# Trials and Tribulations of Novices Working with the Arduino

Kayla DesPortes
Steinhardt School of Culture,
Education and Human Development
New York University
New York, NY USA
kayla.desportes@nyu.edu

Betsy DiSalvo
College of Computing
Georgia Institute of Technology
Atlanta, GA USA
bdisalvo@cc.gatech.edu

## ABSTRACT

Physical computing has grown over the past decade leading to diverse experiences and tools for novices. Despite the variety of tools, Arduinos remain a leading choice in education. However, few studies examine how novices are learning about the programming and electronics concepts, and how tools impact their experience. The research presented reports on the qualitative analysis of a laboratory study in which 31 novices work with the Arduino for the first time. Video and audio recordings captured participants' actions and thoughts as they used the Arduino platform with a blocks-based programming environment, and two electronics prototyping tools—the standard Breadboard and a modular breadboard called BitBlox. The study presents three main contributions to the literature: first, it provides a codebook of the common breakdowns faced by novices; second, it offers insight into the work processes of novices; and third, it demonstrates ways that the tools used by novices can affect their experience.

## CCS CONCEPTS

• Social and professional topics ~ Computational science and engineering education

## KEYWORDS

Physical Computing, Arduino, Novice Programming, Think-Aloud, Blocks-Based Programming

**ACM Reference format:**
Kayla DesPortes and Betsy DiSalvo. 2019. Trials and Tribulations of Novices Working with the Arduino. In *Proceedings of the ACM International Computing Education Research conference (ICER'19). ACM, Toronto, ON, CA, 9 pages.* http://dx.doi.org/10.1145/3291279.3339427

## 1 Introduction

Physical computing continues to gain traction in computing educational spaces. The combination of the physical capabilities of the electronics with the power of computer science provides unique pathways to integrate computing into disciplines where the physicality is important to the experience. For instance, combining computing and sports [14], dance [11], and clothing design [7].

As the opportunities grow, so do the options for physical computing tools. The Arduino open-source prototyping platform now has over 25 variants [46]. Further, the tools have continued to drop in price—the Raspberry Pi Zero can run a Linux operating system and retails for $5 [34]. Physical computing tools have been developed to attend to specific environmental needs, such as the GoGo Board—built to be low-cost and locally produced [39], the Micro:Bit—designed with children in mind [21], and the Circuit Playground—created to minimize the tools necessary to create with the hardware [22]. Despite the numerous options educators now have, the Arduino remains a prevalent choice in education.

The Arduino platform was developed in 2005 to help artists and designers *build, tinker,* and *evolve* their work through providing access to programmable hardware [38]. Touted for its low cost and flexibility, the platform has integrated into a wide range of educational experiences from engineering [19], to dance [12], to entrepreneurship [45]. Though its use is widespread, the complexity of the tool has led researchers to criticize its use with novices [3]. In light of these concerns, it is important that research is directed towards understanding how to support novices in these environments such that we can develop the literature surrounding designing tools and scaffolds for novices in physical computing. The study presented in this paper closely examines how 31 novices are learning with the Arduino, investigating:

***RQ. What are novice students' experiences when working with the Arduino for the first time?***

*(i) What are the common breakdowns, misconceptions and obstacles of novices?*
*(ii) What are the work processes of novices?*
*(iii) How do the software and hardware tools affect novices' experiences?*

## 2 Background Literature

Physical computing has demonstrated the power to expand the concepts learners can engage with and the types of computing environments it can create [9, 30, 32, 35]. The literature illustrates promise for broadening learners' perspectives of computing [26] and diversifying the learners who are interested in computing [4]. However, working with programming and electronics together is difficult. Grover et al. found that while the Arduino was a low-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
*ICER '19, August 12–14, 2019, Toronto, ON, Canada*
© 2019 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6185-9/19/08...$15.00.
DOI: http://dx.doi.org/10.1145/3291279.3339427



RIGHTSLINK

cost and effective tool for their college mechatronics course, they concluded that in future work they would require prerequisites of programming and electronics [19]. Given its prevalence and complexity, we need to develop a better understanding of these learning environments.

## 2.1 Learning in Physical Computing

It was not until recently that researchers began to understand the types of learning that occur within physical computing learning environments. Wagh et al. for example, frame the learning in terms of the computational practices that manifest [41]. They found that the two most salient practices were (1) understanding the various hardware and software components and how they interact, and (2) debugging [41]. A complexity of factors affect learners' understanding of the components, such as the physical and virtual tools and how knowledge is dispersed in the socio-technical environment. The complications that arise have led some researchers to find ways around learning electronics by providing functional circuits so learners can focus on coding [37]. While this is a valid solution in some situations, it is also important to understand how to better support learners with the electronics and code. Some researchers have shed light on the complexities. Deitrick et al. aptly apply a lens of distributed cognition demonstrating how "a system of students, teachers, and tools" use the "physical and virtual affordances of different tools to organize work, externalize knowledge, and create new demands for problem solving" [9]. Their case study demonstrates how the design of tools are linked to the types of interactions they enable emphasizing the importance for the literature to build our understanding of technology's role in the learning environment.

The other prominent work in this space has focused on the difficulties of debugging in physical computing. Most notably the work centered on novice experiences with the LilyPad Arduino [18, 23, 26, 31]. Kafai et al. specifically integrate debugging exercises into their curriculum yet still find that it posed great challenges for novices [26]. Noting the importance of debugging, Fields et al. present a *deconstruction kit*, "in which students fix, or debug, strategically built-in problems" as a way to understand what students are learning [18]. Jayathirtha et al. explore the debugging practices of novices through documentation in portfolios and retrospective interviews [23]. Their study found bugs evenly distributed across coding, crafting, and electronics. Two thirds of the coding errors were *simple* involving syntactical mistakes and mislabeling of variables, while the other third involved complex conceptual errors such as correctly translating mathematical concepts within their code. The crafting errors dealt with the difficulties faced integrating the electronics into the materials. Depending on the physical computing application, the materiality poses a variety of problems for novices. The electronics presented equally challenging problems with learners having difficulties translating their own paper circuits into their physical circuits. While their reflective analysis provides insight into some of the errors, they do not draw out the nuances in the types of errors. The authors call for an examination of novices debugging *in the moment*. Our study does just this, providing real-time analysis of novices working with the Arduino.

