ideas about bugs across or between tables (Fields et al., 2019). In this study, we take closer account of the role of sustained collaboration in debugging a complexly designed set of problems.

2.4. Learner perspectives and reflections around debugging

While studies of debugging have tended to focus more on learner strategies and practices, increasingly scholars have argued for taking a more holistic view that acknowledges the importance of students' personal experiences, perspectives, and emotions during these challenging experiences (e.g. Sengupta et al. (2021, 2018)). Debugging can be a challenging experience for any programmer, but perhaps especially novices where facing bugs and trying to fix them can generate feelings of helplessness, frustration, and anxiety (Perkins et al., 1986). These feelings can lead to a "fixed" mindset, where students feel that they cannot progress in their ability to debug and program (Morales-Navarro et al., 2021; Scott & Ghinea, 2013). Furthermore, students' beliefs about themselves (as capable of learning and growth or as incapable and "fixed") can lead to withdrawal and a lack of perseverance in computer programming (Margolis et al., 2017; Scott & Ghinea, 2013). Student self-perceptions can also shape peer interactions, even moving them towards inequitable ways of engaging with debugging, for instance when students perceive someone (even themselves) as less capable of debugging and sideline that individual from a task (Shah & Lewis, 2019). On the other hand, debugging experiences can lead to positive or growth mindsets where students develop stronger feelings of capability that help them persevere in the face of challenges and "failures" (Morales-Navarro et al., 2021).

For all of the above reasons, scholars have begun calling for studies of student perceptions around debugging that acknowledge the deep relationship between thinking and emotion and provide insight into supporting students' positive perspectives about debugging (Dahn & DeLiema, 2020; Dahn et al., 2020). This idea highlights ways in which debugging is a complex process where problem solving, emotion, and sense of self (or identity) intersect (Dahn & DeLiema, 2020). "Affect surrounds moments of critical thinking during failure," and we must account for the role of affect in learning computer science (DeLiema & Dahn, 2020, p. 218). The question is how to support a positive affect

where students learn to frame themselves as capable problem solvers. Dahn and DeLiema (2020) explore the ways in which identifying and solving bugs was usually accompanied by emotional responses which then shaped students' perceptions of the activity itself, in this case their relation to programming. Further, some studies demonstrate that students do not even need to be entirely successful in working through bugs or fixing mistakes in order to persevere in a later activity (Kapur, 2008; Morales-Navarro et al., 2021).

Reflecting on debugging can also be beneficial to students' computing development whether at the undergraduate level (e.g. Whalley et al. (2021)) or in K-12 education (e.g. DeLiema and Dahn (2020)). Sharing stories of failure in the moment can help students recognize that they are not alone in facing challenges (Fields, Jayathirtha, & Kafai, (2019)). More formal reflections such as journaling, storytelling, and art making can also help students build confidence as competent debuggers (Dahn & DeLiema, 2020). In this paper, we seek to study students' debugging as part of a more holistic process of learning how to design and debug. Thus, in addition to studying students while they debugged challenging projects with multiple, overlapping errors, we also recorded reflections about their experiences both immediately at the end of the task and several weeks later after they had a chance to return to their designs and work through one additional e-textile project. Combined with the observations of think aloud interviews during debugging, students' reflections provide insights into their perspectives and emotions during and after the task.

3. Methods

3.1. Context and 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 the students from ethnic groups underrepresented in computing and 54% from economically disadvantaged families). The class was implementing the e-textiles unit (Y. B. Kafai et al., 2019) of the *Exploring Computer Science* curriculum (Goode et al., 2012) in which students make four projects (see Figure 1), exploring textile crafts, circuitry concepts such as electric polarity, and computing concepts such as sequences, conditionals, digital and analog inputs and outputs (Fields et al., 2018). Students were not intentionally taught any debugging strategies in the curriculum. Instead, students debugged within their own projects, sometimes with the help of their peers and the teacher, as noted in our earlier study (Fields et al., 2019).

During their third project, students were paired to work on a collaborative project – to make an interactive mural. During the project, students made their project after getting approval of their circuit and aesthetic designs from the teacher. The circuit designs had to fulfill the project requirement and had to be error free. Creating the project required students to craft the physical artifact, sew on the circuit elements, and plan and program the microcontroller to produce the desired outcome. Students were supported with a storyboarding worksheet where they could plan the light patterns and the interactions with two buttons that would cause it. They were also provided with a starter code with basic if-conditional statements and students were tasked with making functions for each light pattern and calling them within suitable if-statements. Students within pairs were

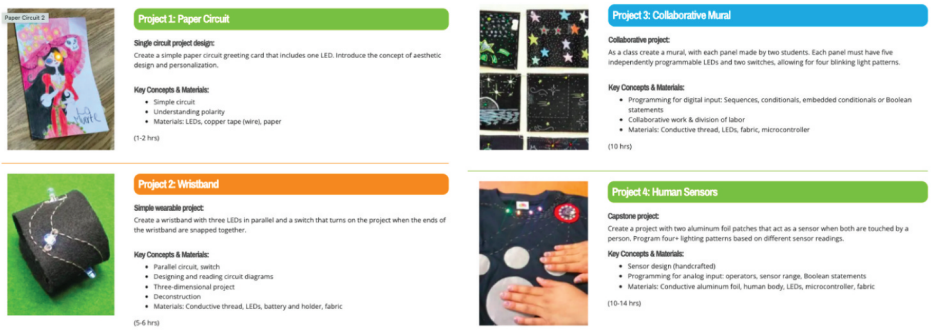

Figure 1. The four e-textiles curricular projects.

encouraged to take turns and distribute crafting and programming responsibilities so that both students in a pair got to engage in both aspects of the project. Students approach the teacher or their friends as and when they face errors during this process (Fields et al., 2019). The teacher would generally guide them to fix bugs by providing them clues about potential places of error and did not discuss any specific strategy for problem solving. Students here would have developed working relationships and collaborative problem-solving strategies that we then tried to capture during our think-aloud debugging exercise.

Following the third e-textile project, student pairs were invited to debug a researcher-designed buggy project (what we call a Debug-It) within a 55-min class period. We created four pairs of Debug-Its (eight in total; for more details see Table 1 and Figure 2); each pair of Debug-Its had identical errors that were set into different looking projects to avoid student pairs discussing solutions with one another (Jayathirtha et al., 2020). A total of 14 high school students (eight male and six female students) participated in the think-aloud sessions, with one of the pairs debugging two projects. We framed the debugging exercise as a student project from a different class and invited student pairs to think-aloud about what they were doing in order to make it easier for that (imaginary student) designer to later debug this project. Further, we provide representations of the problem space usually used in the context of physical computing such as circuit drawing, physical circuit, a copy of the code, code on the compiler, and a set of sewing and electronic supplies.

For this study, we analyzed the videos further to understand student interactions within pairs, use of different tools and representations in the problem space, and their perspectives of the activity. Video analysis was followed by an analysis of student interviews where they reflected on this experience immediately after debugging the given project (~15 min) and at the end of the unit (~15 min).

3.2. Debug-it designs

Each of the Debug-Its was designed to be one of the two types of projects in the e-textile curricular unit: either with two digital switches as inputs and LED lights as digital outputs similar to the third project (designed as either a 2D Captain America Shield or a 3D Tote

Table 1. Details of the different representations, tools, and Debug-Its provided to every student pair.

