Cite as:
Fields, D. A., Kafai, Y. B., Morales-Navarro, L., & Walker, J. T. (in press). Debugging by Design: A Constructionist Approach to High School Students' Crafting and Coding of Electronic Textiles as Failure Artifacts. British Journal of Educational Technology

Debugging by Design: A Constructionist Approach to High School Students' Crafting and Coding of Electronic Textiles as Failure Artifacts

Deborah A. Fields, corresponding author, Utah State University, deborah.fields@usu.edu,
2830 Old Main Hill, Logan, UT, 84322
Phone: 435-797-2694
Toll-Free: 866-782-9301
Fax: 435-797-2693

Yasmin B. Kafai, University of Pennsylvania, kafai@upenn.edu

Luis Morales-Navarro, University of Pennsylvania, luismn@upenn.edu

Justice T. Walker, University of Texas, El Paso, jtwalker@utep.edu

Deborah A. Fields is an associate research professor of Instructional Technology and Learning Sciences at Utah State University. In her research she seeks to inspire and advocate for children's creative expression with digital media, coding, and everyday craft materials. Her work has appeared in journals such as *Mind, Culture and Activity*, the *International Journal of Computer Supported Collaborative Learning*, and the *Harvard Educational Review*. She is a fellow of the International Society of Design and Development in Education.

Yasmin B. Kafai is Lori and Michael Milken President's Distinguished Professor at the University of Pennsylvania. She is a researcher and developer of tools, communities, and materials for the promotion of computational participation, crafting, and creativity across K-16. She wrote *Connected Code: Why Children Need to Learn Programming* (2014, with Quinn Burke), and co-edited *Textile Messages: Dispatches from the World of Electronic Textiles and Education* (2013, with Leah Buechley, Kylie A. Peppler, and Michael Eisenberg) and *Designing Constructionist Futures: The Art, Theory and Practice of Learning Designs* (2020, with Nathan Holbert and Matthew Berland).

Luis Morales-Navarro is a graduate student in the Learning Sciences and Technologies program at the University of Pennsylvania. His background is in middle school computer science, open-source software development, and physical computing. He is interested in issues of access, inclusion, and motivation in learning to code, and the role communities play in the development of computational fluency.

Justice T. Walker is an assistant professor of Science Education at The University of Texas at El Paso. He also has more than a decade of high school and university science teaching experience. Walker has a learning sciences research background in emerging technologies in biology and computer sciences and their use in middle and high school education. His work focuses on examining how learners construct knowledge, engage literacy practices, and use design in science learning.

Debugging by Design: A Constructionist Approach to High School Students' Crafting and Coding of Electronic Textiles as Failure Artifacts

Deborah A. Fields, Yasmin B. Kafai, Luis Morales-Navarro, and Justice T. Walker

Abstract

Much attention in constructionism has focused on designing tools and activities that support learners in designing fully finished and functional applications and artifacts to be shared with others. But helping students learn to debug their applications often takes on a surprisingly more instructionist stance by giving them checklists, teaching them strategies or providing them with test programs. The idea of designing bugs for learning—or *debugging by design*—makes learners agents of their own learning and, more importantly, of making and solving mistakes. In this paper, we report on our implementation of "Debugging by Design" activities in a high school classroom over a period of eight hours as part of an electronic textiles unit. Students were tasked to craft electronic textile artifacts with problems or bugs for their peers to solve. Drawing on observations and interviews, we answer the following research questions: (1) How did students participate in making bugs for others? (2) What did students gain from designing and solving bugs for others? In the discussion, we address opportunities and challenges that designing personally and socially meaningful failure artifacts provides for becoming objects-to-think-with and objects-to-share-with in student learning and promoting new directions in constructionism.

Keywords

Debugging, E-Textiles, Computer Science Education, Physical Computing, Productive Failure

Structured practitioner notes

What is already known about this topic:

- There is substantial evidence for the benefits of learning programming and debugging in the context of constructing personally relevant and complex artifacts, including electronic textiles.
- Related, work on productive failure has demonstrated that providing learners with strategically difficult problems (in which they "fail") equips them to better handle subsequent challenges.

What this paper adds:

- In this paper, we argue that designing bugs, or "failure artifacts" is as much a constructionist approach to learning as is designing fully functional artifacts.
- We consider how "failure artifacts" can be both objects-to-learn-with and objects-to-share-with.
- We introduce the concept of "Debugging by Design" (DbD) as a means to expand application of constructionism to the context of developing "failure artifacts."

Implications for practice and/or policy:

- We conceptualize a new way to enable and empower students in debugging—by designing creative, multimodal buggy projects for others to solve.

- The DbD approach may support students in near-transfer of debugging and the beginning of a more systematic approach to debugging in later projects and should be explored in other domains beyond e-textiles.
- New studies should explore learning, design, and teaching that empower students to design bugs in projects in mischievous and creative ways.

1. Introduction

Much attention in constructionism has focused on designing tools and activities that support learners as creators of fully finished and functional artifacts or applications—games, stories, robots or sandcastles—to be shared with others (Papert, 1991). Prior studies provide substantial evidence for the benefits of learning programming in this context: constructing personally relevant and complex applications rather than in writing short pieces of code or solving classic code puzzles (Harel & Papert, 1990; Kafai & Burke, 2014). However, less attention in constructionism has been paid to the potential of learners as creators of personalized *failure artifacts* when the constructed applications have deliberate bugs or mistakes in them that need fixing. Further, failure artifacts can also incorporate the constructionist priority of an authentic audience, if they are shared with others that engage in fixing the bugs to make the artifacts fully functional. While this approach provides a different perspective on constructionist learning, it makes equally transparent functions and structures of computational designs.