## 2.2 Arduino Lab Studies

Three studies have begun to provide empirical evidence for understanding the types of complications that arise while building with the Arduino. Booth et al. investigate a set of experienced adult participants using an Arduino to control LEDs using a temperature sensor [5]. Despite having experience, only 6 of the 20 participants successfully completed the task. The participants ran into more software issues than hardware issues, however, the majority of issues that inhibited the participants from completing the task were hardware issues. The issues stemmed from using the hardware tools incorrectly and from their inability to debug their system. Booth et al. conclude that there is a need for better tools for debugging, and for understanding how to better educate the end-user developer [5].

Sadler et al.'s study is one of the few studies investigating the issues novices have with the Arduino [36]. The study analyzed video data from a study conducted by Jung et al. [24] in which 68 participants learned from an animated agent for 10min, then explored with the Arduino for another 15min. Sadler et al. reported that 47% of the participants could not get through the Blinky LED tutorial without researcher intervention. The most prevalent error was incorrectly connecting components using the breadboard—the same issue reported on in Booth et al.'s study [5]. They further identified hardware issues participants encountered, such as theoretical misunderstandings leading to incorrect circuits, leaving out wires in their circuits, misconceptions about how the breadboard works, and confusion over the hardware components. The participants also encountered syntactical and usability issues when using the text-based programming environment. Both Sadler et al. [36] and Booth et al.'s [5] findings convey a need to further understanding the tools.

Adding to this work, Booth et al. [6] closely explored the software, investigating blocks-based versus text-based languages when using the Arduino. In this study, each participant completed two tasks: one task involved remixing code to extend its functionality, and the other task involved creating code from scratch to complete a task. The participants were given the completed circuits to use with their code. They found that the task completion was low in both environments but lowest in the text-based environment; further the text-based environment correlated to a higher perceived workload and lower self-efficacy [6]. In terms of the specific errors participants encountered, the text-based environment contributed to added syntactical issues, which mirror findings in studies comparing blocks and text-based environments [42, 43]. Participants also had issues in the blocks-based environment—i.e. making the blocks do what they wanted them to, connecting blocks correctly, and navigating the variety of blocks in the environment. Both environments encountered issues with users having low visibility into their code [6].

In conjunction to highlighting the complexity of these learning environments, the studies present three key opportunities to build upon. First, while the Arduino is often used with novices, Sadler et al.'s work [36] is the only study that examined the real-time issues novices face. Due to the open-ended nature of the experiment, the findings were not linked to the participants' activities. This leaves an opportunity for understanding learners' work processes in a controlled task to contextualize the problems novices encounter. Second, the studies demonstrate hardware errors, but the studies do not closely examine how the design of the hardware related to these experiences. Similar to Booth et al.'s study examining the two software environments [6], there are opportunities to comparatively explore hardware prototyping tools and their impact on the learners' experiences. Third, the prior work demonstrates the difficulty of debugging. We provide insight into novices' debugging processes and how they might be improved.



RIGHTSLINK

# 3 Methods

The study comprised of 31 novice college students working through a set of activities to learn about and build with the Arduino. For the software, learners used a blocks-based programming environment called ModKit (Figure 3). For the hardware, learners worked with two tools: the standard Breadboard (Figure 1), and the BitBlox modules (Figure 2).

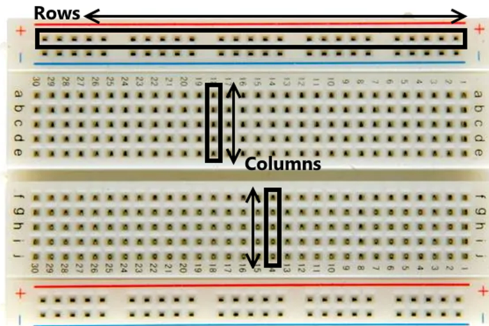



**Figure 1: Breadboard with outlines of connections schemes for the sections of electrically connected Rows and Columns**

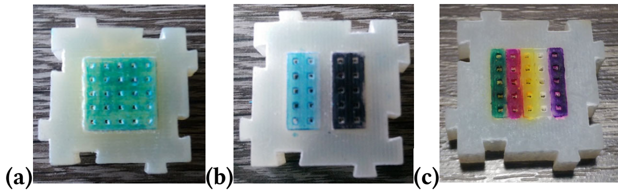



**Figure 2: BitBlox Modules (a) module with one section that is electrically connected, (b) module with two sections, and (c) module with five sections**

The tools work in the same way—i.e. plug in components to a section that is electrically connected. However, the Breadboard has two connection schemes: the outer rows, which are electrically connected and the inner columns, which are electrically connected. The BitBlox, have more variations in their connections and use colors to identify this. Users can snap them together building their prototyping tool as they build their circuits. The two tools were used to understand how novices interact with them and how their experience is shaped by them.

## 3.1 Participants

The study recruited novice participants from 8 universities at both the graduate and undergraduate levels. In order to qualify for the study, participants completed a pre-study screening survey that inquired about their previous experiences (informal and formal) with programming and electronics and their self-reported expertise in the subjects. Participants with any college course in electronics or programming were excluded, along with participants that rated themselves as having high expertise. Further, if they had any CS or multiple electronics courses in high school, or participated in extracurricular activities within the disciplines, they were excluded.

The 31 participants that qualified were admitted on rolling admission to the study and upon admission they were divided into two groups in order to determine which tool they would use first. Group 1 [G1] used the BitBlox then the Breadboard, and Group 2 [G2] used the Breadboard then the BitBlox. Within each group, five participants engaged in a concurrent think-aloud protocol, which required the participants to verbalize their thoughts while working through the study [17]. These ten participants provided added insight into the confusions and thoughts of participants. The participant demographics are outlined in Table 1.

**Table 1. Participant Demographics per Group.**

| | *Total* | *Gender* | | | *Level* | | *Age* |
|---|---|---|---|---|---|---|---|
| | Num. | M | F | O | UGrad | Grad | Range |
| ***Group 1*** | 15 | 5 | 10 | 0 | 13 | 2 | 18-25 |
| ***Group 2*** | 16 | 7 | 8 | 1 | 14 | 2 | 17-26 |

*M = Male; F = Female; O=Gender Fluid;*
*UGrad = Undergraduate; Grad = Graduate*

## 3.2 Study Protocol and Materials

The study protocol took about two hours per participant and consisted of eight steps. Depending on the participant's group, she/he would use a different hardware tool for a particular step.