Representations provided	Tools available	Scenarios	Circuitry errors	Coding errors
-Intention Statement -Code on the paper and Google document -Circuit drawing -Physical artifact	-Computer with IDE -Alligator clips & multimeter -Sewing Supplies -Paper and Pencil -Previous project circuit drawing and/or code	2-Buttons (2D Shield or 3D Bag)	-Short circuits -Reverse polarity -Weak connections	-Inconsistent variable names -Missing declaration -Incorrect logical expressions
		Human Sensor (2D Bear or 3D Elephant)	-Short circuits -Reverse polarity -Weak connections	-Incorrect variable initialization -Wrong function call -Incorrect logical expressions

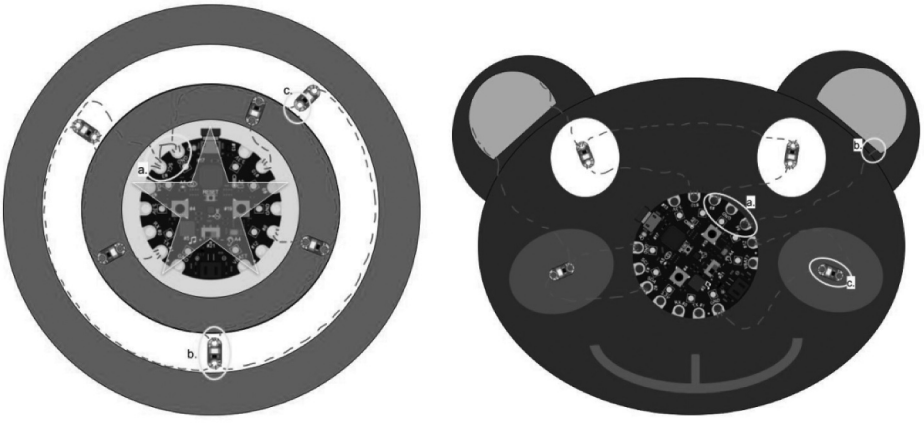

Figure 2. Representations of buggy artifacts that were presented to student pairs: two buttons Captain America Shield (left) with (a.) short connections, (b.) reverse polarity, and (c.) weak sewing; human sensor bear (right) with (a.) short connections at the back, (b.) weak connections, and (c.) reverse polarity.

Bag), or with a hand-crafted analog "squeeze" sensor and LED lights as digital outputs similar to the fourth project in the unit (designed as either a 2D Bear face or a 3D plush toy elephant). Both the projects involved physical interactive artifacts that would respond with different lighting patterns depending on the human action of button presses or sensor touches. However, they were not working as intended due to errors in the code and the circuitry as described in Table 1.

Each Debug-It 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 Debug-It had bugs such as loose connections causing short circuits and reverse polarity (see Figure 2), 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 on the paper and Google doc, approximately 30 lines long, included all three parts of a standard Arduino program – details related to the physical circuit connections in the first two sections (variable declaration and the setup() sections) and the behavior of the artifact (sequential statements in the loop() section). They were also accompanied by an intention statement which stated the desired outcome. And yet, both the representations include errors such as missing initializations, incorrect logical expressions and mismatched variables, based on challenges novices face while making Arduino-based projects (Booth et al., 2016; Jayathirtha et al., 2018).

3.3. Data collection

We collected both observational and interview data to answer questions about student strategies, practices, and perspectives. During the debugging activity, we videotaped all eight think-aloud debugging sessions, each roughly 45 min long. As described above, each pair was given a researcher-designed Debug-It, printed code with an intention statement, a "teacher-approved" circuit drawing (see Figure 3, right), a computer with the programming environment, sewing supplies, and paper (see Figure 3, left). We invited student pairs to debug the Debug-Its in a separate classroom during their class period, away from their busy classroom setting. We intentionally invite student pairs so that they verbalize their thinking during collaborative problem-solving and allow us to capture their strategies. We asked all student pairs to think aloud as they debugged the Debug-Its and video-recorded them. The researcher regularly probed students to verbalize their thoughts as per the think-aloud protocol (Ericsson & Simon, 1998). Unlike the traditional pair programming activities where the collaboration would be structured by assigning certain roles to participants and interchanging them, we let the pairs collaborate in ways that suited them so that we could study the emergent patterns of collaboration during this debugging activity.

In addition to capturing students thinking aloud and interactions while debugging, we recorded their reflections about their experiences at two instances: once, immediately at the end of the debugging activity and then again, later at the end of the unit,

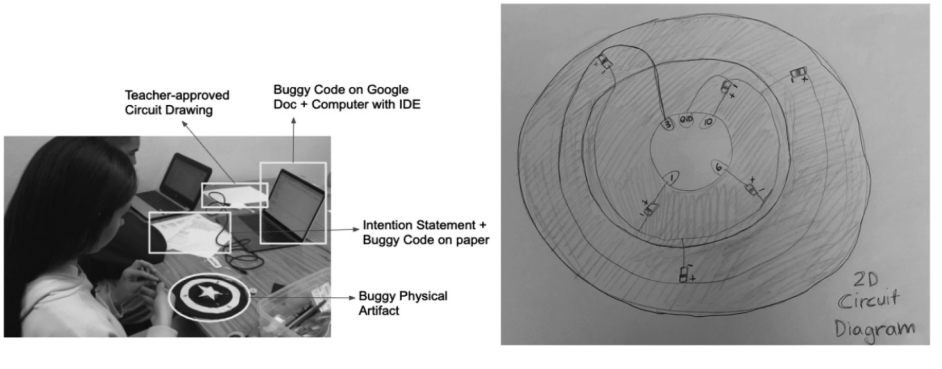

Figure 3. A snapshot of a pair in action depicting different tools and representations that were available to debug (left); the teacher-approved circuit drawing provided to student pairs (right).

which was about 2–3 weeks after the activity. Student pairs responded to prompts around the ease of debugging the given buggy projects and their takeaways from this experience.

3.4. Data analysis

We analyzed the videos and interviews separately to answer the research questions. First, we describe the two video analyses we conducted to understand students' debugging strategies as well as how their collaborative interactions influenced their debugging. Then, we describe the analysis of students' reflections in interviews.

For video analysis, we adopt a systematic and iterative approach to analyze videos in a deductive fashion (Derry et al., 2010). An earlier version of this paper (Jayathirtha et al., 2020) focused solely on students' debugging strategies. In this paper, we extend the analysis to deepen our understanding of collaboration and pay attention to the different elements of the problem-space such as tools and representations by paying attention to the interactions (inspired by Jordan and Henderson (1995)). To this end, 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).

To prepare data for analysis, as a first step, each observational video was indexed and sliced into 5-min 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 et al. (2010). The first two authors jointly generated these intermediate representations, consisting of annotated screen captures and transcripts of the videos, which then enabled discussions among researchers to observe patterns and generate codes.

3.4.1. Problem solving strategies and interactions

To capture student problem-solving strategies across the distributed system of learners, tools, and representations, all three authors collectively watched one of the videos and iteratively generated codes to identify 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 and Olson (1986); Katz and Anderson (1987); Vessey (1985)). We further looked at the usage of these strategies in relation to their relative success in identifying and solving bugs. At the same time, we created narrative descriptions and intermediate representations about the pair's debugging strategies until we reached a consensus with a set of codes. We drafted the coding scheme, then revisited it as a team to refine it and clarify details (see Appendix A). Then, two researchers independently coded each of the 5-min segments for one new video to capture phases of problem-solving, different strategies and reasoning. These were compared and discussed until we reached 100% consistency. Finally, the first author watched and analyzed the remaining seven videos, discussing findings in weekly consultation with the other two authors for 10 weeks to ensure rigor and consistency of interpretation.

3.4.2. Collaborative and distributed debugging interactions

In addition to capturing the debugging strategies for each of the 5-min episodes, we wrote extensive notes on all the interactions between the student pairs, the tools, and the representations in the space. Guided by Barron (2000)'s work on collaborative problem-solving, the first author analyzed interactions within one of the videos by writing extensive analytical descriptions on how the student pair established a shared problem space, joint attention, and mutuality (themes from Barron (2000)) for each of the five-minute segments. The first author continued to engage with the entire dataset, analyzing iteratively to generate codes, sub-codes, and their definitions (Appendix B). Another researcher independently coded the same video, following the codebook confirming the definitions and the sub-codes after which they analyzed the remaining videos. Further, the first author generated notes for each of the video segments to capture the different tools and representations that the student pairs interacted with for all eight videos. The first and the second authors met regularly over 10 weeks, with the third author joining occasionally to discuss notes and observations. These notes were jointly analyzed to identify emerging patterns in interactions between people, tools, and representations and then compared with codes for problem-solving strategies to understand the relationship between the distributed collaboration across people, tools and representations, and debugging.

3.4.3. Student reflections and perceptions

To answer the third research question, we conducted a two-phase grounded analysis of transcriptions of student interviews (Creswell & Poth, 2016). In the first phase, the first and the second authors independently analyzed all the transcripts to generate codes to capture student experience holistically, both along the lines of strategic problem solving and their perceptions of debugging as a learning activity. We then compare and iteratively discuss these codes to generate a final set of codes that capture all the student sentiments about debugging, writing up definitions with examples (see Appendix C for the themes and example quotes from the data). We elaborate on the details of each of these areas below, further considering how these interactions relate to the students' problem-solving as a whole.