We argue that designing failure by intentionally including mistakes or bugs can be as much a constructionist approach to learning as is designing fully functional artifacts. Drawing on constructionist philosophy (e.g., Papert, 1980) we build on a longstanding tradition of putting learners in control of their own learning by designing applications for others (Harel & Papert, 1990, Kafai, 1995). We propose having learners intentionally design *buggy* (rather than functional) computational artifacts for their peers to fix. The idea of designing buggy artifacts for learning—or "Debugging by Design" (DbD)—builds on two core principles of constructionism that artifacts of learning are (1) objects-to-think-with (Papert, 1980) and (2) objects-to-share-with others (Kafai & Burke, 2014). DbD also provides students with control over bugs, a contrast to school cultures where failure can be a very negative experience rather than a productive one (e.g., Dahn & DeLiema, 2020). In the context of designing projects with mistakes, DbD brings both consideration of audience and student control over design of mistakes.

In this paper, we explore the feasibility of DbD for learning and teaching about debugging in classrooms in which students created and then exchanged and solved buggy electronic textiles projects. A physical computing activity, electronic textiles (e-textiles) involve stitching circuits with conductive thread to connect sensors and actuators to microcontrollers (Buechley, Peppler, Eisenberg & Kafai, 2013) and provide multiple opportunities for bugs across modalities of hardware and software (Resnick, Berg & Eisenberg, 2000). We implemented DbD over a period of eight hours within an introductory high school computing class with 25 consenting 9th grade students (ages 14-15) in the United States. Since this was our very first exploration of the DbD unit, we sought to understand: (1) *How did students participate in making bugs for others?* (What did they create; what unexpected directions did they take; what challenges did they face in the process?) To answer this first question we conducted close video analysis of four case studies of student teams participating in DbD, documenting their step-by-step design, creation, and solving of buggy projects. (2) *What did students perceive as gains from designing and solving bugs for others?* (How did students feel about the experience afterward; what benefits did they see?) To

answer the second question, we analyzed individual reflections written immediately after the DbD unit and focus group interviews conducted a few weeks after the unit was complete. In the discussion, we address how debugging by design provides a new perspective on artifact construction and constructionist learning.

2. Background

Debugging skills are difficult to develop. When students create complex applications, they often make errors—or bugs—of various types which hinder their program completion. These bugs can range from simple syntactic problems such as forgetting commas or making typos to more complex challenges that involve dealing with thorny run-time errors or logic design (e.g., McCauley et al., 2008). Designing activities and tools that support students in these challenges is important because debugging requires not just considerable technical skills and program understanding but also emotional intelligence and perseverance (e.g., Patil & Codner, 2007). However, helping students learn to debug often takes on a surprisingly instructionist stance by giving them checklists, teaching them strategies, or providing them with test programs or buggy programs to fix (e.g., Prather et al, 2019). Further, debugging is often done with small, isolated bugs (ibid.), which while effective in demonstrating specific techniques, miss the challenges that occur in open-ended projects, the latter a hallmark of constructionist activities.

In the design of our DbD approach, we addressed these challenges in multiple ways. First, we adopted a more positive stance towards debugging: bugs became an intentional feature of the learning product rather than an accidental stumbling block. This stance is inspired by the "productive failure" instructional approach (Kapur, 2008), whereat learners performed better on subsequent tasks after first engaging with more difficult ill-structured tasks, in part, because they had developed problem solving strategies that could be leveraged, or transferred, to solve future problems. While much of the extensive research on productive failure focuses on identifying which dimensions are most productive for which students and under what conditions (Kapur & Bielaczyc, 2012), it is often limited to failure in the context of well-structured canonical problems. Our DbD approach differs from classic productive failure by encouraging more open-ended, creative bug design. Further it focuses on bugs within whole, aesthetically-motivated projects rather than on individual, isolated bugs. This draws on the constructionist ethos of personally-driven projects and provides students with extra motivation to persist in problem solving and troubleshooting (Dahn & DeLiema, 2020; Hughes, Morrison, Mamolo, Laffier, & de Castell, 2019).

Second, we situated DbD in a nontraditional area for computing education: physical computing, namely e-textiles. In e-textiles, multiple bugs often co-occur in students' designs, presenting across on-screen and off-screen modalities (e.g., within and across circuitry, crafting, coding). This creates challenging situations for students seeking to isolate, identify, and fix problems (Searle, Litts, & Kafai, 2018). Debugging e-textile projects thus presents a particular opportunity to consider buggy projects as objects-to-think-with in their own rite, with multiple debugging challenges in a single design (e.g., Maltese, Simpson, & Anderson, 2018). These intersecting failures provide within-task feedback that supports the generation of a wide array of successful and unsuccessful solutions. The task of making a buggy project functional provides a practical means of constructing knowledge about debugging strategies.

Third, we put learners (rather than teachers or researchers) in charge of creating productive—and personally-meaningful—failure projects. Earlier studies have demonstrated the rich learning opportunities researcher-designed problem artifacts in physical computing contexts offer. For instance, Sullivan (2008) presented students with a carefully designed set of robotics

dilemmas and examined students' intricate inquiry skills. Others have developed e-textile problem sets for students to solve collaboratively as a means to assess student knowledge and skills (e.g., Fields, Searle & Kafai, 2016; Jayathirtha, et al., 2020). While these studies have revealed the utility of solving bugs as assessments and explorations of student thinking, the control of bug design was always in the hands of researchers. Building on previous constructionist approaches where students became instructional software designers (Harel & Papert, 1991), we turned students into instructional bug designers and encouraged creativity and personal expression in their designs.

