Debugging by Design: Students' Reflections on Designing Buggy E-Textile Projects

Deborah A. Fields, *deborah.fields@usu.edu*
College of Education and Human Services, Utah State University, Logan, UT, USA

Yasmin B. Kafai, *kafai@upenn.edu*
Graduate School of Education, University of Pennsylvania, Philadelphia, PA, USA

Abstract (style: Abstract title)

Much attention has focused on designing tools and activities that support learners in designing fully finished and functional applications such as games, robots, or e-textiles 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 again agents of their own learning and, more importantly, of making and solving mistakes. In this paper, we report on our first implementation of "debugging by design" activities in a classroom of 25 high school students over a period of eight hours as part of a longer e-textiles unit. Here students were asked to craft buggy circuits and code for their peers to solve. In this paper we introduce the design of the debugging by design unit and, drawing on observations and interviews with students and the teacher, address the following research questions: (1) What did students gain from designing and solving bugs for others? (2) How did this experience shape students' completion of the e-textiles unit? In the discussion, we address how debugging by design contributes to students' learning of debugging skills.

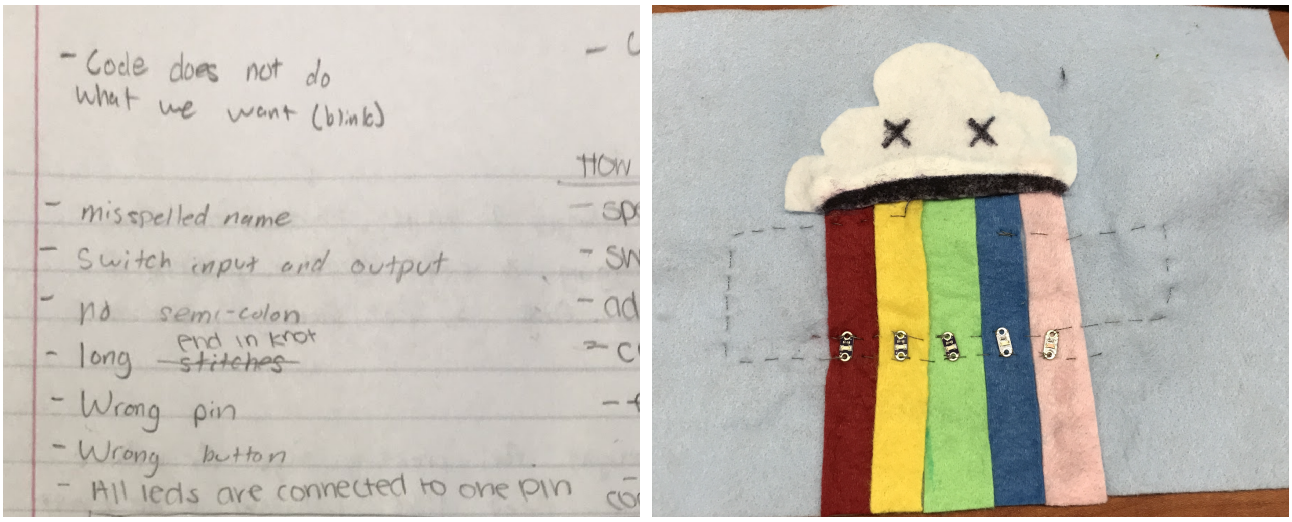

Figure 1. Example of Students' Buggy Project Design: Bugs (left) and E-textile (right)

Keywords

Programming, Debugging, E-Textiles, Computer Science Education, Physical Computing

Introduction

Much attention in Constructionism has focused on designing tools and activities that support learners as creators of fully finished applications—games, stories, robots or sandcastles—to be shared with others (Papert, 1980). Prior studies have provided substantial evidence for the benefits of learning programming in the context of designing personally relevant and complex applications rather than in writing short pieces of code (Kafai & Burke, 2014). When students design such complex applications, they often make mistakes—or bugs—of various types which hinder program completion. These bugs can range from simple syntactic problems such as forgetting commas or making typos (problems overcome by visual programming languages like Scratch) to more complex challenges that involve dealing with thorny run-time errors or logic design (McCauley et al., 2008). Helping students debug their programs often takes on a surprisingly instructionist stance: explicitly telling learners how to read and understand compiler messages or creating projects for them with specific bugs to identify and solve.