**Table 2. Protocol for Study**

| |
|---|
| (1) Pre-Tests: Self-Efficacy and Knowledge |
| (2) Review of Electronics |
| (3) Introduction to Arduino and Modkit [Tool#1] |
| (4) Intro to Prototyping Tool and Blinky LED [Tool#1] |
| (5) Task #1: Blinking Two LEDs Separately [Tool#1] |
| (6) Introduction to Prototyping Tool [Tool#2] |
| (7) Task #2: Two LED circuit [Tool#2] |
| (8) Post-Tests: Self-Efficacy and Knowledge |

After the study was introduced and participants signed the IRB approved consent form, they began the experiment following a PDF guide on the computer, which took them through the study procedure. First, the participants took the pre-tests gauging their self-efficacy and knowledge. Next, the learners were provided a refresher to basic high school electronics covering how electricity flows through a circuit and the functions of LEDs, resistors and batteries as they constructed a circuit with alligator clips. The learners proceeded through an introduction of the Arduino, Modkit (blocks-based software IDE), and their first prototyping tool—G1: BitBlox; G2: Breadboard. They were then walked through step-by-step instructions on how to apply their prototyping tool and the code blocks to create Blinky LED—an introductory Arduino activity that makes an LED blink forever. Next, they completed the tasks, which were problems the learner had to complete without step-by-step instructions. The instructions for Task #1 and Task #2 are in Table 3 below. After the participant completed Task #1, they were provided an introduction to their second prototyping tool, and then completed Task #2. Next, the participants took the post-tests (same as pre-tests), completed a short interview and wrapped-up the study. The think-aloud participants went through the study first, and their data provided feedback to improve the PDF guide before the non-think-aloud participants entered the study.

The participants inevitably had difficulties as they were working and needed hints from the researchers. In the think-aloud participants, the researchers followed a protocol offering

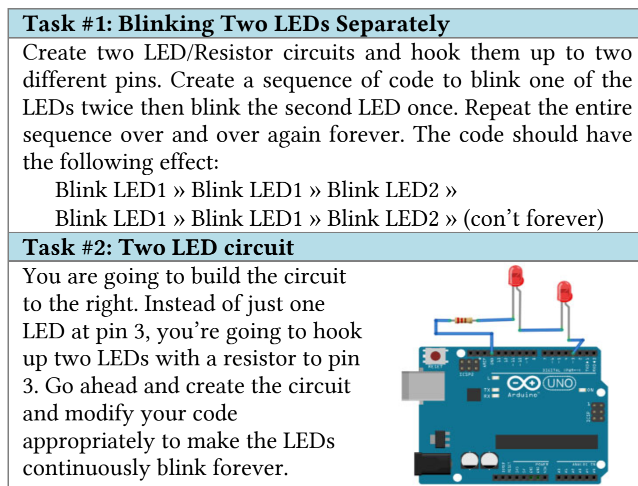


RIGHTSLINK

more specific hints each time. After the participant had been stuck for 1min or unsuccessfully debugging for 5min, the hints were provided in the following order: (1) identify the location of the problem (HW-hardware/SW-software/HW+SW) » (2) Provide general debugging hints for the location of the problem (ex. HW—make sure your wires are pushed down) » (3) Identify the specific issue » (4) Identify how they could solve the problem.

In order to minimize the researcher interactions in the non-think-aloud protocol, we created four sets of hints based on findings from the think-aloud group: (1) HW debugging hints, (2) SW debugging hints, (3) Task #1 hints, and (4) Task #2 hints. To receive the hints participants had to ask for them. The hardware and software hints consisted of a sheet of debugging checks the participants received all at once (ex. HW—make sure your wires are pushed down). The task hints were distributed one at a time and successively explained how to complete the task. The guide informed and reminded the participants about the hints.

**Table 3. Participant Tasks**



| **Task #1: Blinking Two LEDs Separately** |
| --- |
| Create two LED/Resistor circuits and hook them up to two different pins. Create a sequence of code to blink one of the LEDs twice then blink the second LED once. Repeat the entire sequence over and over again forever. The code should have the following effect:<br>Blink LED1 » Blink LED1 » Blink LED2 »<br>Blink LED1 » Blink LED1 » Blink LED2 » (con't forever) |
| **Task #2: Two LED circuit** |
| You are going to build the circuit to the right. Instead of just one LED at pin 3, you're going to hook up two LEDs with a resistor to pin 3. Go ahead and create the circuit and modify your code appropriately to make the LEDs continuously blink forever. |

## 3.3 Data Collection and Analysis

The data collection consisted of pre-/post-tests and video and audio recording of the sessions. The pre- and post-test were identical and measured the self-efficacy and the knowledge of the participants. Self-efficacy links one's perceived competence in a particular domain to perform tasks and overcome challenges [2]. The measure of self-efficacy in the domains of programming, electronics and physical computing, allowed us to understand how the tools might affect one's motivation to work with physical computing in the future. The self-efficacy test was composed of 21 seven-point Likert Scale questions constructed using four of the questions from Ramalingam and Wiedenbeck's computer programming self-efficacy scale [33]. The remaining questions use the same structure but the content was modified for our context. One question was accidentally clipped to a 5 point scale, but it was done in all of the think-aloud participants pre-/post-tests, so its impact was minimal.

The participants' knowledge was tested using a 15-point test. The first three questions (worth 2-points each) were adapted from Osborne's [29] method of drawing a complete circuit, which was previously adapted in a physical computing study [31]. This was supplemented by 9 multiple choice questions (worth 1-point each)—three questions required analysis of a circuit, three required analysis of code, and three required analysis of a circuit and code together. The test was designed to understand participants' ability to analyze forever loops, the effect of code on a circuit, the effect of delays in the code, pin numbers within the write functions, polarity of the LEDs, sequential reasoning in electronics and the effect of using multiple resistances.

In addition to the tests, 59 hours of video and audio data were recorded. Screen-capture data from the laptop documented how participants traversed the PDF guide and their interactions with the software. A webcam was synced to the screen capture and pointed at the participants' workspace to record the electronics (Figure 3). A backup camera provided a second perspective of the workspace. Since the non-think-aloud participants did not have the added audio to contextualize their actions, the researcher recorded notes to log interactions with the hardware in conjunction with the status of the code and circuit to link the notes to the video data.