As such the DebugIts—our name for the failure artifacts—became objects-to-think-with as students contemplated what kind of bugs and where to include them, as well as objects-to-share-with as students considered those who would find and solve the bugs. With students free to design their e-textiles, we expected variety in the kind of DebugIt artifacts as well as in the type, location, combination and intentions of bugs, challenging both the bug designers and debuggers. Furthermore, we were interested in exploring the emotional side of failure artifacts. What students designed, how they went about those designs, and what they felt they learned from the whole process is the focus of this exploratory study.

3. Methods

3.1 Participants

The participating class was located in a high school in a large metropolitan area in the southwestern United States. This 9th grade (primarily ages 14-15) introductory computing class included 25 consenting students (out of 26 students total): 11 girls and 14 boys aged 14-18 years old: 72% speaking languages other than English at home, 80% with no prior computer science experience, and 20% with no family members with college experience. The class was racially diverse, with 48% Latino, 36% Asian American/Pacific Islander, 8% White, 4% Other, and 4% race not reported. The teacher (Ben) had three years of experience teaching the e-textile unit and helped co-develop DbD. Students were assigned to 12 groups (2 groups of three and 10 groups of two students). From the class, four collaborative groups were selected by the teacher for further study in order to represent a range of student interaction and performance: two groups of two and two groups of three.

3.2 Debugging by Design Context

The DbD unit was situated about three-fourths of the way through the e-textiles unit of *Exploring Computer Science* (ECS), a year-long, equity-focused and inquiry-based course providing an introduction to computing (Goode, Chapman, & Margolis, 2012; http://exploringcs.org/e-textiles). The e-textiles unit lasted 12 weeks and consisted of a series of four projects that allow increasing flexibility in design and personalization while learning challenging new technical skills: 1) a paper-card using a simple circuit, 2) a wristband with a parallel circuit, 3) a classroom-wide collaborative mural project that incorporated switches to computationally create light patterns, and 4) a project that used handmade sensors to create lighting effects.

The DbD unit took place over eight, 50-minute long class periods between projects 3 and 4 (see Table 1 for the DbD timeline). The student-designed DebugIts had to contain at least six bugs, including two coding bugs, with one undetectable by the Arduino compiler. This latter constraint helped students move beyond simple syntax problems in their designs. DebugIts also had to involve either a switch or a sensor to ensure a level of coding challenge with conditionals and functions. Finally, students had to include a description of how the project should function when fixed. This allowed for the inclusion of design errors (or "intention errors" as the class named

them) where a project might function but not as desired. The final DebugIt design included: a list of problems and solutions, a circuit design showing any circuitry errors, code, and a statement of how the DebugIt should work.

Table 1. Debugging by Design Unit.

Class 1	**"Hall of Problems":** As partners then as a whole class, students list e-textile problems. Then they categorized these problems into groups, which are written on posters on the classroom walls.
Class 2	**DebugIt Design:** Students plan their DebugIts, turning in a list of problems with solutions as well as a circuit diagram showing any circuitry bugs. Designs had to be approved by the teacher. Most groups revised their designs after teacher feedback, which continued into Class 3.
Classes 3-5	**DebugIt Construction**: After receiving teacher approval on their design, students created their DebugIts, sewing, and coding their projects.
Classes 6-7	**DebugIt Solving:** Directed by the teacher, students exchanged projects and had 1.5 class periods. Students then reflected on what the best, most frustrating, and surprising parts of the entire debugging by design experience were.
Class 8	**Reflection on Problem-Solving Strategies:** Individually then as pairs and as a class, students reflected on the kinds of strategies they used to solve DebugIts.

The design of the DbD unit has several characteristics: First, it was situated in the latter half of the larger e-textile unit, allowing students to build on earlier bug experiences in designing their DebugIts and to apply their DbD experiences on their final projects. Second, the unit began with group discussions where students named problems that had come up in their own previous designs and categorized these problems. This promoted class-wide transparency of problems across students' prior projects. Third, students received teacher approval on their DebugIt designs before they could construct them. The approval process enabled the teacher to challenge students to either make problems more interesting and creative or consider whether the problems they created were potentially solvable within a single class period. In other words, students received feedback on both the difficulty level and number of problems they introduced. Fourth, after students exchanged and solved each other's problems, they presented their solutions to the class, letting the designers see to what degree and how their peers had solved the designed problems. Finally, the class participated in reflective journaling and discussion about how they felt designing and solving DebugIts and the kinds of strategies they employed in solving problems.

3.3 Data Collection and Analysis

Data for our analyses was drawn from daily observations (fieldnotes and videos, including cameras on each of four case study groups), recorded teacher reflections during the DbD unit, pictures of student projects, code files, student reflections (n=24 students) written after the unit, and post-interviews with students in focus groups (n=21 students). Case study groups included a total of 10 students (two groups of three and two groups of two students each).

Analysis was completed in two parts. First, to obtain a closer look at student participation during the DbD unit, we analysed the moment-by-moment designing and debugging processes of the case study groups (Yin, 2017). We assembled all available data about each group from daily videos and fieldnotes, end-of-unit participant interviews, daily teacher reflections, and daily documentation of students' designs (pictures of physical products at different stages of creation

and students' code). This provided rich, detailed data for creating design narratives that included multiple perspectives on each group's DebugIt design, including contextual details that influenced design decisions, such as peer interactions (within and between groups), teacher support, and whole class instruction. We further analyzed design narratives to uncover: (1) how students developed a bug from idea to implementation, (2) what kinds of conversation/engagement occurred surrounding bug design, and (3) how students responded to others' bug design (i.e., debugging exchanges). We followed the creation of each designed bug from start to finish (including bugs that were dropped for a variety of reasons and bugs that students accidentally designed). Importantly, this analysis allowed us to trace the intention behind different bugs and student groups' holistic approaches in considering the audience (or end-user) of their designs.