While these approaches to teaching debugging can provide valuable learning experiences, they ignore promising opportunities afforded by putting learners themselves in charge of designing buggy artifacts for their peers to solve. This approach builds on earlier successful work where students were asked to become designers of software (or games) to teach younger students in their school about academic content, such as mathematics or science (Harel, 1990; Kafai, 1995). Here we propose a paradigmatic twist to this idea by having students intentionally design *buggy* (rather than functional) software, games, robots, electronic textiles (e-textiles, see below), or other computational artifacts to teach them about bugs and the debugging process. The idea of designing bugs for learning—or "debugging by design" (DbD)—builds on the core tenet of constructionism that learners are agents of their own learning and construct mental models when they create things; in this case mistakes. Debugging by design provides a promising pedagogical application of constructionism.

In this paper, we examine the feasibility of DbD for learning and teaching about debugging. We present findings from a classroom study in which students created and then solved buggy projects for each other. The study took place in a 12-week long e-textiles unit in a 9th grade high school introductory computing class in the United States. E-textiles involve stitching circuits with conductive thread to connect sensors and actuators to microcontrollers (Buechley et al., 2013). Making an e-textile involves not only designing functional circuits but also writing code that controls interactions—thus providing myriad opportunities for bugs in crafting the physical artifact and designing the circuits and programming (Litts Lui, Widman, Walker, & Kafai, 2017). We implemented DbD in a classroom of 25 high school students over a period of 8 hours about three-quarters of the way through the larger e-textiles unit. In this paper we introduce the design of the debugging by design unit and, drawing on observations and interviews with students and the teacher, address the following research questions: (1) What did students gain from this experience? (2) How did participation in DbD shape students' completion in the e-textiles unit? In the discussion, we address opportunities and challenges that debugging by design provides in student learning.

Background

In general, teaching debugging strategies has focused on succinct screen-based problems with one bug in a piece of code. Yet bugs that arise in open-ended design situations are often much more complex, with multiple problematic bugs overlapping and difficult to identify. While there is significant research around tools and programming environments designed to support learners through the process of debugging (e.g., Sorva, Lönnberg, & Malmi, 2013), there is little research about how to support learners dealing with multiple debugging challenges in their designs. Understanding and designing learning environments that support debugging is especially important since the complexity associated with debugging demands not just programming skills but also other skills such as decision-making, emotional intelligence, and perseverance (e.g., Patil & Codner, 2007).

In addition, in physical computing contexts of e-textiles or robotics, bugs may occur both on-screen and off-screen and even in ways that stretch across both spaces, creating even more challenging situations for students seeking to isolate, identify, and fix problems. Students must not only attend to a variety of code-based errors, but also to construction errors such as incorrect placement of a sensor or incorrect circuitry, and even crafting issues in an e-textile project (Fields, Jayathirtha, & Kafai, 2019). Thus, identifying, debugging, and solving these problems is at the crux of being able to design functional computational and engineering projects. Researchers have taken these insights to design problem sets for physical computing to teach students about programming concepts and debugging. For instance, Sullivan (2008) presented students with a carefully designed set of robotics dilemmas and examined how the intricate inquiry skills students demonstrated. Fields, Searle and Kafai (2016) developed identical e-textile problem sets that students collaboratively solved. More recently, Jayathirtha, Fields and Kafai (2019) carefully studied students' debugging strategies and practices in navigating the multi-representational problem space of carefully designed buggy e-textile projects, particularly the system-level strategies students employed as they worked across spatial, material, and programmed issues. These studies provided early insights that researcher-designed problem sets in physical computing contexts are useful as meaningful assessment tools and for teaching students about debugging problems.