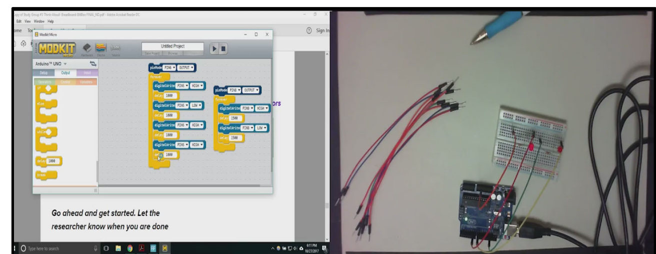



**Figure 3: Screen capture linked with webcam recording**

To analyze the data, one researcher coded one third of the video data for emergent themes and found that the coding scheme from Booth et al.'s prior study investigating skilled adult participants working with the Arduino matched closely to the identified themes [5]. Based on the initial coding, the researcher expanded the codebook to provide greater insight into the types of breakdowns that the novice participants encountered. Similar to Booth et al.'s study [5], the videos were coded for *obstacles*, *bugs*, and *breakdowns* and where the issues occurred (i.e. HW, SW, or a combination HW+SW). The *obstacles* referred to questions or confusions participants had. The *bugs* were errors that led to broken or improper functionality and were also coded for whether they were being introduced or fixed. While the bugs indicated when an error was introduced in the setup, a *breakdown* indicated whenever there was any type of misconception, mistake, or error even if it did not lead to the introduction of a bug. All bugs had at least one breakdown, but not all breakdowns led to bugs.

The researcher then applied the expanded codebook to the final two thirds of the data. To ensure validity of the coded data, a second researcher reviewed the codebook over transcriptions of 5% of the coded segments. The researchers clarified and refined the codebook and process for coding the data to ensure it was consistent and represented the data appropriately. The first and second researcher separately analyzed and coded another 10% of the video transcriptions using the updated codebook. The researchers reached an inter-rater reliability Pooled Kappa Cohen [8] score of 84% based on this analysis. The first researcher used the refined codebook (Table 4) to go back through the data, reviewing and correcting the rest of the segments. The final codebook resulted in seven software (SW) breakdowns, seven hardware (HW) breakdowns, and four breakdowns spanning the hardware and software (HW+SW).



RIGHTSLINK

**Table 4. Codebook of Breakdowns**

| SW | HW | HW+SW |
|---|---|---|
| Blocks Not Connected | Incorrect Circuit Formation | Programming without Plugging in the Arduino |
| Delay | Leaves out Component | Reprograms Arduino with the Same Code |
| Pin Signal (HIGH/LOW) | LED Backwards | Wrong Assessment of Bug Location |
| PinMode | Open Circuit | Incorrectly Thinks Problem is Solved |
| Sequentiality of Code | Short Circuit | |
| Wrong Arduino Pin | Wrong Arduino Pin | |
| Other | Other | |

# 4 Findings

## 4.1 Self-Efficacy and Knowledge Tests

We will briefly touch on the pre- and post-tests, then expand on the more relevant findings from the qualitative analysis. The scores were broken up by group (Group 1 [G1]: BitBlox » Breadboard; Group 2 [G2]: Breadboard » BitBlox) to see if the ordering of the tool had a significant impact. Further, they were separated by think-aloud [T] participants (5 per group) and non-think-aloud [NT] participants (10 per group) to understand whether having to think-aloud might have affected their scores.

Participants entered the study with a relatively low self-efficacy (between 53-62pts), and had a similar improvement in score across groups. Out of 145pts G1[T]'s average score improved by 50pts and G2[T] by 48pts. Similarly, out of 147pts, G1[NT] improved by 51.2pts and G2[NT] by 62.5pts. The knowledge pre-tests were correspondingly low. Out of 15pts, the averages across the groups were between 3-4.5pts. Of the 31 participants, only one correctly placed paper components of a battery, LED, and resistor and drew wires properly in order to make a working circuit, confirming the success of the pre-study screening survey to recruit novices. In the post-test, 26 of the 31 participants succeeded in this task. The average change in knowledge for the groups was—G1[NT] 7.1 | G2[NT] 7.4 | G1[T] 6.2 | G2[T] 9—with final scores between 9.2-11.5 out of 15. The think-aloud participants had the highest and lowest change in knowledge scores showing there was not a correlated drop in knowledge based on the learners having to think aloud. The spread also shows there was not a significant difference based on the tool participants used first.

## 4.2 Working with the Software

The participants encountered more breakdowns with the software than with the hardware. The six most common types of breakdowns were: referencing the wrong Arduino pin, misunderstanding how code would execute sequentially, incorrectly using (or not using) a pinMode to initialize a hardware pin as an input or output, incorrectly sending signals to the pins as HIGH (5v) or LOW (0v), incorrectly using or interpreting the delay block, and incorrectly connecting blocks in the IDE (see Figure 4). The *Other* category was for less prevalent errors.

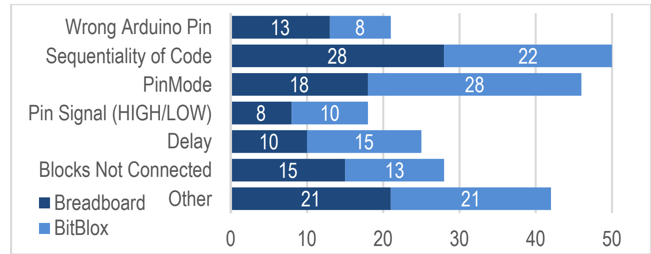



**Figure 4. Software Errors by Tool**

*4.2.1 Understanding Code Compilation and Execution.* The largest number of breakdowns in the software came from participants' lack of understanding of how their code would execute—i.e. *sequentiality of code.* The majority of these mistakes occurred during Task #1 where the participants had to create two circuits and make them blink in a defined sequence (i.e. Blink LED 1 → Blink LED 1 → Blink LED 2 [repeat forever]). The three most common errors were: (1) creating two chunks of code with two forever loops, (2) attaching code after a forever loop, and (3) nesting one forever loop inside of another. When Modkit compiles the code for two separate forever loops it does not run them at the same time, since the Arduino cannot support this. Instead, it runs the code that the user last *touched* but gives no indication of which piece of code this is. Participants ran into obstacles when something that was working would stop working because the participants *touched* another piece of code as they were working.

Outside of the misconception of how the computer interprets code, the thought process for creating these two chunks of code is inherently incorrect. If the two blocks ran in parallel, the LEDs would blink at the same time not sequentially, as the task instructed. Participants were generally unsure of how to add code for a second LED and often switched between several incorrect configurations, which gave varying responses based on which code was *touched* last and their current code configuration.