Second, to better understand the breadth of student experiences in the class, we analyzed student reflections: more immediate written reflections and retrospective interview reflections. This part of analysis involved two-step, open coding of reflections and transcribed interviews (Charmaz, 2014). We began by identifying overarching themes from reflections and interviews, then followed by creating sub-categories within codes and comparing across codes to develop a richer coding scheme. This was done iteratively across several meetings amongst the research team (four people) until we reached interpretive agreement. Finally we applied the revised coding scheme across all the data and looked for frequency across students (i.e., how many students spoke to a particular theme) to identify how prevalent or rare trends were.

4. Findings

The 12 student teams successfully designed personalized DebugIts with varying dysfunctionality intended to befuddle their peers. Projects ranged from flat pieces to plushies (see Figure 1) with nearly 200 bugs that varied in number and complexity. These bugs ranged from simpler problems in short circuits (long threads stretching between active circuit lines) and missing semicolons in code to more complex problems in conditional logic and mismatches between code and crafted circuits. Several groups used the projects to advance into new domains of e-textiles, not previously covered in the class, pursuing interests such as fading lights, playing music, or using sensors. Figure 1 presents a sampling of students' DebugIts to convey the range of personally meaningful

projects with aesthetics that represented students' interests and were carefully designed for their peer audience.

Figure 1. Examples of student teams' DebugIts (clockwise from upper left): Smiley face, peace sign, Pokemon ball, two suns, Mario star, and "Starry Night", which illustrate different aesthetic choices and configurations of circuit designs, LEDs and functionalities in code.

To illustrate the complex and nuanced ways that student teams participated in the Debugging by Design unit, we present two case studies (limited because of space), chosen because they provide clear illustrations of a commonality across the e-textile DebugIts in the class, namely using the aesthetics and distributed modalities of e-textiles to challenge their peers. Throughout the descriptions we highlight the complex histories and intent behind specific bugs and the combination of bugs as whole, including the role that consideration of audience played in design

4.1. Case 1: Evelyn and Nicolás

For their DebugIt, Evelyn and Nicolás created a "sick cloud throwing up a colorful rainbow" (see Figure 2, upper left). In the description they gave to their peers, they explained that the project was supposed to "blink all together at half a second when button 1 (pin19) is pressed with the colors of the rainbow," otherwise the lights should all be on. Overall, within this seemingly simple project, the pair included several bugs in the code and a few in the physical craft and circuitry. Some bugs were simple coding bugs, e.g., missing semicolons and misspellings of variable names. Other bugs included circuit bugs such as one light with reversed polarity (negative and positive mixed up) and crafting bugs such as loose thread creating a short circuit and one gap in a circuit connection. These are typical bugs many novices make in crafting, designing circuits and writing code for their e-textiles (Fields et al., 2016).

A critical feature of many DebugIts was a type of design bug which the class emergently named "intention errors": where the project is functional but does not work according to the intention of the designers. For instance, in Evelyn and Nicolás' DebugIt, a comparison of the sick cloud intention with its original buggy code and project design reveals three intention errors. First,

pin19 is the button the group intends to trigger their project, but the actual button that was coded was different (pin4). Second, the description clearly explained that all the lights should blink at once, even though the code had five variables for lights: rainbow, yellow, green, blue, pink. Third, the primary conditional statement in the code utilizes two button values (butt1Val and butt2Val) when, according to the project description, only one is needed.

The decision-making behind selecting and creating bugs was thoughtful and complex. Early on in design, Evelyn and Nicolás brainstormed ways to create a bug that would involve nuisance crafting, with the idea that stitches are "easy to fix" (Video Day 2). However, during the ensuing design time, the pair began to discuss how long it would take for others to solve their project. For this pair, a major challenge in each crafting bug was making sure that it would not require too much restitching so that the receiving team could fix it in the time available (Video Day 2). In the end, the "sharing" part of objects-to-share-with played a key part in the pair's decision-making to keep crafting bugs very limited: one light sewed on backwards (reverse polarity), one short circuit, and one area of unknotted thread (i.e., an open circuit).

This consideration of audience came up many times, even when the pair problem-solved their own accidental bugs. For instance, while sewing the negative side of the lights on the rainbow, Evelyn neglected to pull thread all the way through in one of the stitches, creating a giant knot with several inches of extra thread at the back of the project. In admitting to her partner that, "I messed up," Evelyn suggested that they turn her accidental mistake into a bug for their audience (Video Day 4). However, after examining the issue and how others might resolve it, Nicolás disagreed. The pair debated together how difficult it would be to solve this bug—how much time it might take—and then decided to fix it by pulling the extra thread through and weaving it behind the negative thread going through the other lights (Video Day 4). This fix actually turned out to be a confusing element for the team that solved their project (see below), but it was also a clever, time-saving choice in a situation where design time was highly limited,