In this study, we approached teaching of debugging skills from a different perspective: rather than using researcher-designed problem sets, we engaged students themselves as designers of problem sets, or buggy e-textiles, for their peers to solve. A critical impetus for this approach to teaching debugging came directly from constructionist theory itself, which emphasizes the need to provide more agency to students as learners. Furthermore, we saw complex debugging situations as opportunities, or contexts for "productive failure." The concept of productive failure, introduced by Kapur (2008, 2012, 2015), highlights the counterintuitive notion that failure precedes later success in learning. Kapur's idea for designing effective learning activities was to provide student groups with carefully designed tasks in which early failures help them later in completing the problems more successfully. Today's extensive research on productive failure focuses on better understanding the role of multiple representations and solutions, their role in activating prior knowledge, and the nature of peer support during the problem generation phase to identify which dimensions are most productive for which students and under what conditions (Kapur & Rummel, 2012; Kapur & Bielaczyc, 2012). Most of these studies, however, have focused on getting students to solve well-defined canonical problems rather than on the role that failure plays in solving open-ended design problems more common in software and engineering applications such as e-textiles (Searle, Litts, & Kafai, 2018). Moreover, researchers or teachers designed these problems that provided students with experiences of productive failure. In our constructionist approach to productive failure, the learners themselves become the designers of challenging problem sets. Our research focused on understanding how such design experiences of productive failure can be realistically implemented by students within a classroom setting, how the teacher facilitated the design, and what students thought about their experiences.

Methods

Participants

The participating school was located in a large metropolitan school district in the southwestern United States. This particular class included 11 girls and 14 boys from 14-18 years old: 72% speaking a language other than English at home, 80% with no prior computer science experiences prior to the course, 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 had three prior years of teaching the e-textile unit and helped co-develop the DbD unit.

Data Collection and Analysis

Data for our analyses was drawn from daily fieldnotes and teacher reflections during the DbD unit, student reflections (n=24 students) written after the unit, and post-interviews with pairs of students (n=21 students) and the teacher, Ben, after the entire e-textile unit was completed,. Analysis primarily consisted of two-step, open coding of transcribed interviews and reflections (Charmaz, 2014), with several meetings amongst the research team (four people) to interpret, revise, and re-apply the coding scheme consistently. We compared student interview and reflection analysis with the teacher's daily reflections and field note observations to provide a fuller picture of the experience and expand on design recommendations.

Design & Context

Broader Educational Context: E-Textiles Unit for ECS

The DbD unit took eight hours, about three-fourths of the way through the e-textiles unit designed for *Exploring Computer Science* (ECS), a year-long course providing an introduction to computing with equity-focused and inquiry-based teaching (Goode, Chapman, & Margolis, 2012; http://exploringcs.org/e-textiles). The e-textiles unit took place over 12 weeks and consisted of a series of four projects that allow increasing flexibility in design and personalization in the context of learning challenging new technical skills: 1) a paper-card using a simple circuit, 2) a wristband with three LEDs in parallel, 3) a classroom-wide mural project completed in pairs that incorporated two switches to computationally create four lighting patterns (the only collaborative project), and 4) a project that used handmade sensors to create conditions for lighting effects (see Kafai & Fields, 2018).

Debugging by Design Unit

The DbD unit contained several intentional characteristics. First, it took place in the latter half of the larger e-textile unit, allowing students to build on their experiences of errors in designing their buggy projects (or DebugIts) and to apply their experiences with DbD in their final projects. Second, the unit began with partner and group discussions where students named all of the problems that had come up in their designs by a given point and then categorized these problems. This promoted class-wide transparency of problems arising across students' prior projects. Third, students had to receive teacher approval on their DebugIt designs before they could construct them. This requirement was added in the moment during the unit, after the teacher noticed that many groups were proposing identical types of problems. 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 needed feedback on both challenging and containing the 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 problems they designed. Finally, the class engaged in a reflective journaling and discussion about how they felt about designing and solving DebugIts, and what kinds of strategies they employed in solving problems. Table 1 shows the timeline of debugging by design.