*4.2.2 Software Initialization of Hardware Pins.* The *pinMode* code block caused the second most breakdowns in the participants' reasoning. The block initializes the Arduino pins as either *input* or *output* so that they are setup to receive or send information. The two most common errors were: (1) not using a pinMode to initialize one of their pins as an output, or (2) using a pinMode and leaving it as an input instead of changing it to an output. Participants generally included at least one pinMode for the first LED, but then chose not to use it for the second or forgot it was necessary. One participant trying to figure out where she should put a second pinMode asked, *"can one pinMode go under another pinMode though? Maybe?"* Other participants put the pinMode into the into the forever loop unsure of whether this would work. Not knowing where the pinMode could go, led some participants to introduce a bug creating two chunks of code.

Some participants did not change the default initialization of the pinMode. While most participants overlooked the need to change from the default setting of *input*, some struggled with the general concept of input and output, *"yeah I still don't understand this whole input/output thing but I trust that it says output".*

*4.2.3 Delay Block: Avoidable Misconceptions and Indications of Being Stuck.* The delay code block demonstrated how poor explanations could cause misconceptions. The wording in the initial guide for the think-aloud section caused many participants to think that the delay caused the blinking of the LED. As one



RIGHTSLINK

participant reflected, "*hmm* [reading from the guide] *'we will now add a second digitalWrite but this time make it so we turn the LED off' see like that's where I'm kind of confused because if we want it to blink forever then why are we turning it off? cause like the delay is what makes it blink not off.*" After the 10 think-aloud participants completed the study, the wording in the guide was modified to describe the blink using an analogy of light switches and removed the wording that led participants to believe the delay caused the blinking. After this change, the misconception diminished—i.e. 6 of 21 non-think-aloud participants encountered it compared to 8 of 10 in the think-aloud.

The other common breakdown with the delay function was using it to fix other issues. For example, participants changed the delay to, get one of their LEDs to turn on, change the order in which their LEDs blinked, and to make an LED brighter. Often times when a participant would change the delay, it was indicative that the participant was stuck on an issue.

*4.2.4 Slips Using Software Tools.* Setting the wrong Arduino pin and sending the wrong pin signal were two of the breakdowns usually caused by slips rather than conscious decisions. These slips introduced bugs into the code. Participants would often forget to change the pin that the code block referenced, the signal that a digitalWrite was sending, or they would accidentally choose the wrong selection on the drop-down menu.

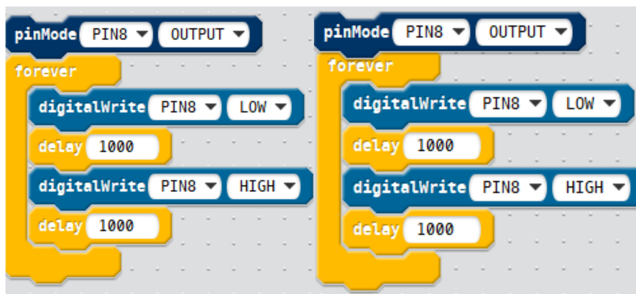



**Figure 5. Two pieces of code that demonstrate connected code blocks (left) and unconnected blocks (right)**

*4.2.5 Errors Connecting Blocks.* The last software breakdown was related to the usability of the IDE for connecting code blocks. While participants had few obstacles using the IDE, connecting code blocks consistently led to bugs in the code, which were difficult to discern. While the software has an indicator when you are about to connect blocks (i.e. a shadow under the block), there are no visible indicators to show you when blocks are connected (Figure 5). One participant who was debugging this issue eventually moved her blocks around fixing the issue, but she had no idea how, *"something changed. I don't know what I did...I don't think I changed anything."* These findings were consistent with the barriers Booth et al. identified using same software interface [6].

*4.2.6 Exploring New Code Blocks.* A number of participants had varying success as they experimented with new code blocks. In the guided activities, we covered the usage of four blocks—pinMode, forever, digitalWrite, and delay—which were the only blocks needed to complete the activities. However, upon exploring other blocks, some participants tried using them. Several participants identified and correctly used the repeat block to get the first LED in Task #1 to blink twice. The block used language linked to its usage and did not require additional blocks. This is consistent with prior work identifying the ease of using the repeat block [44].

When participants tried integrating other blocks, such as the *while* and *and* blocks, there were breakdowns in how they were interpreting the blocks. In Kaczmarcyzk et al.'s work they found that students would apply "real-world semantic understanding to variable declarations" [25]. Unsurprisingly, our work suggests that this is how they are initially determining the code blocks' functionality. Characteristics, such as the size and shape enabled participants to quickly rule out certain blocks, which physically did not fit with the other blocks. As one participant stated after unsuccessfully using a while loop, *"so maybe to make them blink at the same time you don't do while maybe you do and?* [she picks up the *and* block from the menu but does not see a way to attach it] *no that doesn't work."* Blocks that were more similar were more likely to end up in the code participants uploaded. However, they never successfully used any additional blocks besides *repeat*.

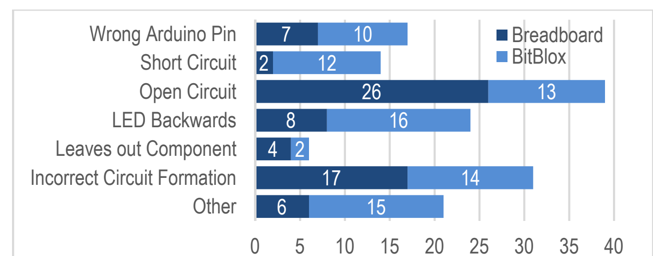



**Figure 6. Hardware Errors by Activity and Group**

## 4.3 Working with the Hardware

The types of hardware breakdowns provide insight into the errors that participants encountered when working with the electronics. There were six consistent breakdowns that emerged: using the wrong Arduino pin, creating a short circuit, creating an open circuit, assessing the LED directionality incorrectly, leaving out a component, and forming their circuit incorrectly (i.e. components in parallel or series when they should not be).

*4.3.1 Breakdowns Correlated with Prototyping Tool.* The most common breakdown was the *Open Circuit,* which when combined with the *Short Circuit* were the two breakdowns that manifested in different ways based on the prototyping tool used. The *Short Circuit* breakdown only occurred twice with the Breadboard, but occurred 12 times with the BitBlox. The participants introduced this error by incorrectly interpreting how to use the tool, purposefully plugging both ends of one component into the same section of the module. This was also noted in prior work [10].