4. Findings

Overall, student pairs adopt multimodal, system-wide problem-solving strategies while engaging in a mix of individual and collaborative approaches to debugging. Our analysis of student pair strategies and interactions highlighted the need for a holistic consideration of all the three aspects – problem-solving strategies, collaborative styles, and tool and representation usage – to better understand the debugging practices adopted by students. Furthermore, the analysis of videos of student pairs debugging Debug-Its revealed different debugging strategies, such as forward and backward reasoning, breadth-first and depth-first hypothesis generation, and iterative testing. *Forward reasoning* involves the search for faults that arise from actual written code or other given representations (Katz & Anderson, 1987). And, *backward reasoning* involves searches that start with the incorrect behavior of the program or the artifact that follows at least a round of testing the code and/or the artifact (Katz & Anderson, 1987). We note that pairs who adopt

a combination of forward and backward reasoning across the whole system, mostly as a result of iterative testing and breadth-first hypothesis generation, debugged more bugs than student pairs who were limited in their reasoning through the different phases of debugging.

Further, in their debugging strategies, novices demonstrated promising ways of navigating the multimodal problem-solving space and managing given representations and tools to better support debugging. Interacting with the overall problem space across code and circuitry-led students to locate and fix more bugs. Student pairs that checked code with the compiler earlier during the debugging process in addition to paying attention to the overall project intention statement solved more bugs than pairs that attended to these aspects much later during the process.

Irrespective of the number of bugs student pairs solved, student reflections shed light on the perspectives they developed about solving these problems and their collaborative nature, all pointing to the pedagogical potential of similar debugging activities. Below we describe each of these findings in more detail.

4.1. Adopting debugging strategies

Our first research question asked what strategies students adopted to debug e-textiles (Jayathirtha et al., 2020). Pairs who identified and solved half or more (3+) of the given bugs are in bold lettering in the left column in Figure 4 (pairs B1–4). 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)'s observation (i.e. no type of Debug-It led to automatic success or failure). Figure 4 also provides a visual for which debugging strategies were used, in which 5-min increments during the course of debugging for each group.

While all pairs used each of these strategies at some points, frequent adoption of strategies such as iterative testing and a combination of forward and backward reasoning across the broader system seemed to be associated with the more successful groups. For instance, less successful pairs cycled through the debugging strategies less frequently

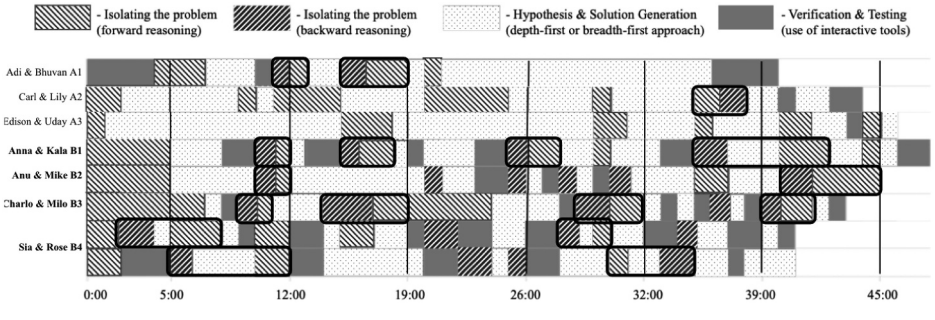

Figure 4. Visualization of debugging trajectories across eight student pairs (horizontal axis is minutes). The different patterns illustrate phases in debugging that pairs iterated through as given by the legend. Student pairs with bold lettered names (B1-B4) debugged half or more problems in their debug-its. Highlighted blocks indicate the mix of forward and backward reasoning that pairs adopted. Pair B4 solved two debug-its and therefore represented across two rows.

overall, spending significant time either isolating a single problem (i.e. a faulty light) and/or hypothesizing causes within a single system of the overall problem space (i.e. circuitry or code). Looking at the groups' narratives, it was clear that these groups did not consider the entire system of the Debug-It (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 pairs in Figure 4. We consider specific debugging strategies in more detail below.

Combined usage of forward and backward reasoning plays a strong role in isolating errors in Debug-Its, setting pairs up for opportunities to test and verify their ideas. This pattern is visible in the forward and backward reasoning blocks highlighted in Figure 4. All pairs spent a significant proportion of time, especially the first few minutes (except pairs A2 and A3), undertaking *forward reasoning* (see Figure 4). With respect to Debug-Its, this generally involves 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, pair B1 spent a significant proportion of their initial 10 min running exhaustive checks of the artifact and the given code in pursuit of errors. They started by checking to see if the wires were connected correctly, inspecting the positive and negative connections between each LED and the microcontroller. This was followed by a visual scan of the code to further check that the circuitry matched the programming. However, pairs B1–4 that further employ backward reasoning by generating causal explanations for unexpected behaviors fixed more issues.

Backward reasoning happens when students observed errors or inconsistencies while compiling the program or examining the runtime artifact behavior to isolate problems. As an example, pair B4's fault isolation, around 30 min 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. They noticed that the lights, although turned on, were not blinking as expected, which led them to brainstorm a few different reasons: from weak electric connections to suspecting certain parts of the program. While further observation of the artifact behavior helped them eliminate some of their hypotheses about weak connections, it also narrowed their investigation to looking at different aspects of the code such as the conditional block or other function calls. Once again, the pair fell back on forward reasoning while investigating different parts of the program, starting from serial scan of the code in the written order of statements as opposed to the execution order, further reaffirming the forward reasoning employed during this process of isolating errors within the program. 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, the latter allowing for an examination of the whole system as compared to the former. A depth-first approach involves generating solutions based only on limited hypotheses (Vessey, 1985). With Debug-Its, this often led to solutions not

rooted in actual problems, taking up substantial time for nonessential work. For instance, pair A3 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 2, right). This was not a malfunction in the project based on the intention for all the lights to blink at once, but the pair decided to redesign the circuit to connect each light to a separate microcontroller pin without testing. Their rapid solution generation from a depth-first approach precluded generation and testing of other probable causes for dysfunctional lights. In contrast, pair B4 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 provides many more opportunities to consider the working of the whole system, eliminate incorrect hypotheses, and generate a viable solution.

Yet another key consideration to debug the given Debug-Its is the presence and frequency of *verification cycles*. Pairs that use verification and testing to prune their hypotheses or dig for more information as part of their backward reasoning solve more issues compared to pairs that are verified later and less often. Verification in this context means examining the observed outcome on the physical artifact in comparison to the intended one, either while isolating sites of errors or after testing a probable solution (solid blocks in Figure 4, see highlighted blocks in Figure 5). A simple type of verification involves using the compiler to identify syntactic bugs in code or logical bugs as animated by the artifact behavior. For example, pair B3, 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 min. 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 by generating a fully functional program that lights up all the LEDs connected to the microcontroller. This early verification also led the pair to run cycles of verification repeatedly throughout their debugging session (see pair B3's row in Figure 5). 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

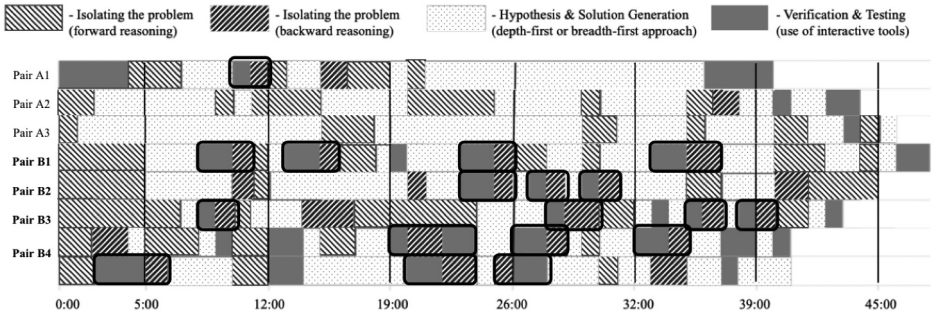

Figure 5. Cycles of verification combined with backward reasoning. Boldened blocks denote verification and testing immediately before or after instances of backward reasoning, occurring more frequently in groups that identified and solved more problems in the debug-its.