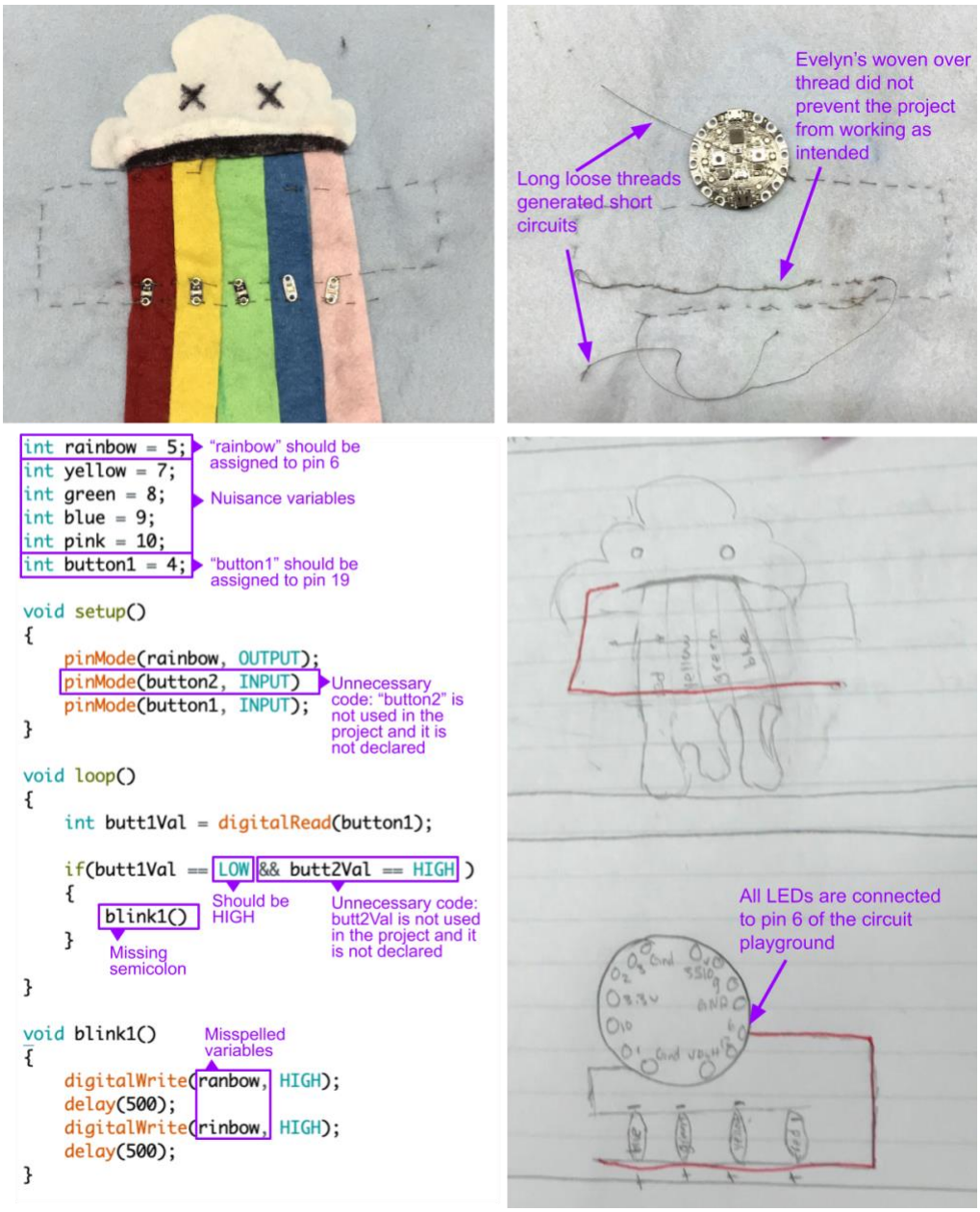

Figure 2. Evelyn and Nicolás' DebugIt (clockwise from upper left): (1) Front of e-textile: "sick cloud throwing up a colorful rainbow"; (2) Back of e-textile: the two loose threads are causing a short circuit; (3) Circuit diagram: In the diagram all LEDs are connected to pin 6 of the microcontroller; and (4) Code (near top): declaring nuisance variables in their code solely to confuse their audience as these variables did not prevent the program from running as expected.

Evelyn and Nicolás' consideration of their peers played out both in generous and mischievous ways. The more considerate attention described above aligned with the teacher's coaching to think through whether their peers would be able to reasonably solve all the designed bugs within one class period, though we should note that not all students were as mindful as Evelyn and Nicolás in limiting errors. The more roguish attitude of the pair came out in their attempts to confuse the peers who received their "sick cloud" project. In fact, the design of one parallel circuit for the rainbow lights with five individual variables for said lights in the code was purposefully to "throw people off" as was the decision to leave in the two-condition logic with two buttons, when

only one was needed (FN Days 5 & 7). Here we see the situated and social dimensions of personally significant objects-to-share-with: both students expressly voiced their desire to confound their peers through these extraneous pieces of code—and succeeded in ways intended and unintended.

As intended, the receiving team—Emma, Lucas, and Lily—spent nearly 10 valuable minutes debating whether the five LEDs in the rainbow needed to be resewn because they were in parallel. After cutting the loose threads (i.e., short circuits) and puzzling over this issue they decided to ask their teacher for advice. The teacher pointed out that bugs depend on the intention of the project, that is, what Evelyn and Nicolás intended when they designed them (FN Day 6). After reading the intention statement of the project, Lucas expressed "that's what they want", and the issue was resolved with little effort. In contrast, the accidental knot with the extra thread woven into the back of the project, unintentionally befuddled the receiving team. They debated whether to cut the threads and resew the connections "because it looks wrong" (FN Day 6) before Emma summarily cut the connection to ground and spent considerable time resewing the entire long connection. Ironically, the bugs intended to confuse the audience successfully kept Lucas, Emma, and Lily occupied for many minutes, but it was the deliberate attempt to make their peers' problem solving easier that took most of their time. Through both designing and solving, the "sick cloud" became an object-to-think-with that required examining and revising connections between old and new knowledge, it introduced each group to new problems across modes of crafts, circuit and code and challenged each to new solutions.

