International Society of the Learning Sciences

Scaffolding the Debugging Process in Physical Computing with Circuit Check

Michael Schneider, University of Colorado Boulder, Michael.J.Schneider@colorado.edu

Abstract: Physical computing projects provide rich opportunities for students to design, construct, and program machines that can sense and interact with the environment. However, students engaging in these activities often struggle to decipher the behavior of hardware components, software, and the interaction between the two. I report on the experiences of middle school students using a software tool, Circuit Check, designed to scaffold the debugging process in physical computing systems. Through think-aloud problem-solving exercises, I found Circuit Check facilitated rich instructor-student discussions. Incorporating these preliminary observations, I discuss design considerations for physical computing tools that support productive struggles and student sense-making.

Introduction

Physical computing engages students in creative learning where they design and create interactive systems, which incorporate sensors, actuators, and a programmable microcontroller. Enabling students to design and create their own projects encourages engagement in STEM learning, especially within under-represented populations (Buechley et al., 2008), but students struggle with making sense of the various hardware components and their relationship with their project's software. Their misconceptions can lead to bugs, where the system doesn't behave as the student expects. Working through the process of debugging their project, students will engage in an iterative process of progressively tuning both the system and their mental representation of it so that the system's behavior matches the student's expectations (Gilmore, 1991).

As novices, students are learning in tandem the skills to construct and debug their physical computing projects. Their inexperience can make the debugging process quite frustrating, and if the challenge of debugging becomes too great that frustration can lead to quitting or just stopping (Perkins et al., 1986). Providing proper scaffolding for debugging can be difficult for instructors, in that software tools for debugging physical computing are rather sparse (Desportes & DiSalvo, 2019) and present a goldilocks-like problem for beginners in that they either provide too few or too many features (Dolinay et al., 2021). For example, the original Arduino Integrated Development Environment (IDE) provides only the bare minimum for debugging, print statements. While a standard in software development, print statements can be difficult for beginners to interpret and use in debugging physical computing systems (Booth et al., 2016). To progress beyond print statements, students must turn to professional IDEs, e.g. Atmel Studio or Arduino IDE 2.0, which can be overwhelming, present a steep learning curve (Beller et al., 2018), and provide information beyond the beginner's abilities, such as viewing registers or program memory. Neither of these extremes adequately support novices through the struggles of debugging, but even worse they are also not designed to support instructor mediation of the debugging process.

I designed a software debugging tool, Circuit Check, to better balance the feature needs of novices, by enabling them to observe and test their system components with greater detail than provided by print statements but without the complexity of professional IDE's.

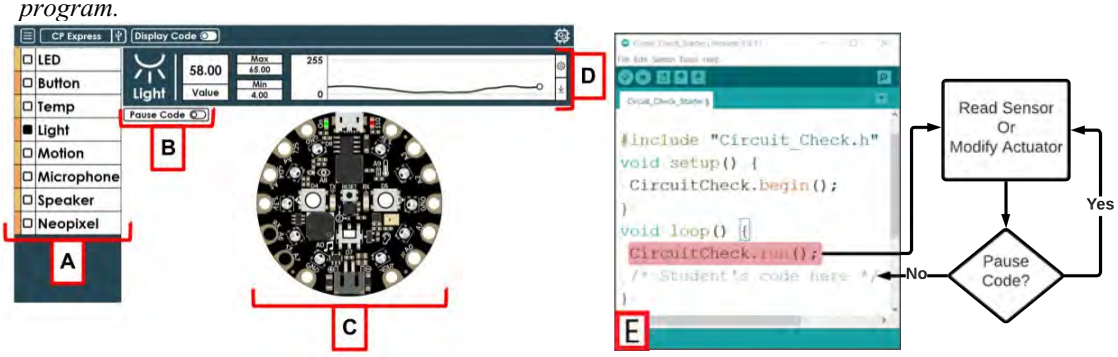
Circuit Check

Circuit Check is designed for system exploration, enabling students to better test their debugging hypotheses and observe their physical computing system's behavior. To enable system exploration, Circuit Check provides students with a graphical user interface (see Figure 1) where they can observe live data readings (e.g. light sensor) or test different actuator states (e.g. turning an LED on and off). By incorporating the Circuit Check library into their program (See Figure 1E) students gain access to this system information without needing replace or modify their code, preventing them from accidentally introducing new bugs while debugging (Vessey, 1985). This is because the Circuit Check library runs alongside the student's program, enabling the student to observe their system's state while their code is running. This differs from existing debugging tools, which are modeled after software IDE's and primarily focus on supporting code tracing and monitoring the values of variables with breakpoints. While these are important features, they are leaving out support for the *physical* side of physical computing. To further support testing their physical hardware components, Circuit Check not only enables students to observe their system while their code is running but also enables students to pause their program to isolate and test hardware components. As shown in Figure 2E, Circuit Check pauses the execution of the student's

program at the start of their forever loop, thus preventing the student's code from altering the hardware components while the student is testing them.