The *open circuit* breakdown was twice as prevalent in the Breadboard than the BitBlox, but persisted across both tools. The participants using the Breadboard plugged components into holes that were next to one another but not electrically connected. Prior work noted similar open circuit issues with the breadboard [5, 36]. Open circuits in the BitBlox were sometimes caused because participants setup their circuits incorrectly; however, they more frequently occurred because the BitBlox that were not snapped together would spread apart and accidentally pull the components out of the blocks as they moved. Both of the tools indicated there were usability issues and a misunderstanding of how electricity was running through their circuits.

*4.3.2 Translating Circuit Connections.* The second most common breakdown was creating *incorrect circuit formations*—i.e.



RIGHTSLINK

placing components in series or parallel when they should not be. Of the 31 participants, 18 encountered this breakdown. In the first task, participants should have created two circuits with an LED and resistor (Figure 7 (left)), but they often combined both LEDs into one circuit (Figure 7 (right)). This mistake inhibits the LEDs from being able to blink separately. Participants in the second task encountered this error from connecting the LEDs and/or resistors in parallel instead of series (see diagram in Table 3). The participants either incorrectly interpreted how components in the diagram connected, or how components in their prototyping tool connected (and sometimes both). Understanding the purpose of certain circuit formations and translating the picture to their representational model of the circuit and then back to the prototyping tool proved difficult for the participants. The participants' issues seemed to be caused by a mix of theoretical misunderstandings and difficulties with spatial reasoning.

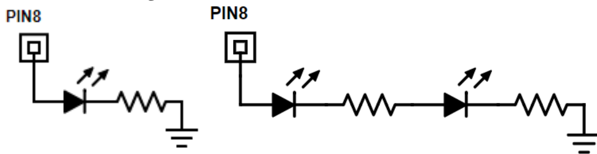



**Figure 7. Task #1 correct circuit formation (left) and common incorrect circuit formation (right)**

4.3.3 *Issues Using Small Components.* The small size of some of the hardware components led to breakdowns of participants using the *Wrong Arduino Pin* and inserting the *LED Backwards*, which led to bugs. These findings provide evidence for why participants in other studies had similar issues [5, 10, 36]. One participant demonstrates her difficulty assessing the positive side of the LED: "*The positive end is the longer side and the curved side. That one is longer…the whole thing is curved, so that's not really a good representation.*" Participants sometimes introduced a bug in order to check that the directionality of the LED was correct. One participant in the think-aloud group who was debugging an open-circuit explained why he switched the LED directionality: "*when I switched it I turned* [the] *positive side and negative side just to see if it would do anything different, but it didn't*".

Participants were also found counting the Arduino pins in order to make sure they were putting components in the right pin header. In the think-aloud group, participants had to rely on the labels on the board, while participants in the non-think-aloud had labeling directly on the pin headers themselves. Half the think-aloud participants, introduced a *Wrong Arduino Pin*, while only 6 of the 21 participants in the non-think-aloud section did.

4.3.4 *Correctly Parsing Visual Information.* Participants had obstacles correctly interpreting the relevance of certain visual cues. For example, not knowing whether or not the wire colors mattered, wanting to know if the "~" next to the digital pins (indicative of pulse width modulation capabilities) mattered, interpreting the pin numbers as voltages, or mistaking the blinking of the Rx and Tx LEDs on the Arduino (which blink when code is uploaded) for changes they created using their code. This also happened with the coloring between the different BitBlox. Participants sometimes thought they needed to find two separate BitBlox with the same colored section so that they could *properly* connect their components. These obstacles indicate that the participants are particularly attuned to the various visual cues in the hardware, but information can be misinterpreted.

4.3.5 *Importance of Work Processes for Usability.* The BitBlox had the ability to connect the various blocks by sliding the tongue and groove edges of the tool together. However, the video footage revealed that connecting the blocks did not fit seamlessly into their work processes. Participants often tried connecting the BitBlox together at the same time they were trying to insert the component into them (contrasted to connecting two BitBlox first then sticking the component into the blocks). The participants therefore usually had to readjust the parts for each component added to the circuit.

## 4.4 Working with Hardware + Software

The HW+SW breakdowns capture mistakes and misconceptions about the interaction between the hardware and software. There were four breakdowns in this category: incorrectly assessing where a bug was in their setup, reprogramming the Arduino with the same code that was already uploaded, programming code to the Arduino without the Arduino plugged in, and incorrectly thinking a problem was solved (Figure 9).

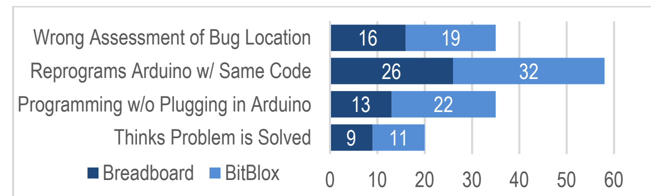



**Figure 9. Hardware + Software Errors by Activity and Group**

4.4.1 *Understanding Uploaded Code.* The most common breakdown was reprogramming the Arduino with the same code. This was a harmless breakdown and perhaps avoided bugs that could have occurred had participants not been as diligent in uploading code. However, it became clear that participants did not realize the Arduino stored the code on the board. Further, reprogramming the Arduino consecutively without changing anything often indicated participants were stuck.

4.4.2 *Slips in Plugging in the Arduino.* Participants would accidentally try programming the Arduino without plugging it into the computer. This was the only breakdown in this category that introduced a bug and could be harmful to the participants' success if not identified quickly. The Modkit software indicates there was an error programming, but participants did not always see this and if they did, they sometimes interpreted it incorrectly. Participants that were not able to catch the error quickly iterated through several versions of their code not knowing what was correct or incorrect. Not getting the expected feedback from their circuit skewed participants' interpretation of what would work in their code. One participant went through six different iterations of her code without uploading any of them to the Arduino. At one point during these iterations, she had the correct code, but never realized it and ended up not being able to complete the task. The learners' attention is often drawn in many directions and when they upload code they are often looking at the circuit for its response, which caused them to miss messages on the computer.

4.4.3 *Difficulties Debugging.* The breakdown of *Wrong Assessment of Bug Location* did not introduce a bug but led to the persistence of bugs. The issue usually occurred when the participant had a software issue and thought it was in their circuit.



RIGHTSLINK

For example, when participants created two forever loops or placed one forever loop after another, it led to a program that would only light up one of the participant's LED circuits. Not realizing their code was wrong, participants often assumed the issue they were having was in their circuit rather than with their code leading participants to make changes in their hardware. Participants would often guess and check certain configurations, which relied on buggy code or circuits, rather than narrowing down their problem based on what was working.