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.

4.2. Engaging in distributed problem-solving

Our second research question focused on how interactions between tools and representations shaped fixing the Debug-Its. Even within the tool usage, attention to the overall system stood out as playing a significant role in shaping debugging outcomes: student pairs who debugged more bugs also interacted with the interactive code on the computer in addition to paying attention to the overall intention statement (see Figure 6). However, despite students' limited experience with e-textiles and success debugging all the embedded bugs, their fluid interaction with tools and representations and their flexible collaborative styles demonstrated their developing abilities to recognize relationships between different parts of the computing e-textiles system. Below, we elaborate on the visualization that demonstrates students' interactions with tools and representations and 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).

Interacting with the code on the compiler early on and more often in relation to the physical artifact was important. For instance, most of the student pairs who debugged more than three bugs, pairs B1–4, started engaging with the code on the compiler earlier during the debugging process as visible within the first two segments in Figure 6. Although investigating the physical circuit on the project alone may have initially helped students get familiar with the problem space, this did not take them far. This is evident in the case of two pairs, A2 and A3 who struggled to find as many bugs despite continuous engagement with the physical artifact. On the other hand, when investigation of the physical circuit was coupled with interactive code on the compiler, students attended to the overall problem space spread across different modalities, comparing different representations of the physical computing project (interactive code and the physical circuit) and reasoning about the outcomes they noticed on the circuit by attempting to compile the code in order to upload it to the physical circuit. This also gave an opportunity for the pairs to connect the physical project to the power source and engage with it interactively

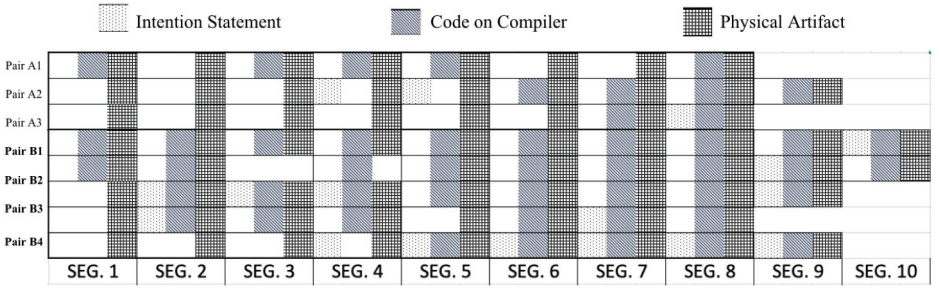

Figure 6. Student pairs and their interaction with the intention statement, code on compiler, and physical artifact; shaded regions indicate interaction and plain regions indicate no interaction in every 5-minute segment.

as opposed to examining the circuit without power (e.g. pair A2 investigating the physical project without connecting to the power source for the first 25 min of debugging). Further, alluding to the need for a systemic view of debugging, student pairs had to pay attention to the intention statement (i.e. a description of what the project was supposed to do), failing which impeded their ability to debug. Pairs B1–4 engaged with the intention statement within the first 10 min of the session (see Figure 6) as compared to pair A1 who never attended to the statement or pairs A2 and A3 who included the intention statement in their investigation later in their debugging process.

Despite the differences in outcomes, students' engagement with e-textiles during the unit surfaced their developing ability to navigate the multimodal problem-space. One aspect of the distributed system of e-textiles is the interrelatedness of the subsystems of circuitry, code, and craft (Y. 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 a power source) and comparing it to the circuit drawing (Figure 7, left), often tracing the circuitry lines with their fingers. Four groups 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 7, right). In this way, the students mapped the physical connections from microcontroller pins to LEDs and sensor patches with the corresponding variables defined and set up as OUTPUTs and INPUTs in the code, respectively, 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 (DesPortes & DiSalvo, 2019; Fields et al., 2016).

Students also introduced their own representations into the space, adding to their fluid navigation across the different tools and spaces, using prior projects 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 to bringing in prior project codes, students also introduced their own

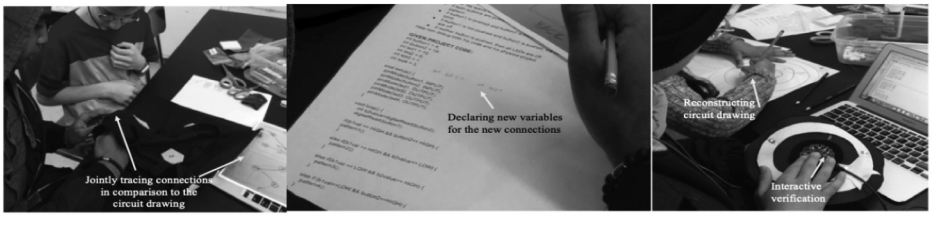

Figure 7. Pair A3 jointly comparing the physical circuit with the circuit diagram (left); Uday in pair A3 appending new lines of code to the given code (middle); Anu in pair B2 redrawing the circuit while Mike verifying the outcome (right).

representations to make sense of the buggy projects, including new notations and diagrams. For example, pair A3 first made changes to the code printed on paper (see Figure 7, middle). They chose to write their changes on paper before putting them into code as a way to coordinate their understanding (i.e. for shared visibility) as well as a way to keep track of the changes they made in case they needed to undo them. "Open" tools and representations such as these better served to create and sustain collaborative arrangements within the problem-solving space (Hutchins, 1995).

Further, student pairs drew on prior knowledge while evaluating circuitry and generating new diagrams. For example, pair B2 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 it to a version they thought would work better. Even though their new diagram included unnecessary changes (since they worked with a naive 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, 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, pair A1 concluded that syntax-error-free code was bug-free, which led them to ignore the logical inconsistencies in the code and instead spend their time hypothesizing issues only with circuitry. Furthermore, both interacting with given representations and generating their own afforded other interesting collaborative arrangements across people, as discussed below.

4.2.1. Collaboration across people

Student pairs demonstrated flexible collaborative arrangements, shifting how they jointly investigated the problem space during their class period. We observe that all the pairs except one (pair A1) mixed joint, collaborative and independent cooperative work throughout their debugging session. Having a mix of joint and individual work with high levels of mutuality worked best for pairs while joint attention alone throughout the debugging task led to pairs debugging fewer bugs (see Figure 8). In most cases, student pairs established a *shared problem space*, i.e. establishing a shared orientation towards problem-solving (Barron, 2000) by jointly investigating the buggy project initially (except B2 as seen in the corresponding row in Figure 8) after which they continue to sustain the shared space by adopting a combination of individual and collaborative work. Nearly all pairs (7 out of 8) worked collaboratively to problem solve for a majority of the time and rarely divide the labor between circuit and coding, using a divide-and-conquer collaborative strategy to analyze subsets of the problem space (Figure 7, right). For instance, pair A3 jointly worked on establishing the problem space and isolating issues by investigating the physical artifact and the circuit drawing together for most of the debugging session (30 of

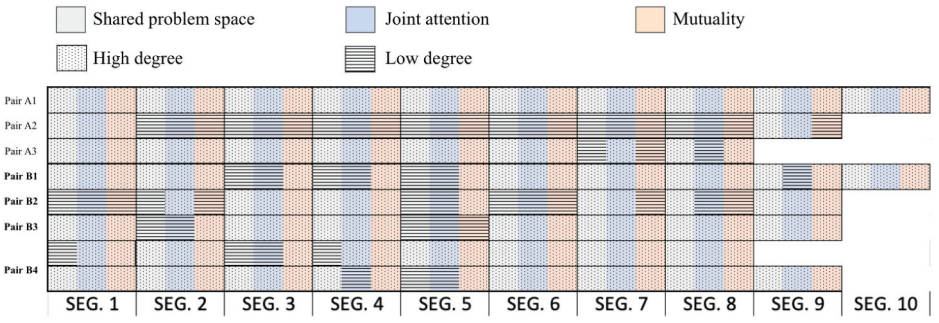

Figure 8. Collaborative arrangement visualization across the eight pairs.

the 40 min) while they divided further investigation between themselves for the last 10 min. Edison (one of the two students in A3) continued to examine the code, making debugging decisions independently, while Uday (the other student in A3) threaded the needle and started sewing their agreed upon solution to a circuitry problem. Dividing up circuitry and code-related work due to logistical constraints and convenience is similar to some of the ways students have traditionally divided e-textiles tasks into circuitry and coding while designing and making e-textiles artifacts (Lui et al., 2019).