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.

DebugIts had to contain at least six problems: at least two problems had to involve code, with one being undetectable by the Arduino compiler. This latter constraint was added after earlier pilots of DbD where we noticed that students tended to focus on simple syntax problems in their DebugIt designs. The requirement to create a bug undetectable by the compiler generally led to more challenging coding problems. The DebugIt had to involve either a switch or a sensor to ensure a level of coding challenge similar to the interactive mural project (i.e., with conditionals and functions). Finally students had to include a description of how the project should act when complete. 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 diagram showing any circuitry errors, code, and a statement of how the DebugIt should work.

Findings

Students' Reflections as Buggy Designers

There was a dramatic difference in students' feelings about the DbD process immediately after the experience 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 generally did not have enough time to solve everything. As Avery clearly expressed, "It felt weird debugging someone else's project because they INTENTIONALLY (sic) put more bugs than a normal project would have so fixing obvious mistakes were easy but if the stitching was perfectly fine but it was crossed or flipped over 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, after they had completed e-textile unit, students expressed feelings of comfort and competence with solving and designing problems in their interviews. All students interviewed said that the DbD unit should be done again next year because it was such a good learning experience for them. Some even asked for it to be done earlier and more often! As for what they appreciated about the unit, many students remembered making problems as fun and mischievous: creating problems "challenging enough to stress someone out is kind of funny and good" (Nicolás). They claimed that this gave them a "new perspective on coding" (Liam) that was the "opposite" from what they normally experienced, making it both challenging and interesting. Further, with some retrospective distance from solving DebugIts, many students also felt that solving others' bugs "was fun" because in figuring out the problems they learned how to "fix it ourselves" (Noah). Related to this, many students felt DbD helped them to feel more comfortable with problems in general. As Evelyn said, "I think 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 helped students feel better able to ask

for help from others since they became more aware that “a lot of people make mistakes” (Camila). This sense of capability or power over problems seems to have been most apparent after they were able to apply their learning from the DbD experience to a later project.

Not only did students feel more comfortable with problems, they also reported learning several important aspects about problems and problem-solving from the DbD unit. 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 in e-textiles. As Nicolás stated, “It helps them get a better understanding of the type of errors there are, like how errors can be prevented and caused and everything.” He and others said that they knew more problems: “common issues, common errors, more hard-to-see errors” (Liam). Knowing more problems also enabled students to avoid problems in their final project creation. Gabriel explained this succinctly:

> “[P]utting in your own bugs into a project and fixing them from another group… makes you more aware that the next time you're creating a project, that you know that those bugs exist and… you know whether you need to watch to 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 that arose in their final projects. However, one thing that was different for many students was that they were better able to identify problems more easily. Some students were able to see coding errors more easily because they were more familiar with interpreting compiler feedback on syntax errors. Others learned processes for detecting errors more effectively: “learning how to spot [errors]” (Logan), using “the process of elimination just see which part is a problem, is it the stitching or the code?” (Nathan), or simply “testing” thread lines or code lines to see if they worked (Lucas). Overall, nearly all students interviewed reported several ways that they applied new knowledge about problems and problem solving from the DbD unit in their final project.