The other common breakdown while debugging occurred when the participant *incorrectly thinks the problem is solved*. This generally occurred in situations where participants would forget to initialize one of their pins as an output. When they would see the LED dimly blinking, they thought they were done the task. Other participants were confused if the circuits were responding the way they intended with their code. For example, if the participant did not turn an LED low before turning the second one high—i.e. one was turning on before the second one was off—the looping response became a difficult problem to reason about. Fast blinking LEDs from short delays exacerbated participants' issues.

The findings suggest that participants lacked methods to check parts of their setup, lacked knowledge of the types of errors they should check for, and did not realize the importance of thoroughly checking their assumptions. However, bugs that participants could easily check for, such as the directionality of the LED were constantly being defaulted to.

## 5 Discussion

### 5.1 Designing around Usability

The findings demonstrated how usability issues, could become more salient and more complicated with the combination of hardware and software. In some cases, there were straightforward solutions, such as enlarging the electronic components, or labelling parts more clearly. In other cases, the issues posed more complicated design challenges, such as needing to understand how learners are interpreting the components. For example, the color and shape of the code blocks provided some indication of how they should be used, but participants mostly interpreted their functionality incorrectly. Learners also struggled understanding the error messages from the compiler and in some cases did not even notice them. Research needs to build a better understanding of how tools could and should communicate with the learners.

The findings also illustrated that participants were particularly receptive to visual information, but had difficulty understanding and parsing the information. The Arduino's visual design has small text, and a number of functions and information labelled on the board. The design of these boards could benefit from pairing down the amount and types of information presented, while enlarging the design to provide more visible and comprehensive information. To aid in this, it could be beneficial to frame the design around multiple boards that guide users through a learning trajectory, rather than expecting one board to serve all purposes. This supports prior work, which has called for *bridges* [3] and *glass-boxes* [13] between the current tools.

### 5.2 Designing around Processes and Practices

The findings also highlighted the need for the environment and tools to be responsive to the processes novices bring into the learning environment. Many of the participants' were making decisions as they tinkered with the components, modifying their choices based on feedback (i.e. as a *bricoleur* [40]). The findings showed how the technology, could inhibit the learner's process if not designed from this perspective. For example, when working with BitBlox participants had to reconfigure their existing components when adding new components. The tools need to take into consideration novices *just-in-time* designs, assisting learners in making and testing their iterative modifications.

Learners also had processes in which the tools could be designed around. When participants were stuck they often would continuously upload the same code (similar to [28]), and consecutively change the delays. While these actions were often innocuous, data on what *getting stuck* looks like could inform designs for technology to offer hints or suggestions.

There were also processes that we need to learn more about, such as the participants' difficulties translating between different representations of their circuits. The issues seemed to stem from issues with spatial reasoning, usability, and theoretical misconceptions. Jayathirtha et al. [23] noted similar issues, confirming a need to understand how to scaffold these processes.

The designs of the tools and materials should account for learners developing new practices—specifically debugging. Similar to other findings in CS [1, 16, 28], debugging was not something the participants were able to do without guidance. Even in Booth et al.'s [5] study debugging was difficult for experienced participants, and in Kafai et al.'s [26] study, where they teach debugging skills, it still remained a challenge. We need to think critically about the ways that the tools can facilitate learners to identify and test their assumptions, develop systematic debugging processes, and understand how to use the tools to isolate components of the hardware and software to validate what in their design is correct and what is not.

### 5.3 Designing around Visibility

Participants' low visibility into how the tools were working often exacerbated conceptual errors. In the software, participants lacked insight into how computer was compiling and interpreting their code, leading to a misunderstanding of what code was running. The looping structure, inherent to many microcontrollers, also caused confusion. Researchers in CS education have examined personifying the compiler [27] and aggregating useful suggestions based on other programmers actions [20]. Our research confirms this as an important consideration in physical computing environments as well.

Similar to the software, participants lacked visibility into the hardware. Unless everything was setup correctly, there was little feedback into what was electrically connected, where voltage was being applied, or where current was actually flowing. Multimeters are the most prevalent tool for this, but they have steep learning curves and their own usability challenges. Researchers are beginning to experiment with other ways to bring visibility to these signals [15], which could improve conceptual understanding as well as usage of the tools.

## 5 Conclusion

The complexity of the physical computing environment presents a challenging design problem. Our findings suggest that the design of technology should continue to be explored to improve usability, more seamlessly integrate and develop the practices and processes of novices, and provide learners with greater visibility into the tools and how their designs are functioning.



RIGHTSLINK

# REFERENCES


[1] Ahmadzadeh, M. et al. 2005. An analysis of patterns of debugging among novice computer science students. *ACM SIGCSE Bulletin.* 37, 3 (Sep. 2005), 84. DOI:https://doi.org/10.1145/1151954.1067472.

[2] Bandura, A. 2006. Guide for constructing self-efficacy scales. *Self-efficacy beliefs of adolescents.* 5, 307–337 (2006).

[3] Blikstein, P. 2015. Computationally Enhanced Toolkits for Children: Historical Review and a Framework for Future Design. *Foundations and Trends® in Human–Computer Interaction.* 9, 1 (2015), 1–68. DOI:https://doi.org/10.1561/1100000057.

[4] Blikstein, P. 2013. Digital fabrication and 'making'in education: The democratization of invention. *FabLabs: Of Machines, Makers and Inventors.* (2013), 1–21.

[5] Booth, T. et al. 2016. Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task. *Proceedings of the 2016 Conference on Human Factors in Computing Systems* (2016), 3485–3497.

[6] Booth, T. and Stumpf, S. 2013. End-user experiences of visual and textual programming environments for Arduino. *International Symposium on End User Development.* (2013).

[7] Buechley, L. and Eisenberg, M. 2008. The LilyPad Arduino: Toward wearable engineering for everyone. *Pervasive Computing, IEEE.* 7, 2 (2008), 12–15.

[8] De Vries, H. et al. 2008. Using Pooled Kappa to Summarize Interrater Agreement across Many Items. *Field Methods.* 20, 3 (Mar. 2008), 272–282. DOI:https://doi.org/10.1177/1525822X08317166.

[9] Deitrick, E. et al. 2015. Using Distributed Cognition Theory to Analyze Collaborative Computer Science Learning. (2015), 51–60.