Except for short instances of divide-and-conquer approaches, pairs *jointly attended* to particular aspects of the problem space at key moments during debugging (Barron, 2000), 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 7, left). Frequently, pairs moved between these two modes depending on the task at hand. For example, pair B4 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 where they would fluidly move between investigating the code and the circuitry as they connected these different parts to debug the given Debug-It.

In sum, joint debugging within a distributed system such as e-textiles 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. The multi-modal and multi-representational complexities of e-textiles as a physical computing system further demands collaboration very different from the highly structured and pre-designed pair programming arrangements that have been extensively studied within on-screen programming contexts (Williams et al., 2000). Similarly, the task of debugging pushes students to adopt fluid collaborative arrangements unlike what is traditionally observed in the context of making e-textiles projects where students adopt more cooperative rather than collaborative working strategies (Lui et al., 2019). We will further highlight the ramifications of these observations in the discussion section below.

4.3. Developing positive perspectives on debugging

While our video analysis pointed at the complexity and student difficulty associated with debugging physical computing projects, thematic analysis of post-activity reflections revealed students' generally positive perspectives on what they gained from this activity, including a variety of problem-solving-related practices, more proactive stances toward collaborating in debugging, and appreciating the opportunity to debug projects not authored by themselves.

4.3.1. On learning problem-solving strategies

Students explicitly articulated some of the problem-solving strategies that they adopted while debugging pre-designed buggy projects. A majority of the students (11 out of 12) thought of debugging someone else's buggy project as a *valuable* learning opportunity for a few distinct reasons: for providing an opportunity to generally learn about problem-solving within the e-textiles context, for being able to recognize specific problem-solving strategies that could be employed while debugging or to help them more successfully debug their own projects later in the class. Some of the students reflected on this exercise as helping them learn better about debugging e-textiles projects. For instance, Anu discussed this as a "great way to learn about problems you haven't seen before" and Adi shared that this experience helped him put his e-textiles-related knowledge such as Arduino-programming and crafting to practice.

These reflections around problem-solving strategies further tied closely to the practices and strategies we observe in videos. For instance, Uday commented on the time it takes to comprehend someone else's project before fixing the errors in it: "you have to figure out how they thought of doing it first just to see what was supposed to happen and then try to take it out and then do it over" (Uday and Edison spent 20 out of 40 min in forward reasoning, forward as they visually scanning the artifact and the code before targeted testing). In addition, Edison commented how he "had to go backwards or had to go through and figure out what was wrong," demonstrating the two kinds of reasoning – backward or forward – that students employed throughout their time debugging.

Some students went further and articulated more general *problem-solving strategies* that they explored as a consequence of participating in this debugging activity. Students reported *learning* specific debugging strategies through this structured debugging experience, expressing in detail debugging techniques and strategies to apply to new scenarios of debugging. These strategies included, for example, "always checking" the mapping between the circuit design and the code to look for obvious errors, taking time to study a project and its intended function before fixing it, and taking a series of steps to first find and then fix mistakes. As an example, Mike articulated his group's overall takeaway that this process "helped teach how to find and fix your mistakes. So, it shows the steps you can take to find your mistakes, and then the steps you should take to fix those mistakes." In sum, although students did not uniformly apply these strategies during the timed debugging task, they were able to articulate and use some of the strategies after the fact.

Furthermore, a few students reported directly *applying improved strategies* of debugging to their later project in the curricular unit. For example, Milo said "when I went back to my project, my project wasn't working... Before, I wasn't really sure

what was wrong with my project. This helped me a little bit, fixing my project," highlighting the role of such a debugging experience in better supporting them to handle bugs in their own projects. As Milo and a few other students mentioned, they attributed the extended learning to their exposure to a broader variety of bugs and debugging experiences outside of what they would have experienced within their own projects. As Anu mentioned, this debugging exercise provided an opportunity to explore a variety of "mistakes and correct it ... and, remember for next time." In sum, this set aside debugging experience moved novices towards more expert practices noted as key while debugging, even for pairs who struggled to solve the Debug-lts (Jonassen & Hung, 2006).

4.3.2. On collaborative debugging

Intriguingly, students expressed that the paired debugging experience positively shaped their perspectives on collaboration and informed their future collaborative arrangements in the class. Students said the experience *lowered barriers to collaboration*; they felt better able to give help and less worry about asking for help. As discussed above, the collaborative nature of this debugging activity required students to lean on each other as they navigated the space and problem-solved; and, this gave opportunities for learners to develop a certain understanding about collaboration itself, specifically while debugging. As Mike expressed, this activity helped him realize that "you have to trust your partner to be looking for the mistakes," pointing to how the division of labor while debugging demanded that they develop trust that their partner is working towards the required goal while debugging. Solving a pre-designed buggy project disguised as another student's buggy project allowed a few students like Sia to feel sympathy for others for their mistakes. This was evident in how Sia reasoned the cause for certain mistakes such as reversed polarity or other sewing mistakes to happen, since she "probably would have done the same to be honest".

In addition to learning about their partners, students reported developing a better understanding about themselves receiving help while debugging. For instance, Anna commented that this activity showed her that "it's okay to take help from others" while debugging, yet another acknowledging the distributed nature of this activity. While on-screen programming environments may attempt to promote joint problem-solving through structured arrangements such as pair programming, debugging physical computing systems such as e-textiles authored by others seems to afford several opportunities for student pairs to jointly problem solve and shift their perspectives about collaboration within these activities. Some students also pointed out how they can leverage collaborative arrangements to better debug by adopting certain strategies. For instance, similar to other pairs, Milo and Charlo explained how they complemented each other by one focusing on the program while other attending to the "cross reference" each other's mistakes, hinting at how the pair can take the help of extra hands and eyes to run consistency checks as they verified information across different representations. These reflections parallel our findings about distributed collaboration in debugging e-textiles.

4.3.3. On positive affect

In addition to reflecting on their problem-solving strategies and collaborative debugging experiences, students reported *positive emotions* about the opportunity to debug others' projects. This is surprising especially since no pair successfully completed debugging the given buggy projects, and three groups did not even come close to identifying, much less solving all of the bugs. Despite this, most of the pairs shared that it was "fun" doing this activity, going through someone else's buggy project to help them fix it. As Adi shared, some of the pairs thought the activity was enjoyable because it was an opportunity outside of their own projects to apply what they have learned coding and crafting their own projects. This is similar to how Anu thought of this opportunity to strike a good balance of being fun and frustrating and providing a unique opportunity to learn about errors outside of what they personally encounter in their own projects. This was reflective in how she said that "seeing where someone went wrong or seeing where yourself went wrong is an opportunity to learn".

In addition to reporting the enjoyable aspect of this experience, a few students also shared how this debugging activity made them more confident. Sia shared that fixing these buggy projects "made her feel smart" for she was able to make a completely unfamiliar project work. Mike further compared this to how they support their friends to debug their projects in their classes. Edison's mention of this activity as helping him help others by developing the ability to "work backwards" further added to how students found this exercise to be productive despite their partial success at realizing a fully functional artifact. Overall, there were positive sentiments that were shared, which opens possibilities for making these kinds of activities a part of classroom activity.

5. Discussion

In sum, our analyses of high school student pairs' debugging strategies, practices, and perspectives illuminated: (a.) different approaches that novices took on to fix the buggy project, (b.) how learner pairs navigated the multimodal space that involved representations and tools across the screen and the physical spaces, (c.) collaborative and individual approaches while establishing and sustaining a shared problem-space, and (d.) productive perspective learners developed from their engagement. Unlike previous debugging research that had mostly involved individual students debugging on-screen programs, using physical computing in this study expanded the problem-space and brought attention to the different student practices outside screens (McCauley et al., 2008). The distributed problem space in the case of physical computing allowed students to adjust their problem-solving practices to accommodate other people, as well as tools and representations across the digital and physical spaces. Further, reflections about the debugging exercise allowed for a more holistic examination of debugging by including learner perspectives. Below, we first consider the limitations of this research study, then discuss some of the unique challenges and opportunities available in debugging physical computing systems, calling for more extensive research on strategies to support learners through pedagogical and tool designs, and further research in debugging in physical computing systems.