4.2 Case 2: Lucas, Emma, and Lily

For their DebugIt, Lucas, Emma, and Lily created a heart with two sheets of felt, one over the other. On one side of the heart, they attached the microcontroller and, on the other, four LEDs (see Figure 3). Similar to Evelyn and Nicolás, their project contained a number of simple code bugs: missing semicolons, mismatched variable names, mismatched pin number assignments (i.e., circuit pin numbers did not match the programmed pins). They also designed a number of common circuitry and crafting bugs: a light with reversed polarity, loose knots, a short circuit, and an open circuit (with a ripped thread). One coding bug was a little unusual: the trio wrote a comment in the code that did not match the conditional statement: "button 1 on and button 2 on" when it should have been "button 1 on and button 2 off". However, the group was particularly devious in the spatial placement of their crafting/circuitry bugs: the bugs were *inside* the heart (i.e., between the two layers of felt) and one LED was placed upside down, with the light shining in toward the fabric.

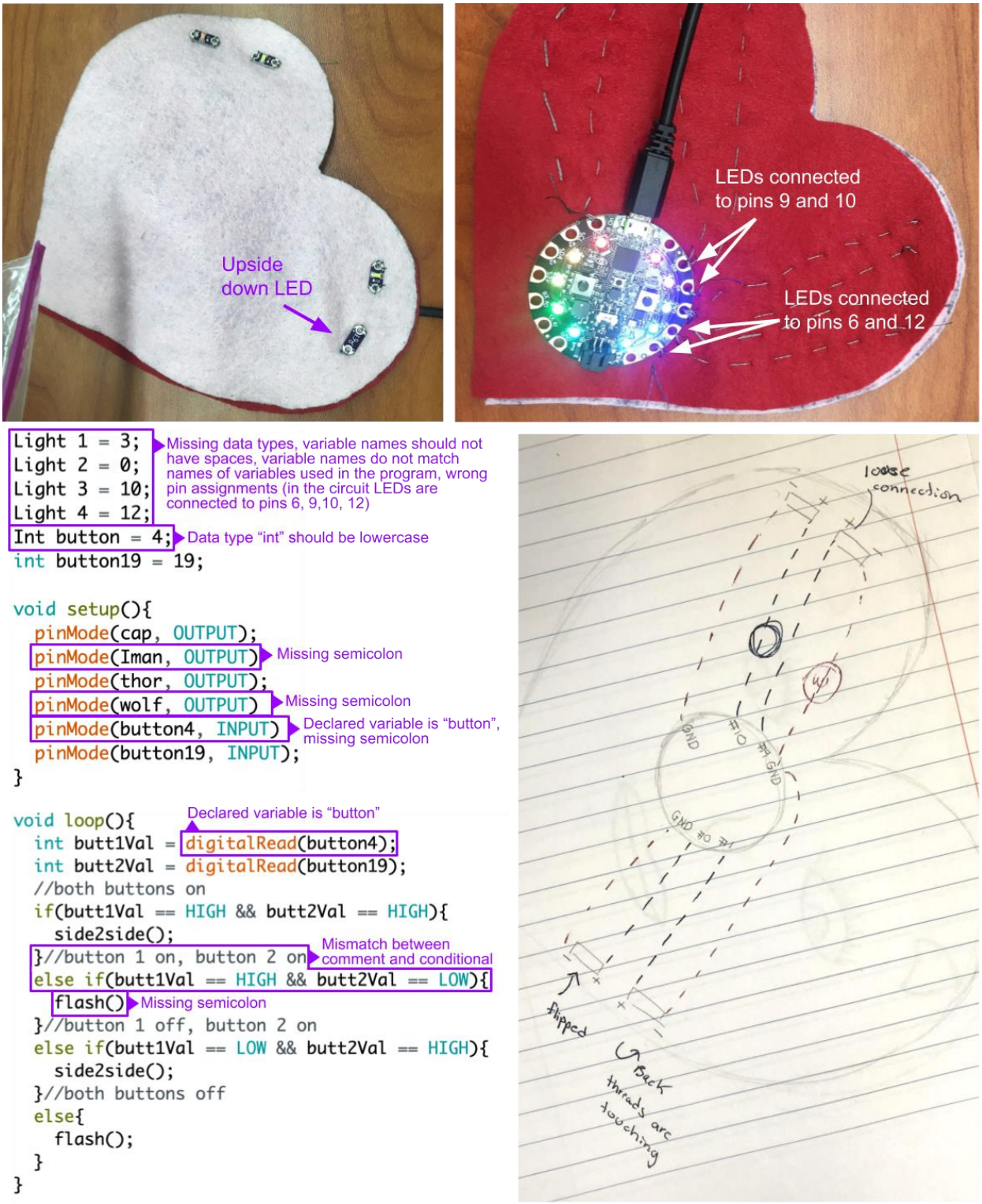

Figure 3. Lucas, Emma and Lily's DebugIt (clockwise from upper left): (1) Front of e-textile: a white heart with four LEDs, one LED is flipped with the light directed towards the felt; (2) Back of e-textile: the microcontroller is sewn over a second layer of felt, threads are loose and LEDs are attached to pins 9, 12, 6 and 10; (3) Circuit diagram: student annotations show the crafting bugs, some of these bugs were created between the two sheets of felt; and (4) Code: including a mismatch between the first conditional statement and the comment next to it.

In contrast to Evelyn and Nicolás, much of the trio's Debugit design was set from their earliest discussions. Lily and Emma (Lucas was absent) identified bugs, wrote them in a list, and started planning where they would go. Although creating a diagram of the project was not required, Lily convinced Emma that the best way to plan the project was by drawing out an aesthetic and circuit diagram (see Figure 3), and created new ways to annotate items such as torn threads, crossed circuit lines, and reversed polarity. As an object-to-think with, this diagram helped guide their discussions and kept them on track in their design. One major concern of the group was time—how much time would different parts take to complete? For this reason they kept the main canvas of the DebugIt simple: the form of a heart. They reused code from a prior project (e.g., the superhero variable names like (e.g., Thor) came from Lucas' prior project) and added mistakes. Besides concern about their own time management, the team also considered the other team's time to solve the bugs (Video Day 4), worrying that the number of crafting bugs would be "too much" (Emma, Video Day 7).