Figure 1

Screenshot of Circuit Check's user interface that shows a live readout from the Adafruit Circuit Playground Express's light sensor. A) Sensor and actuator selector, B) Button for pausing/resuming student's code, C) Interactive display of microcontroller, D) Sensor readout, E) The starter code for a student's Arduino program.



Methods and Study Context

I evaluated Circuit Check during two one-week long summer workshops in which a total of 16 students ages 10 - 14 designed and built e-textile projects. E-textiles are a sub-domain of physical computing where students construct not with wires and breadboards but with conductive thread and fabric, see Figure 2 for an example of a student project. Prior to the workshops, no student had previous experience with physical computing or e-textiles. To understand how Circuit Check supported students with making sense of their e-textile system, I followed a think-aloud practice (Ericson, 1998) where students actively discussed their thoughts and the reasoning behind their actions.

The summer workshops met for 3-hour sessions spread across four consecutive days. The first two days focused on lessons in creating e-textile circuits and programming, respectively. The remaining two days focused on supporting students as they designed and prototyped their projects, with the goal of having a completed project by the end of the fourth day. To create their e-textile project, students were provided conductive thread, a microcontroller (Adafruit Circuit Playground Express), LEDs, craft supplies (e.g. felt, fabric, markers, etc.), and a laptop for programming their circuits with the Arduino IDE.

Starter Project

During the first two days of the workshop students were guided through the creation of a starter project where they designed a constellation with LEDs, conductive thread, and their Circuit Playground (see Figure 2). To control the LEDs, students were taught to write a program that would have the constellation behave like a nightlight. Their program would read the light level of the room using the Circuit Playground's light sensor and either turn the LEDs on when the room was dark or off when the room was bright.

The Circuit Playground's light sensor provides an analog value between 0 (off) and 255 (bright). To understand how these values relate to how bright or dark a room is, students were instructed to connect their constellation project to Circuit Check and watch the light sensor's data display. While watching the display I had students imitate nighttime conditions by slowing covering the light sensor with their hand. From their observations, the students each decided on a boolean condition for their program that would interpret an appropriate range of light sensor values for night and day.

Final Project

After creating their constellations, students designed a personal e-textile project (See Figure 3). In programming their personal project, students were advised to explore the other built-in sensors provided by the Circuit Playground and pick at least one sensor for controlling their e-textile project's behavior with the two most commonly picked sensors being the light and sound sensors.

Findings

This section details a debugging episode in our second e-textile workshop. Dora envisioned a fabric flower with LEDs sewn into the petals (See Figure 2). The LEDs would turn on and off whenever she clapped, i.e. clap once

to turn the LEDs on and clap again to turn them off. But when she tested her prototype, it didn't behave as she intended. When she powered her project, the Neopixel LEDs built into her Adafruit Circuit Playground turned on and then stayed on, while her connected LED did not turn on at all. Dora had observed two bugs in her project: the LED she planned to sew into her project would not turn on and the on-board Neopixels would not turn on and off as she had expected. I asked Dora to hypothesize what might be causing the bugs she was observing.

Checking Code Execution

Dora's first hypothesis focused on the behavior of her Neopixels. She believed that "the code isn't really running". In talking through her bug hypothesis, I found that she believed that her code ran only once to turn on the Neopixels and then halted. She did not believe the code was running in a forever loop, because if it was, the Neopixels would turn off when she wasn't clapping or making loud noises. To test her hypothesis she used Circuit Check's pause feature, which paused her program at the start of the *loop()* function. This acts similar to a traditional breakpoint but with one key difference, the sensors are still able to transmit live data. Because physical computing systems are constantly taking in data, we can't create a traditional breakpoint which freezes and preserves the system's state. Instead, we only pause the student's code which allows them to verify their sensor readings and modify actuators. In this case, Dora paused her code and used Circuit Check to change the Neopixel colors. If her code was only running once, as she believed, then the Neopixels would remain set to their new color when the code was resumed. But when she unpaused her program the Neopixels all went back to bright yellow (the color her code sets). As I discussed with Dora, this showed that her if statement was being continuously executed, but the else clause, which turns everything off, might not.

Bug 1: A low threshold value

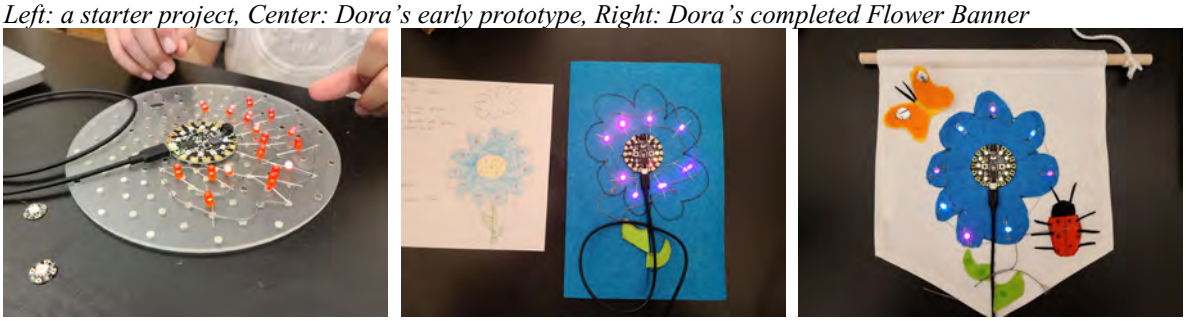
To understand how the *if/else* statements are being evaluated, Dora selected the Microphone sensor tab in Circuit Check's sensor menu. Here she watched live sensor readings from her microphone while her code was running. While observing the microphone readout Dora noticed that the ambient sound level in the classroom was a bit louder than 25. So although it did at times drop below 25, it would only do so for a fraction of a second. I asked Dora to monitor her microphone sensor and pick a new minimum sound level for her \textbf{if} statement. She tried a few different sound tests in which she clapped, tapped the table, and yelled to see how the sensor value changed over time. She settled on modifying her code to have a new minimum loudness value of 75. After updating the code, her Neopixels turned on when loud sound was present and off when the room was relatively quiet. The project still didn't behave exactly as she wanted, but she was a step closer to the desired behavior. Now that her Neopixels turned on and off, Dora turned her attention to her LED, which still did not turn on.