5.1. Limitations

The findings reported above are not without limitations. First, our in-depth video analysis of seven student pairs' debugging was exploratory in terms of both data collection and analysis. It only covered a subset of students in the class since most others either did not consent to be a part of the research or declined participation in order to work on their projects during this class period. Our data preparation and analysis processes could have included measures such as inter-rater reliability if there were already coding schemes available for data of this nature. Second, as a pull-out study, the scenario was somewhat artificial, with limited time and a project with many bugs (albeit bugs chosen based on what students struggle with). As such, the think-aloud protocol and Debug-It scenario provide affordances for capturing student strategies and collaborative arrangements, which should be seen as a complement to more naturalistic settings where students solve their own bugs during design (e.g. Morales-Navarro et al. (2021)). Third, as with qualitative research in general, this study was also situated within a particular context, where student pairs had experience with designing e-textiles projects; it does not necessarily generalize to other contexts. In particular, students had established prior working relationships before the collaborative work in this study. Further, they were two-thirds into the 10–12-week-long e-textiles unit, which implies familiarity with the materials and problem-solving. This is very different from a context where students, new to e-textiles, will be presented with buggy projects to debug. However, none of these limitations invalidate or minimize the importance of the findings in relation to the strategies, collaborative arrangements, and perspectives of the seven student pairs in this introductory computing program.

5.2. Implications of distributed debugging in physical computing

One goal of this study was to update the field on the challenges and processes students use in debugging within newer (especially physical) programming environments where programming is distributed across people, materials, tools, and representations. Indeed, we observed distinct problem-solving practices that were specific to the *distributed nature* of the physical computing task, where problems were spread across different modalities. Rather than move in a linear mode through the previously-established problem solving practices to locate faults, generate hypotheses and solutions, and run system-wide tests, in this physical computing context, students move *iteratively* through these processes, often with success. 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 on-screen programs. Unlike prior studies' findings, debugging within the context of physical computing requires students to problem-solve at the intersection of programs and circuits, which lead to cycles of explorations across the multimodal problem space, coordinating strategies across people, tools, and representations. This points to the need to value these less linear, *more iterative processes* in debugging while at the same time developing means to support students' in developing more sophisticated strategies for debugging physical computing projects. This analysis emphasizes the need to adopt a distributed cognition in physical computing contexts.

The struggle to coordinate different aspects of e-textiles surfaced in ways some pairs failed to account for the whole system during the debugging process, a task that required moving between multiple representations from the computer screen, the problem statement, the circuitry diagram, and of course the actual physical project, sometimes creating new representations to help with the problem solving. This implies the need to equip students with debugging strategies that will be more effective for these types of distributed tasks. One way to do this is to include debugging exercises with bugs across the different domains i.e. circuitry, programming, and their intersections. Student reflections in this study suggest that such a task was enjoyable and highly productive for learning, pointing to the potential of similar pre-designed buggy contexts for diagnostic or formative exercises. Additional activities in e-textiles and in other physical computing contexts should be developed and studied in regard to their efficacy for supporting more strategically distributed debugging practices.

5.3. Debugging in collaborative contexts

Our study contributes new insights into the role of collaboration in debugging in a physical computing context beyond the contributions of highly structured pair programming scenarios. When analyzed through distributed cognition and collaborative problem-solving lenses, student work while debugging e-textile projects surfaced new collaborative practices within this multimodal problem-solving space. Students took advantage of additional hands and eyes to establish joint attention at different parts of the shared problem-space, while the problem-space distributed between the computer screen, physical circuit, and on-paper representations allows pairs to adopt synchronous individual debugging processes and develop a shared understanding of the problem-space. Student reflections mirrored their practices, demonstrating more openness to and seeing the value of collaborating on debugging. The implications for students being more open to collaboration – whether giving and receiving help or sharing in debugging – have implications for classroom environments, both in designing for collaboration with distributed tools, supporting peer pedagogy, and providing another type of productive collaboration in addition to the more structured pair programming. Future research on collaborative debugging in computer science education, therefore, needs to explore these emergent collaborative arrangements and their relationship to the learning debugging.

5.4. Toward a more holistic view of debugging

Taking a more holistic perspective of debugging provides insights into not only cognitive and problem-solving processes but also the role of tools and representations, social interactions, and emotions in debugging. This points to the limits of only taking a cognitive perspective of problem-solving, such as analyzing solely for strategies (Jayathirtha et al., 2020). Looking at problem-solving in relation to collaborative dynamics and multiple representations of the problem space indicates the role of distributed cognition and social dimensions in debugging. Further, attending to students' emotions and subsequent reported practices in debugging provided insights into their positive emotions (even when they did not complete the task) and

productive attitudes toward giving and receiving help. This more holistic perspective not only provided insights into novice students' debugging practices, but they also supported the students themselves in their future design and debugging work. More work needs to be done on the role of debugging experiences like these in student development.

In contrast to giving students small, specific problems to debug, this study also demonstrates a role for providing complicated, intersecting problems, even where students largely fail to attain all the solutions. This aligns with other studies of similar pull-out debugging scenarios where the experience is collaborative and time-limited with multiple, intersecting problems (e.g. Fields et al. (2016)), where the experience can be a formative assessment: revealing students' skills while also helping them to learn productively. More recent work in this area is probing the role not just of having students debug such projects but also of designing such buggy projects for others. Similar (if not stronger) findings about students' sense of capability, their increased willingness to reach out to others to give and receive support, and even the development of growth mindset practices can accompany improved debugging skills in such scenarios (Fields et al., 2018; Morales-Navarro et al., 2021). Capturing student reflections in this study demonstrates the need for research on debugging to consistently include socioemotional and collaborative perspectives alongside the more commonly researched strategic processes (e.g. Dahn et al. (2020)) and to consider the role these experiences have in learning as well as assessing.

5.5. Developing support and tools for learning debugging

As observed and articulated by students in this study, debugging pre-designed buggy projects provided a very different perspective on problem solving and debugging, pushing students to pursue new strategies rather than debugging their own projects. While the debugging experience helped students articulate different strategies, they also recognized the importance of learning how to tap others for support, especially in a context like interest-driven projects (Fields et al., 2018) where students are developing distributed expertise. All of these point to the promise of similar exercises serving as pedagogical activities. Creating debugging activities like those in this paper can be a means of providing students more opportunities for debugging without the pressure of successfully realizing a personally relevant project. Prior research on the less challenging Debug-Its in e-textiles reinforces the idea that students 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 Debug-Its 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).

Another implication of this study is the need to develop new tools specific to physical computing contexts like e-textiles, especially given the significance of tools and representations in students' debugging. Historically, many powerful tools have been specifically developed to support debugging. Yet current compilers, debuggers and multimeters are mostly designed for individual, localized testing of code or circuitry within a single sub-system (e.g. code and circuitry) but not across the whole system at once. Instead, we need new tools that help students navigate across multiple subsystems, such as code, circuitry, and spatially distributed surfaces of physical computing artifacts (Schneider, 2022), and provide students with required ways of understanding the state of the system and its transition during runtime. This is in line with the call for such specific tools by DesPortes and DiSalvo (2019) that can support students testing their own naive conceptions while debugging. Specific tools that make the whole system state visible and help students check across both circuitry and code are one idea for new tools, though there may be many more (Hill et al., 2021). Such tools can support students in isolating simpler problems within only programming (such as syntactic errors) or circuitry (such as short or wrong connections), thereby helping learners focus on the harder problems at the intersection of circuits and programs that predominantly show up during program execution.