This concern about audience also stretched to playfully imagining how to confuse their peers and how they would react, further emphasizing how the DebugIt was also an object-to-share-with. This is one reason behind the decision to flip an LED so that the light was directed inwards (Video Day 2). Indeed, Lucas only half-jokingly suggested flipping *all* the LEDs and even the microcontroller upside down (Video Day 4)! The group considered several other sly bugs that mercifully did not make it into the final DebugIt, including adding more pin errors to confuse their audience (Video Day 3) and glueing the project edges together. Notably, in their design trajectory the group shifted from hesitation about designing bugs (on the first day Lucas wrote in his journal that "creating bugs feels very wrong") to gleeful delight in their mischievous design.

When Evelyn and Nicolás, the receiving team, worked on debugging the heart they found the spatial bugs the most problematic. They easily breezed through the coding errors—they were already adept at designing similar bugs themselves. The misleading comment in the code did not even draw their notice since nothing was actually dysfunctional in the conditional logic. They also quickly resolved the backwards LED by simply flipping the light (twisting it in place)—thereby avoiding having to resew it and saving a "good twenty minutes" (Video Day 7). However, the circuitry errors inside the two pieces of felt were entirely original to Nicolás and Evelyn; they had never encountered "double felt" before and it genuinely confused them (FN Day 8). Getting the code running helped the pair isolate the probable circuitry errors inside the project, since the outside looked fine. Another clue was that the heart was discernibly hot from a short circuit—Nicolás reported feeling "burned" (no injuries occurred in making or solving this project). Applying prior expertise to develop new abilities in problem solving, Evelyn and Nicolás eventually solved the DebugIt and presented a working project to the class.

4.3. Student Reflections

While the case studies illustrated the processes in which two teams participated in making and fixing Debugits, there was a marked difference in students' feelings about the DbD process immediately and several weeks after their final projects were complete. In written reflections after solving their peers' DebugIts, most students in the class explicitly expressed frustration. They noted that the errors were difficult to detect, often involved a lot of cutting and re-sewing to fix, and left them without enough time to solve everything. As Avery expressed, "It felt weird debugging someone else's project because they INTENTIONALLY (sic) put more bugs than a normal project ...I felt uncomfortable and MAD". Some students also found it interesting or even fun to debug, but the most-repeated word (by 18 students) was "frustrating".

However these feelings were drastically different several weeks later after students completed their final e-textile project in the unit. Indeed, all students interviewed expressed feelings of increased *comfort and competence* with solving and designing problems. They said that the DbD unit should be done again next year because it was such a good learning experience, and several even asked for it earlier and more often! As for what they appreciated about the unit, many students remembered creating problems "challenging enough to stress someone out" as "funny and good" (Nicolás), claiming that this gave them a "new perspective on coding" (Liam) that was the "opposite" of what they normally experienced, making debugging both challenging and interesting. As Evelyn said, "it helped me realize I knew if I saw the errors in the next one, I knew how to fix it." Being more comfortable with problems also helped students feel better able to ask for help from others since they became more aware that "a lot of people make mistakes" (Camila). Students located this sense of power over problems as a direct consequence of DbD.

Not only did students feel more comfortable with problems, they also reported learning several important aspects about problems and problem-solving from DbD that emphasize the potential of DebugIts as objects-to-learn-with. All students interviewed claimed to have applied their DbD experience to their final projects. They explained this in several ways. Many students described becoming more familiar with a range of problems: "the type of errors there are, like how errors can be prevented and caused and everything" (Nicolás). Knowing more problems also enabled students to avoid problems in creating their final project. Gabriel explained that designing your own bugs "makes you more aware that the next time you're creating a project… [you] make sure you don't make that mistake in the project." Of course, avoiding some problems in their final projects did not mean that no errors occurred. All students described problems in their final projects. However, students claimed they were able to identify problems more easily: interpreting compiler feedback on syntax errors, applying a process for detecting errors across physical and digital systems, or isolating and testing problems. A few students even described the beginnings of a systematic approach to debugging that is rare amongst novice coders (McCauley, et al., 2008), naming a series of steps in an order that facilitated isolating problems and comparing code and craft.

5. Discussion

In this paper we conceptualize a provocative means to enable and empower students in debugging in a constructionist fashion—by designing personally meaningful, *buggy* projects for others to solve. Case studies of student teams' design actions, intentions, and reflections demonstrated high levels of interest, thoughtfulness, and creativity in design, and illustrated how students were "becoming more articulate about one's debugging strategies and more deliberate about improving them" (Papert, 1980, p. 23).

Reflections from across participating students further showed distinct perceived benefits from DbD, suggesting that constructing buggy projects can be as much objects-to-think-with as creating other personally meaningful computational objects. As with functional constructionist artifacts, failure artifacts serve as concrete representations of learners' knowledge and understanding. Easy bugs are easy to design (e.g., deleting a semicolon) while difficult bugs are also difficult to design (e.g., logic problems or cross-modal contradictions). The combination of bugs must be thoughtful so as to confound but not overly discourage recipients. It is in this spirit that we see the idea of designing buggy projects as a rich way to expand productive constructionist learning in computer science education.

In addition, we observed that buggy projects also acted as objects-to-share-with. This is evident in the important roles of peer collaboration and consideration of audience in design. Audience was particularly important as students considered the number, difficulty, and combination of bugs that must be solved within a short period of time. Considering audience helped students think of the buggy projects more holistically, as a collection of bugs that interacted with each other. Further, the entire DbD experience resulted in reports of enhanced collaborative problem solving, including acute awareness that everyone in the class made mistakes, was familiar with mistakes, and could either offer or might need support with mistakes. This opens up new possibilities for considering the role of audience in learning debugging, a contemplation found little in reviews on debugging in computer science education (e.g., McCauley et al, 2008; Prather et al, 2019). When debugging is framed as the object of design, audience becomes important and may support students in evaluating the relative difficulty and combinations of bugs.