Bug 2: A short circuit

To verify there was a hardware error with her LED Dora ran two tests. First, she used Circuit Check to pause her code again and tried to turn the LED on and off. Using the Circuit Check web interface a student can turn an LED on, off, or change its brightness level. Dora tried to turn the LED on through Circuit Check's web interface, but after she toggled the switch to ON nothing happened. Because the LED would not turn on, it seemed likely that a hardware fault was the culprit. To check this, I gave Dora a simple voltage meter called a Logic Probe and she directly tested her hardware pin. Now when she toggled the hardware pin, the Logic Probe turned blue (LOW) or red (HIGH). As I explained to Dora, the meter showed that her microcontroller was providing power to the LED. This pointed to a potential issue with her circuit connections. Dora reconfigured her conductive thread traces by spreading the power and ground wires further apart. Once she reset the threads, the LED was able to turn on. This meant that her conductive threads (which are non-insulated) were probably shorting out. Next, Dora unpaused her code, and the LED was able to blink on with the Neopixels when she clapped her hands. After fixing the bugs in her project, I taught Dora to add a Boolean variable to her code so that project could clap on and then clap off.

Figure 2

Left: a starter project, Center: Dora's early prototype, Right: Dora's completed Flower Banner



Instructor Feedback and Discussion

Circuit Check has been designed around the needs of middle school STEM and science classrooms. Early design decisions were made with the input of two STEM teachers who teach an e-textile curriculum and the researchers who designed the e-textile curriculum. It was through their advice that the unique pause feature of Circuit Check was invented. Early prototypes incorporated a traditional breakpoint where students could view and modify variable values as well as the states for hardware pins (HIGH/LOW). The e-textile team brought up two major issues with this design, the amount of information being displayed and complexity of incorporating breakpoints into the student's program. Because their students lacked experience with working in an IDE, a busy user interface with variable data, sensor data, code, and breakpoint controls could overwhelm them. Instead I trimmed down the amount of information being displayed, creating the current interface shown in Figure 1. Next, I looked at how to present system information, without requiring the students to incorporate actual breakpoints. From the instructors' and researchers' experiences, students had difficulty building an understanding around how Arduino's infinite *loop()* functioned. Adding a breakpoint into their running program for debugging would add additional complexity, which would in turn could make debugging more difficult rather than less. Taking their feedback, I designed the *run()* function which would more easily integrate the debugging tools into the student's program.

Through these teachers interviews and student testing, I have found Circuit Check's integrated design enables self-guided exploration in debugging as well as improves instructor mediation in the debugging process. Specifically, Circuit Check removes the need for an instructor to micro-manage their students, e.g. place a print statement at line X to view sensor Y or write additional code to toggle your hardware pins on and off. Instead, instructors can respond to student questions with talk moves that support the cognitive demands of debugging. I am preparing further research into how Circuit Check can support the Productive Struggles Framework (Warshaeur, 2015) which details how an instructor can raise, maintain, or lower the cognitive demand of a problem-solving task when answering a student question.

References

Beller, M., Spruit, N., Spinellis, D., & Zaidman, A. (2018, May). On the dichotomy of debugging behavior among programmers. In *Proceedings of the 40th International Conference on Software Engineering* (pp. 572-583).

Booth, T., Stumpf, S., Bird, J., & Jones, S. (2016, May). 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* (pp. 3485-3497).

Buechley, L., Eisenberg, M., Catchen, J., & Crockett, A. (2008, April). The LilyPad Arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 423-432).

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

Dolinay, J., Dostálek, P., & Vašek, V. (2021). Advanced debugger for Arduino. *International Journal of Advanced Computer Science and Applications*.

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.

Gilmore, D. J. (1991). Models of debugging. *Acta psychologica*, *78*(1-3), 151-172.

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.

Warshauer, H. K. (2015). Productive struggle in middle school mathematics classrooms. *Journal of Mathematics Teacher Education*, *18*(4), 375-400.

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

Acknowledgements

This work was supported by the Stem+C Award #1742081 from the National Science Foundation. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation or the University of Colorado, Boulder.