6. Conclusion

This study expands on previous debugging education research by identifying emergent practices by novice students in collaboratively debugging physical computing projects, specifically e-textiles with strategically chosen multiple intersecting problems. This study expands our understanding of debugging, drawing attention to the distributed problem space that requires attending to multiple tools and representations *across* digital and physical spaces. This revealed the productivity of iterative and cyclical rather than linear debugging strategies in these problems. Further, productive collaborative strategies emerge during and after the debugging tasks that provide a complement to pair debugging research in demonstrating the benefits and challenges in coordinating across people. Finally, taking a more holistic lens on debugging that included both cognitive, social, and affective elements, our study demonstrated that the isolated debugging task, situated within a larger unit on e-textiles, helped students develop positive dispositions toward debugging. More studies are needed to identify emergent debugging practices and challenges in other physical computing domains.

Acknowledgments

This study was funded by the National Science Foundation (#1742140). We thank the teacher and the students who participated in this study. Special thanks to Debora Lui, who partnered during data analysis.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the National Science Foundation [1742140].

References

Anwar, T., Jimenez, A., Bin Najeeb, A., Upadhyaya, B., & McGill, M. M. (2020, August 10–12). Exploring the enacted computing curriculum in K-12 schools in South Asia: Bangladesh, Nepal, Pakistan, and Sri Lanka. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, New Zealand, ACM, NewYork, NY, USA. (pp.79–90). https://doi.org/10.1145/3372782.3406251

Barron, B. (2000). Achieving coordination in collaborative problem-solving groups. *Journal of the Learning Sciences*, *9*(4), 403–436. https://doi.org/10.1207/S15327809JLS0904_2

Booth, T., Stumpf, S., Bird, J., & Jones, S. (2016, May 07–12). 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*, San Jose, CA, USA. (pp.3485–3497). ACM.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 Annual meeting of the American Educational Research Association*, Vancouver, Canada (Vol. 1, p. 25).

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. https://doi.org/10.1080/10749039.2014.939762

Buechley, L., Peppler, K., Eisenberg, M., & Yasmin, K, (Eds.). (2013). Textile messages: Dispatches from the world of E-Textiles and education. In *New literacies and digital epistemologies* (Vol. 62, p. 10006). Peter Lang Publishing Group.

Campe, S., Denner, J., Green, E., & Torres, D. (2020). Pair programming in middle school: Variations in interactions and behaviors. *Computer Science Education*, *30*(1), 22–46. https://doi.org/10.1080/08993408.2019.1648119

Creswell, J. W., & Poth, C. N. (2016). *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications.

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

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

DeLiema, D., & Dahn, M. (2020). Envisioning debugging cultures at the intersection of emotion, problem solving, and identity. In D. Weintrop, G. W. Choi, A. Maltese, & M. Tissenbaum (organizers), what does computer science and maker education look like in 2030? In M. Gresalfi & I. S. Horn (Eds.), The Interdisciplinarity of the Learning Sciences, 14th *International Conference of the Learning Sciences (ICLS)*, Volume 2, 1519–1524.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education*, *46*(3), 277–296. https://doi.org/10.1080/15391523.2014.888272

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

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*, ACM, Toronto, ON, CA. (pp.219–227). http://dx.doi.org/10.1145/3291279.3339427 .

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. https://doi.org/10.1207/s15327884mca0503_3

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. K. Lund, G. Niccolai, E. Lavoué, C. Hmelo-Silver, G. Gweon, & M. Baker (Eds.) In *the Proceedings of the 13th International Conference on Computer Supported Collaborative Learning*, Lyon, France. (pp.472–479).

Fields, D. Jayathirtha, G. & Kafai, Y.(2019). Bugs as a nexus for emergent peer collaborations: Contextual and classroom supports for solving problems in electronic textiles. *Computer Supported Collaborative Learning (CSCL'19)*. *19*, 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. https://doi.org/10.1080/10665684.2018.1436998

Fields, D. A., Searle, K. A., & Kafai, Y. B. (2016, October 14-16). Deconstruction kits for learning: Students' collaborative debugging of electronic textile designs. In *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education*, Stanford, CA, USA. (pp. 82–85). ACM.

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: The exploring computer science program. *ACM Inroads*, *3*(2), 47–53. https://doi.org/10.1145/2189835.2189851

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Gugerty, L., & Olson, G. (1986). Debugging by skilled and novice programmers. *ACM SIGCHI Bulletin*, *17*(4), 171–174. https://doi.org/10.1145/22339.22367

Hill, C., Schneider, M., Eisenberg, A., & Gross, M. D. (2021, February 14–17). The threadboard: Designing an e-textile rapid prototyping board. In *Proceedings of the Fifteenth International Conference on Tangible, Embedded, and Embodied Interaction*, Salzburg, Austria. (pp.1–7).

Hodges, S., Sentance, S., Finney, J., & Ball, T. (2020). Physical computing: A key element of modern computer science education. *Computer*, *53*(4), 20–30. https://doi.org/10.1109/MC.2019.2935058

Horn, M. S. (2018). Tangible interaction and cultural forms: Supporting learning in informal environments. *Journal of the Learning Sciences*, *27*(4), 632–665. https://doi.org/10.1080/10508406.2018.1468259

Hubweiser, P., Armoni, M., Brinda, T., Dagiene, V., Diethelm, I., Giannakos, M. N., Knobelsdorf, M., Magenheim, J., Mittermeir, G., & Schubert, S. (2011). Computer science/informatics in secondary education. In Proceedings of the *ITiCSE-WGR '11*, ACM, New York, NY, USA, 19–38.

Hutchins, E. (1995). How a cockpit remembers its speeds. *Cognitive Science*, *19*(3), 265–288. https://doi.org/10.1207/s15516709cog1903_1

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

Jayathirtha, G., Fields, D., & Kafai, Y. (2020) Pair debugging of electronic textiles projects: Analyzing think-aloud protocols for high school students' strategies and practices while problem solving. In *Proceedings of International Society of the Learning Sciences (ISLS) 2020* (pp.1047–1054).

Jonassen, D. H., & Hung, W. (2006). Learning to troubleshoot: A new theory-based design architecture. *Educational Psychology Review*, *18*(1), 77–114. https://doi.org/10.1007/s10648-006-9001-8

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

Kafai, Y. B., Fields, D. A., Lui, D. A., Walker, J. T., Shaw, M. S., Jayathirtha, G., & Giang, M. T. (2019, February 27). 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*, Minneapolis, MN, USA. (pp. 1176–1182). https://doi.org/10.1145/3287324.3287426

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. https://doi.org/10.17763/haer.84.4.46m7372370214783

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

Katz, I. R., & Anderson, J. R. (1987). Debugging: An analysis of bug-location strategies. *Human-Computer Interaction*, *3*(4), 351–399. https://doi.org/10.1207/s15327051hci0304_2

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, *37* (2), 83–137. https://doi.org/10.1145/1089733.1089734

Kummerfeld, S. K., & Kay, J. (2003). The neglected battle fields of syntax errors. *Proceedings of the fifth Australasian conference on Computing education (ACE2003)*, Adelaide, Australia, *20*, 105–111.

Lewis, C. M. (2010, March 10–13). How programming environment shapes perception, learning and goals: Logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*, Milwaukee, Wisconsin, USA. (pp. 346–350).

Lewis, C. M. (2012, September 9–11). The importance of students' attention to program state: A case study of debugging behavior. In *Proceedings of the ninth annual international conference on International computing education research*, Auckland, New Zealand. (pp. 127–134).

Liebenberg, J., Mentz, E., & Breed, B. (2012). Pair programming and secondary school girls' enjoyment of programming and the subject Information Technology (IT). *Computer Science Education*, *22*(3), 219–236. https://doi.org/10.1080/08993408.2012.713180

Litts, B. K., Searle, K. A., Kafai, Y. B., & Lewis, W. E. (2021). Examining the materiality and spatiality of design scaffolds in computational making. *International Journal of Child-Computer Interaction*, *30*, 100295. https://doi.org/10.1016/j.ijcci.2021.100295

Lowe, T. (2019). Debugging: The key to unlocking the mind of a novice programmer? In *2019 IEEE Frontiers in Education Conference (FIE)* (pp. 1–9). IEEE.

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), 1–30. https://doi.org/10.1080/08993408.2019.1682378

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 1–15. https://doi.org/10.1145/1868358.1868363

Margolis, J., Estrella, R., Goode, J., Holme, J. J., & Nao, K. (2017). *Stuck in the shallow end, updated edition: Education, race, and computing*. MIT press.