[10] DesPortes, K. et al. 2016. BitBlox: A Redesign of the Breadboard. *Proceedings of the The 15th International Conference on Interaction Design and Children* (2016), 255–261.

[11] DesPortes, K. et al. 2016. Interdisciplinary Computing and the Emergence of Boundary Objects: A Case-Study of Dance and Technology. *International Society of the Learning Sciences.* (In Press 2016).

[12] DesPortes, K. et al. 2016. The MoveLab: Developing Congruence Between Students' Self-Concepts and Computing. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (2016), 267–272.

[13] DesPortes, K. and DiSalvo, B. 2017. Where are the Glass-Boxes?: Examining the Spectrum of Modularity in Physical Computing Hardware Tools. (2017), 292–297.

[14] Dittert, N. 2014. TechSportiv: constructing objects-to-think-with for physical education. *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational* (2014), 569–577.

[15] Drew, D. et al. 2016. The toastboard: Ubiquitous instrumentation and automated checking of breadboarded circuits. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (2016), 677–686.

[16] Edwards, S.H. 2004. Using software testing to move students from trial-and-error to reflection-in-action. *ACM SIGCSE Bulletin.* 36, 1 (Mar. 2004), 26. DOI:https://doi.org/10.1145/1028174.971312.

[17] Ericsson, A. and Simon, H.A. 1980. Verbal Reports as Data + Ericsson.pdf. *Psychological review.* 87, 3 (1980), 215.

[18] Fields, D.A. et al. 2016. Deconstruction kits for learning: Students' collaborative debugging of electronic textile designs. *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education* (2016), 82–85.

[19] Grover, R. et al. 2014. A competition-based approach for undergraduate mechatronics education using the arduino platform. *Fourth Interdisciplinary Engineering Design Education Conference* (2014), 78–83.

[20] Hartmann, B. et al. What Would Other Programmers Do? Suggesting Solutions to Error Messages. 10.

[21] Inspire every child to create their best digital future: 2019. *https://microbit.org/about/.* Accessed: 2019-04-01.

[22] Introducing Circuit Playground: 2016. *https://learn.adafruit.com/introducing-circuit-playground/overview.* Accessed: 2019-04-01.

[23] Jayathirtha, G. et al. 2018. Computational concepts, practices, and collaboration in high school students' debugging electronic textile projects. *Proceedings of International Conference on Computational Thinking Education* (2018).

[24] Jung, M.F. et al. 2014. Participatory materials: having a reflective conversation with an artifact in the making. *Proceedings of the 2014 conference on Designing interactive systems* (2014), 25–34.

[25] Kaczmarczyk, L.C. et al. 2010. Identifying student misconceptions of programming. *Proceedings of the 41st ACM technical symposium on Computer science education* (2010), 107–111.

[26] Kafai, Y.B. et al. 2014. A Crafts-Oriented Approach to Computing in High School: Introducing Computational Concepts, Practices, and Perspectives with Electronic Textiles. *ACM Transactions on Computing Education.* 14, 1 (Mar. 2014), 1–20. DOI:https://doi.org/10.1145/2576874.

[27] Lee, M.J. and Ko, A.J. Personifying Programming Tool Feedback Improves Novice Programmers' Learning. 8.

[28] Murphy, L. et al. Debugging: The Good, the Bad, and the Quirky – a Qualitative Analysis of Novices' Strategies. 5.

[29] Osborne, R. 1983. Towards modifying children's ideas about electric current. *Research in Science & Technological Education.* 1, 1 (1983), 73–82.

[30] Papert, S. 1993. *Mindstorms: Children, computers, and powerful ideas.* Basic Books.

[31] Peppler, K. and Glosson, D. 2013. Stitching Circuits: Learning About Circuitry Through E-textile Materials. *Journal of Science Education and Technology.* 22, 5 (Oct. 2013), 751–763. DOI:https://doi.org/10.1007/s10956-012-9428-2.

[32] Perlman, R. 1974. "TORTIS: Toddler's Own Recursive Turtle Interpreter System.

[33] Ramalingam, V. and Wiedenbeck, S. 1998. Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research.* 19, 4 (1998), 367–381.

[34] Raspberry Pi/Pi Zero/Pi Zero V1.3: 2019. *https://www.adafruit.com/category/934?src=raspberrypi.* Accessed: 2019-04-01.

[35] Resnick, M. 1995. New Paradigms for Computing, New Paradigms for Thinking. *Computers and Exploratory Learning* (Berlin, 1995), 31–43.

[36] Sadler, J. et al. 2017. Building blocks in creative computing: modularity increases the probability of prototyping novel ideas. *International Journal of Design Creativity and Innovation.* 5, 3–4 (Oct. 2017), 168–184. DOI:https://doi.org/10.1080/21650349.2015.1136796.

[37] Searle, K. et al. 2016. The E-Textiles Bracelet Hack: Bringing Making to Middle School Classrooms. *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education* (Standford, CA, 2016), 107–110.

[38] Severance, C. 2014. Massimo Banzi: Building Arduino. *Computer.* 47, 1 (2014), 11–12.

[39] Sipitakiat, A. et al. 2004. GoGo board: augmenting programmable bricks for economically challenged audiences. *Proceedings of the 6th international conference on Learning sciences* (2004), 481–488.

[40] Turkle, S. and Papert, S. 1990. Epistemological pluralism. *Signs: Journal of Women in Culture and Society.* 1, 16 (1990), 11.

[41] Wagh, A. et al. 2017. The Role of Computational Thinking Practices in Making: How Beginning Youth Makers Encounter & Appropriate CT Practices in Making. *Proceedings of the 7th Annual Conference on Creativity and Fabrication in Education* (Standford, CA, 2017), 1–8.

[42] Weintrop, D. and Wilensky, U. 2015. To block or not to block, that is the question: students' perceptions of blocks-based programming. *Proceedings of the 14th International Conference on Interaction Design and Children* (2015), 199–208.

[43] Weintrop, D. and Wilensky, U. 2015. Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. *Proceedings of the eleventh annual International Conference on International Computing Education Research* (2015), 101–110.

[44] Weintrop, D. and Wilensky, U. 2015. Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. (2015), 101–110.

[45] Zhang, X. et al. 2018. Design and Practice of Arduino Experiments for "E&I" Oriented Education. *Proceedings of ACM Turing Celebration Conference-China* (China, 2018), 21–26.

[46] 2019. List of Arduino boards and compatible systems. *Wikipedia.*




RIGHTSLINK Copyright Clearance Center