Two other interesting areas of learning came up in students’ interviews about what they learned from the DbD unit. First, many students said that they actually learned new code. Most often this arose from debugging projects where three groups of students used coding techniques that had not been introduced to the entire class: fading lights, playing music, and using a light sensor (instead of a switch). Each of the students in the groups who received their projects reported learning new content about code, and some of the students who designed those projects also claimed to learn new content because of this design opportunity. In part these introductions of new code resulted from the teacher’s constructive feedback on designs, where he challenged certain groups to go further than they had in the mural project in the debugging project, demonstrating the value of this one-on-one design feedback in the mini-unit. Second, three students went further than others in describing an actual method of problem solving that they could apply to future e-textile projects. This often came up in how they helped other students after the DbD unit. As Liam described:

> “So when I was helping other people, I picked out important parts that we should look at first for them, and so I said, "look at the front of your code where you named everything." They should've named these correctly, and then after that, we looked at the threading.”

Here Liam described a methodical approach to identifying and solving problems in an e-textile project: looking first at the beginning of the code where students often mis-declare variables, then looking at the crafted circuits if the problem has not already been fixed. The methods these three students described demonstrate the beginnings of a systematic approach to debugging that is rare amongst novice coders (Simon, et al., 2008). The teacher had further insights into these findings.

Teacher Reflections on Classroom Implementation

The teacher’s daily and post-unit reflections, mirrored students’ experiences during the DbD unit and several weeks afterwards. Ben, the teacher, observed that students were highly engaged and

interested in designing buggy projects for each other. Overall, “[the students] seem very focused. I did not see a single group goofing off or stuck on an idea” (DbD Day 2). In fact, one of the students who previously tended to show disinterest in the class, was more involved than usual because he liked to be mischievous. Ben also suggested that designing bugs made problems less “painful” because “it’s like a puzzle and so you’re creating puzzles for people” (DbD Day 2). This made bugs into a “designed experience” rather than an accidental one that interrupted one’s personal design. Further, he found that designing bugs was a very valuable activity because “actually having to like think through all of the different things that could go wrong and trying to plan them in advance made some sort of like click in, in some of them” (post-interview). In other words, designing bugs forced students to think through them in a more intentional way than simply dealing with bugs as they came up in their personal projects.

At the same time, the teacher reflected that students’ early designs featured simple problems. To counter this tendency, he used constructive criticism during design approval to challenge students to think more creatively and to consider the recipients, or audience, of their buggy designs. Indeed, this thoughtfulness about audience was reflected in many students’ interviews (noted in the prior section) where they mentioned curtailing their bug designs to consider whether their peers would be able to fix them within a class period or whether the problems would be too many or too difficult for them to find.

When it came to debugging each others’ projects, Ben noted the “excitement” and “hussle bussle” amongst the students. At the same time, he noticed that students often used naive strategies to debug projects, taking more difficult approaches than necessary to solve problems. In contrast, *after* DbD he observed that students had a stronger knowledge base for constructing their final project. Even more, after the entire e-textiles unit was complete and students had moved on to the next unit, Scratch, Ben commented that for the first time ever, “[students] have a system of debugging. It might not be complete, you know... they're still learning how to build that system. But at least it now exists. Because it didn't before.” He expounded on this point in describing the ways that students were more independent in tackling problems in the Scratch unit than they had been in prior years, “meticulously and methodically trying to figure out what the bugs are,” making hypotheses for what might be wrong and applying those systematically to identify and fix bugs. This suggests that, at least in Ben’s class, the DbD experience helped students not only to apply knowledge about bugs and debugging to their e-textile designs but also supported them in more systematically debugging in another programming environment.

Discussion

In this paper we outline a new way to engage and empower students in debugging—by designing bugs for others to solve. Analysis of student and teacher reflections on the unit demonstrates high levels of interest and engagement with the DbD unit as well as benefits for after the unit. These benefits included a broader knowledge base of types of problems that can arise in designing and crafting e-textiles, improved abilities to detect and fix problems and collaborative problem solving with others, and finally greater confidence in debugging. The DbD approach may support students in near-transfer of debugging in what the teacher described as the beginning of a more systematic approach to debugging in at least one other domain after the e-textiles unit.