Martinez, P., Lopez, J., Rodríguez, F. J., Wiggins, J. B., & Boyer, K. E. (2020, March 11–14). Novice debugging in block-based and hybrid environments. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, Portland, OR, USA. (pp. 1291–1291).

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

Morales-Navarro, L., Fields, D. A., & Kafai, Y. B. (2021). Growing mindsets: Debugging by design to promote students' growth mindset practices in computer science class. In *Proceedings of the 15th International Conference of the Learning Sciences-ICLS 2021*.

Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: The good, the bad, and the quirky–a qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin*, *40*(1), 163–167. https://doi.org/10.1145/1352322.1352191

Nakajima, T. M., & Goode, J. (2019). Transformative learning for computer science teachers: Examining how educators learn e-textiles in professional development. *Teaching and Teacher Education*, *85*, 148–159. https://doi.org/10.1016/j.tate.2019.05.004

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, *2*(1), 37–55. https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL

Przybylla, M., & Romeike, R. (2014). Physical computing and its scope–towards a constructionist computer science curriculum with physical computing. *Informatics in Education*, *13*(2), 241–254. https://doi.org/10.15388/infedu.2014.14

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Schneider, M. (2022). Scaffolding the debugging process in physical computing with circuit check. In A. Weinberger, C. Wenli, D. Hernández-Leo, & C. Bodong (Eds.), *Proceedings of Computer Supported Collaborative Learning (CSCL) 2022* (pp. 391–394). International Society of the Learning Sciences.

Scott, M. J., & Ghinea, G. (2013). On the domain-specificity of mindsets: The relationship between aptitude beliefs and programming practice. *IEEE Transactions on Education*, *57*(3), 169–174. https://doi.org/10.1109/TE.2013.2288700

Sengupta, P., Dickes, A., & Farris, A. (2018). Toward a phenomenology of computational thinking in K-12 STEM education. In M. S. Khine, (Ed.), *Computational Thinking in the STEM Disciplines: Foundations and Research Highlights* (pp. 49–72). Springer.

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

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

Tofel-Grehl, C., Fields, D., Searle, K., Maahs-Fladung, C., Feldon, D., Gu, G., & Sun, C. (2017). Electrifying engagement in middle school science class: Improving student interest through e-textiles. *Journal of Science Education and Technology*, *26*(4), 406–417. https://doi.org/10.1007/s10956-017-9688-y

Tubaishat, A. (2001). A knowledge base for program debugging. In *Proceedings ACS/IEEE International Conference on Computer Systems and Applications*, Beirut, Lebanon. (pp. 321–327). IEEE.

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. https://doi.org/10.1002/jaal.612

Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies*, *23*(5), 459–494. https://doi.org/10.1016/S0020-7373(85)80054-7

Wagh, A., Gravel, B., & Tucker-Raymond, E. (2017, October). The role of computational thinking practices in making: How beginning youth makers encounter & appropriate CT practices in making. In *Proceedings of FabLearn 2017 conference (FABLEARN 2017)*, Palo Alto, California, USA. (pp.1–8). https://doi.org/10.1145/3141798.3141808

Whalley, J., Settle, A., & Luxton-Reilly, A. (2021, March 13–20). Novice reflections on debugging. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*, New York, NY, USA. (pp. 73–79). https://doi.org/10.1145/3408877.3432374

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software*, *17*(4), 19–25. https://doi.org/10.1109/52.854064

Zhao, Z. (2014). *Solo Programming Vs Pair Programming: Strategies for Debugging* [Doctoral dissertation]. University of Georgia.

Appendices

Appendix A. A snapshot of the Codebook that was developed and used in analysis of video examples

Data Source	Themes/Codes & Definitions	Episodes from data
Annotated Video Transcripts	Isolating the issue: Forward Reasoning Scanning the problem space such as reading code from the top or scanning the physical circuit or circuit drawing to find bugs	The pair starts with understanding the circuit by mapping the circuit drawing to the physical artifact (Milo with the drawing and Corey with the artifact). Milo then moves to looking at the code on the Google Doc, making changes to the variable name as they both compare it to the physical artifact. Milo seemed to be searching exhaustively the sewn artifact for any obvious, visible errors in sewing; he also later pulled out his previous project code to compare and do pattern matching.
	Isolating the issue: Backward Reasoning Identifying an incorrect functioning such as lights not working and moving towards generating a solution	The pair recognize that the white lights stopped working after Sia touched it once. Rose opens the serial monitor to see "what numbers" they get. They start hypothesizing [different reasons].
	Hypothesis Generation Reasoning an error captured, either through forward or backward reasoning	The pair reason that the lights would not have been blinking if the connections were weak, thereby dismissing the need to restitch. Sia is picking on different parts of the code for why the serial monitor may be showing 1 (should it be 9600?). They also think it could be the lighting pattern, Sia starts to read the functions in the code w.r.t. the intention statement. Rose is going line-by-line, comparing her previous code with the given and pattern matching.
	Solution Generation Thinking aloud potential fixes for the bug or error noted	The pair thinks they will have to sew the connection to the negative patch to make it better instead of a single stitch.
	Testing and/or Verification Running a test across the problem space to either identify the functionality or to ensure the fix worked	The pair connected the artifact to the computer after Sia stitched the positive connection, and were happy to see the white cheek light glowing. Sia starts to test the touch sensor to see the behavior of the artifact.

Appendix B. A snapshot of the Codebook that was developed and used in analysis of video for collaboration

Data Source	Themes/Codes & Definitions	Episodes from data
Annotated Video Transcripts	Shared problem space Establishing a shared orientation towards problem-solving	**High**: Lily and Carl inspect the physical artifact by running exhaustive checks of the connections between the lights and the microcontroller. Then they split the process into half as Lily suggests Carl to check the code on the paper while she unstitches the "wrong" connections. Here, the students determine the path towards the solution jointly.

(*Continued*)

(Continued).

Data Source	Themes/Codes & Definitions	Episodes from data
		Low: Although Mike and Anu jointly worked on the task of declaring new variables for the two LEDs, they never discussed the strategy. Instead, Mike's strategy was adopted by the pair while Anu was directed to make the required changes in the code and the sewn circuit.
	Joint attention Degree to which attention is jointly focused on particular aspects of the problem space during critical moments	**High**: Milo and Charlo attempt to understand the circuit by mapping the circuit drawing to the physical artifact, with both of them looking at the same aspects of the problem space consistently. **Low**: Milo then moves to looking at the code on the Google Doc, making changes to the variable name as they both compare it to the physical artifact while Charlo is scanning the physical artifact.
	Mutuality Reciprocity with potential for both students to contribute to the debugging process	**High**: Milo is scanning the circuit drawing exhaustively which Charlo then takes over to see the same way (checking the circuit but through an active representation). **Low**: Anu is on the computer, looking at the code, but Mike tells that he thinks the code is correct but the sewing is off (Anu had worked on resewing a part of the circuit earlier)

Appendix C. A snapshot of the Codebook that was developed and used in analysis of interview transcripts examples

Data Source	Themes/Codes	Quotes from data
Student Reflection Interview Transcripts	Learning to strategically problem solve and reason through mistakes	"Just double check it"
		"this time I had to go backwards or I had
		to go through and figure out what was
		wrong"
		"it shows the steps you can take to find
		your mistakes, and then the steps you
		should take to fix those mistakes"
	Learning about collaborative arrangements	"it showed me that it's okay to get help
		from other people. You don't need to do
		it yourself".
		"you have to trust your partner to be
		looking for the mistakes as well. so you

(*Continued*)

(Continued).

Data Source	Themes/Codes	Quotes from data
		can cross reference the mistakes. . . so each
		of you are doing a part to help finish, help
		complete"
	Developing a deeper understanding about debugging	"it kind of taught me that one little
		bracket could change everything".
		"looking through those mistakes and
		looking how the mistakes were made, then
		we got to continue learning, learning further".
	Emotions around debugging projects not authored by themselves	"It was fun, frustrating, interesting, and it's
		great way to solve problems that you haven't"
		"different from debugging a code that you're
		writing yourself because you know what you
		want your code to do, and you know which
		part is your weak point. . . we have to focus on
		everything"
		"it's not your project, so you have this fresh
		view on it where you've never seen it before. . .
		it's harder to see mistakes on your own
		project"