One unexpected theme running across our findings was the expression of emotion in designing and solving failure artifacts. Mischievousness and fun as well as empathy and sensitivity (for peers receiving DebugIts) were productive emotions exhibited during bug design, and shifts away from frustration to increased comfort, security, and a sense of control with bugs were expressed retrospectively weeks afterward. Given the problematic connotation of failure in school cultures that rarely value it, this suggests a productive angle of future research on *the role of emotion* in Debugging by Design or in dealing with bugs more generally in constructionist settings (see Dahn & DeLiema, 2020). What is the role of emotion in tackling and/or designing bugs in learner-driven designs? What scaffolds, classroom support, and curricular designs support perseverance, a sense of capability, or resoluteness in debugging? More attention is needed on the roles and ranges of emotion and motivation in designing for and experiencing failure while making personally meaningful, socially shared objects.

In future research, we will examine other aspects that contribute to empowering students to design bugs in projects. For one, we plan to expand implementation of DbD in the e-textiles unit to multiple teachers. This will enable us to examine variances in and best teacher practices of implementing DbD. The case study analyses already hint at the importance of the teacher in providing constructive criticism at key points of design and creating a space where students can freely share their mistakes and problems. Comparative analyses may make these and other practices more clear. Further, it is difficult in the analysis of just one classroom to identify what role DbD had in students' increased comfort with and sensemaking of debugging. A quasi experimental design with some classrooms doing just the e-textiles unit and others doing the e-textiles unit with DbD is planned to help illuminate the particular role that DbD may play in students' growth.

6. Conclusions

This paper presents a provocative angle on foregrounding debugging in constructionist activities. It draws attention to bugs not as accidents to be solved but as objects-to-think-with and objects-to-share-with. This approach of making students designers of bugs, or mistakes, is not limited to CS education and could also be applied to other constructionist contexts in which students design artifacts. We look forward to insights from creative applications of Debugging by Design or similar interventions to other domains in physical computing, general software design, and many other areas of design more broadly.

Acknowledgments

This work was supported by a grant from the National Science Foundation to Yasmin Kafai (#1742140). Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF, the University of Pennsylvania, or Utah State University. Special thanks to Lindsay Lindberg for support in data collection and Gayithri Jayathirtha for feedback on this paper.

Statements on open data, ethics and conflict of interest

Data for this project are unavailable due to data protection and privacy concerns.
Ethics approval for this research was obtained from the University of Pennsylvania Institutional Review Board (IRB) and procedures were in accordance with U.S. policies for protection of research subjects. All names in this paper are pseudonyms to protect privacy.
The authors do not wish to declare any conflicts of interest.

References

Buechley, L., Peppler, K., Eisenberg, M., & Yasmin, K. (Eds.) (2013). *Textile Messages.* New York, NY: Peter Lang.

Charmaz, K. (2014). *Constructing grounded theory*. Sage publications, Thousand Oaks, CA.

Dahn, M., & DeLiema, D. (2020). Dynamics of emotion, problem solving, and identity: Portraits of three girl coders. *Computer Science Education, 30*(3), 362-389.

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

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond Curriculum: The Exploring Computer Science program. *ACM Inroads, 3*(2), 47-53.

Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive learning environments*, *1*(1), 1-32.

Hughes, J., Morrison, L., Mamolo, A., Laffier, J., & de Castell, S. (2019). Addressing bullying through critical making. *British Journal of Educational Technology*, 50(1), 309-325.

Jayathirtha, G., Fields, D. A., & Kafai. Y. B. (2020). Pair debugging of electronic textiles projects: Analyzing think-aloud protocols for high school students' strategies and practices while problem solving. In M. Gresalfi, M. & I. S. Horn (Eds.). *The Interdisciplinarity of the Learning Sciences, 14th International Conference of the Learning Sciences (ICLS) 2020*, Volume 2, Nashville, TN: International Society of the Learning Sciences, pp. 1047-1054.

Kafai, Y. B. & Burke, Q. (2014). *Connected Code.* Cambridge, MA: MIT.

Kapur, M. (2008). Productive Failure. *Cognition and Instruction*, *26*,(3), 379-424.

Kapur, M., & Bielaczyc, K. (2012). Designing for productive failure. *Journal of the Learning Science*s, 21(1), 45-83.

Maltese, A. V., Simpson, A., & Anderson, A. (2018). Failing to learn: The impact of failures during making activities. *Thinking Skills and Creativity*, 30, 116-124.

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.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.

Papert, S. (1991). Situating constructionism. In. I. Harel & S. Papert (Eds.). *Constructionism* (pp. 1-13). Norwood, NJ: Ablex.

Patil, A., & Codner, G. (2007). Accreditation of engineering education: review, observations and proposal for global accreditation. *European Journal of Engineering Education, 32*(6), 639-651.

Prather, J., Pettit, R., Becker, B. A., Denny, P., Loksa, D., Peters, A., Albrecht, Z. & Masci, K. (2019, February). First things first: Providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 531-537). ACM.

Resnick, M., Berg, R., & Eisenberg, M. (2000). Beyond black boxes: Bringing transparency and aesthetics back to scientific investigation. *The Journal of the Learning Sciences, 9*(1), 7-30.

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

Sullivan, F. R. (2008). Robotics and science literacy: Thinking skills, science process skills and systems understanding. *Journal of Research in Science Teaching*, 45(*3*), 373-394.

Yin, R. K. (2017). Case study research and applications: Design and methods. Sage publications.