Our analyses of student and teacher reflections further suggests several important aspects of the design of the DbD unit, including constraints on the number and types of problems that students design, teacher feedback and approval process for the designs, and class-level reflections about types of problems, solutions, and problem-solving approaches. Attending to the audience of the designed bugs (i.e., for other students) as well as challenging students to be original in the problems they designed was important. Without these supports, students tended to create simplistic problems or problems that were too time-consuming to solve (i.e., that required substantial sewing to fix). Teacher constructive criticism allowed these naive conceptions of problem design to be a starting point but not an end point for students’ bug designs. Further, it was very important that this unit took place within a larger unit where students had significant time

to develop baseline knowledge of the domain (e-textiles) and problems that could arise as well as an entirely new project after the DbD mini-unit so that they could follow through on what they had learned. Thus the DbD mini-unit was a mid-point in the e-textiles unit, rather than a starting or an endpoint.

This paper represents our pilot efforts to design, understand, and study this new approach to debugging education. In upcoming months, we plan to study more closely the problems that students designed, their processes of design, and their processes of debugging. Further, our pilot implementation was conducted by one highly experienced teacher. Later studies need to look at how other teachers apply the DbD unit in different classrooms, the variances that occur, and best practices across multiple teachers for this approach. Debugging by design could also be applied to other domains in physical computing and general software design. We look forward to the continued explorations in empowering students to design bugs in projects in mischievous and creative ways and thus ultimately "becoming more articulate about one's debugging strategies and more deliberate about improving them" (Papert, 1980, p. 23).

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.

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.

Fields, D. A., Kafai, Y. B., Nakajima, T. M., 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, and Education,* 51(1), 21-35.

Fields, D. A., Searle, K. A., & Kafai, Y. B (2016). Deconstruction kits for learning: Students' collaborative debugging of electronic textile designs. In *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. (1990). *Children Designers*. Norwood, NJ: Ablex.

Jayathirtha, G., Fields, D., & Kafai, Y. (2018). Computational Concepts, Practices and Collaboration in High School Students' Debugging Electronic Textiles projects. *Proceedings of the International Conference on Computational Thinking Education 2018*. Hong Kong: The Education University of Hong Kong. 27-32.

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 *Proceedings of the 14th International Conference of the Learning Sciences*, Nashville, TN: International Society of the Learning Sciences.

Kafai, Y. B. (1995). *Minds in Play*. Mawhaw, NJ: Ablex.

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

Kafai, Y. & Fields, D. (2018). Some Reflections on Designing Constructionist Activities for Classrooms. *In Proceedings of the Constructionism Conference 2018*. Vilnius, Lithuania, 601-607.

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.

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

Kapur, M., & Kinzer, C. K. (2009). Productive failure in CSCL groups. *International Journal of Computer-Supported Collaborative Learning*, *4*(1), 21-46.

Litts, B. K., Lui, D. A., Widman, S. A., Walker, J. T., & Kafai, Y. B. (2017). Reflections on Pair E-Crafting: High School Students' Approaches to Collaboration in Electronic Textiles Projects. In *Proceedings of the International Conference of Computer Supported Collaborative Learning* 2017, Philadelphia, PA (2) 569-572.

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.

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.

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.

Simon, B., Bouvier, D., Chen, T.-Y., Lewandowski, G., McCartney, R., & Sanders, K. (2008). Common sense computing (episode 4): debugging. *Computer Science Education, 18*(2), 117-133.

Soloway, E., & Spohrer, J.C. (Eds.). (1989). Studying the novice programmer. Hillsdale, NJ: Lawrence Erlbaum.

Sorva, J., Lönnberg, J., & Malmi, L. (2013). Students' ways of experiencing visual program simulation. *Computer Science Education*, *23*(3), 207-238.

Sullivan, F. R. (2009). Robotics and science literacy: Thinking skills, science process skills and systems understanding. *Journal of Research in Science Teaching*, 45(*3*), 373-394.