What to Teach First, Hardware or Software? Improving Success in Introductory Programming Courses

Dr. Richard Whalen, Northeastern University

Dr. Richard Whalen is a Teaching Professor at Northeastern University in Boston, MA and is Director of First-year Engineering. The mission of the First-year Engineering team is to provide a reliable, wide-ranging, and constructive educational experience that endorses the student-centered and professionally-oriented mission of the University. He also teaches specialty courses in the Department of Mechanical and Industrial Engineering at Northeastern and has published and presented papers on approaches and techniques in engineering education.

Dr. Joshua L. Hertz, Northeastern University

Dr. Hertz earned a B.S. in Ceramic Engineering from Alfred University in 1999 and then a Ph.D. in Materials Science and Engineering from the Massachusetts Institute of Technology in 2006. Following this, he worked at the National Institute of Standards and Technology as a National Research Council postdoctoral fellow. He joined the Department of Mechanical Engineering at the University of Delaware as an Assistant Professor in September 2008, leading a lab that researched the effects of composition and nanostructure on ionic conduction and surface exchange in ceramic materials. In 2014, he moved to Northeastern University to focus on teaching and developing curriculum in the First Year Engineering program.

What to Teach First, Hardware or Software? Improving Success in Introductory Programming Courses

Abstract

This complete evidence-based practice paper presents an analysis and lessons learned in introductory engineering courses with content that includes problem-solving, algorithmic thinking, the use of microcontrollers, and C++ at a medium-sized private urban university. These courses specifically incorporate the integration of hands-on, project-based design projects with computer programming. The goal of the project work is to provide an authentic experience and give students the opportunity to develop process-driven problem-solving skills. A large focus of these classes is developing algorithmic thinking skills, and an introduction to computer programming has been used to facilitate meeting this objective. In addition, with the ubiquitous use of microcontrollers and platforms such as Arduino, faculty now can integrate hands-on experiences with hardware to motivate student learning. This paper presents the results of qualitative and quantitative analysis of two ways to introduce programming concepts and the use of microelectronics in a first-year engineering course. In one approach, students are first taught algorithmic thinking and programming in C++ in a traditional sense, without an introduction to hardware applications. Once they have gained facility in the programming language, they then apply this knowledge to hardware applications. In an alternative approach being piloted during this study, students are introduced to programming and algorithmic thinking via the hardware applications; the material is introduced concurrently instead of sequentially.

Findings from pre and post-surveys indicate that students taught using both approaches had similar improvements in self-efficacy to code and build projects with basic circuitry. In addition, most students appreciated the approach used in their class; if taught with a hardware-first approach, they thought a hardware-first approach provides greater learning, and if taught with a software-first approach, they thought a hybrid software-first approach provides greater learning. Most students expressed little frustration in learning the material using either approach. Of those who did express frustration, most suggested that using the other approach would have led to increased frustration in learning the material.

Keywords

Programming, First-year Engineering, Programable Microcontrollers, Arduino

Introduction

At Northeastern University College of Engineering, all first-year students follow a common curriculum, as part of a “Cornerstone to Capstone” educational program adopted in 2014 [1]. The first-year Cornerstone course uses projects to emphasize the ways in which engineering can develop practical problem-solving applications. In Cornerstone, there are essential topics in which students should become competent across disciplines, including effective communication, teamwork, design thinking, knowledge application, technical skills, and problem-solving [2]. The Cornerstone course was carefully designed to help first-year students achieve success in the program and develop the necessary technical and professional competencies regardless of the specific engineering major they select in their second year [3]. There is a strong emphasis on applying technical knowledge in a practical way, developing analytical problem-solving and decision-making skills, and demonstrating resourcefulness. As a result, students tend to be more vested in the learning process, appreciate what they have learned, remain engaged, and retain more of the material [4-7].

One of the main elements of the Cornerstone course is coding and algorithmic thinking. As many instructors know, learning to code can be very intimidating to many students. At Northeastern University, this aspect of the Cornerstone course centers on practical, code-based solutions to real-life problems using Mathworks' MATLAB and the C++ programming language. In the past decade or so, low-cost microcontrollers such as Arduino have increased student engagement and brought programming to life. Students enjoy the tactile, real-world use of their new programming skills, and introducing microcontrollers to the learning of computer programming helps overcome the educational challenges typically seen in first-year courses that include programming. In comparison to the teaching of design, teaching programming is more difficult due to the wide range of students' past experiences. Approximately 73% of incoming students have Advanced Placement (AP) credits along with a range of high school STEM experiences from 'none to advanced' such as First Robotics or Vex Robotics. The first-year is common to all engineering students and Cornerstone is populated by students who have declared from all engineering majors, students who select engineering but are undeclared in their major and students who are undecided in their college and explore class offerings from across the university. With such a wide range of experiences, faculty struggle with students that sometimes feel inadequate as they have difficulty understanding new programing concepts while their fellow classmates with experience seem to just get it with ease. This has been increasingly evident with the introduction of microprocessors and hardware to go along with learning software development.

In order to infer best practices, this study examines a pilot approach in teaching programming and algorithmic thinking with hardware applications within the context of an introductory engineering program. In the traditional approach, students are first taught algorithmic thinking and programming in C++ in a traditional sense, without an introduction to hardware applications. Once they have gained facility in the programming language, they then apply this knowledge to hardware applications by programming and building Arduino projects. In the piloted approach, students are introduced to programming and algorithmic thinking *via* Arduino applications; basic programming concepts and the microprocessor hardware and circuitry are introduced concurrently instead of sequentially.

The purpose of this paper is to outline the differences of the two approaches and present the student learning outcomes from each. The research questions the paper aims to answer is whether the piloted approach provides equivalent learning outcomes and course satisfaction. The paper will also describe the applications used in the various projects and analyze the above forms of assessment to qualify the approaches. Additional takeaways include how both the students and instructors are affected by each approach and the lessons learned along the way to make both successful.

Literature Review

There is ample evidence in the literature of programs designed to implement a hands-on, low-cost, easily integrated design component in a first-year engineering course used to emphasize the importance of understanding how software and hardware are interlaced, increase retention and student satisfaction [8] [9] [2] [10] [11]. The byproduct of this approach is increased student engagement often in a meaningful way by making a strong connections to the many embedded computing applications used in students' everyday experience and in society in general [4] [12]. In addition, the use of these components provides an increased opportunity to teach lessons in troubleshooting and conducting experiments in order to systematically solve a problem [13]. Another byproduct of using these components is that they open the door to solving a wide variety of problems and research has shown the positive impacts of giving students choice on which problems to attack along with an increase in student satisfaction [14]. The general approach in many of these courses is to begin with developing a student's problem-solving skills and introducing them to structured software. Programming languages such as Matlab, C++, Python and others are used as the tools to teach algorithmic thinking [15-18]. In another possible approach, such as a computer science course used to introduce students to the computer environment, students might be exposed to Ohm's Law, logic gates, and how these elements can combine to form basic computer hardware, then spend time working on code and an application like being able to control a radio controlled car [19]**.** The use of short theory sessions followed by immediate practical hands on sessions of "Learning by Doing" was shown to increase student performance on final exams and student satisfaction [20]. In whichever approach is used, the goal is to bring design and programming to life in order to increase student self-efficacy, and better retain and improved student perceptions of the value of computer programming in the first year [21-23].

Background

First-year Engineering Program

The first year is common for all engineering majors and there are two general engineering courses offered each semester. In total, about 30 separate sections are run with approximately 30 students in each. The Cornerstone 1 and 2 courses specifically incorporate the integration of hands-on, project-based design projects along with computer programming and the use of microcontrollers. Project-based Cornerstone has, as one of its challenges, the ability to have incongruent learning of course content due to the nature of problem solving and design. By highlighting that engineering problem solving brings together groups of competencies in a networked fashion rather than in a linear fashion, we can help increase the quality of instruction for all students by showing them that this incongruence is acceptable. Specifically, the emphasis is that Cornerstone is a lens by which engineering learning can come together to develop practical applications to solving problems.

The Cornerstone 1 course focuses on learning the principles of engineering and design; this is accomplished through active learning in areas such as problem definition, conceptual design, preliminary and detailed design, design communication and implementation, engineering ethics, along with report writing and presentations in relation to projects that students produce in teams. There is a strong emphasis on applying technical knowledge, developing problem-solving and decision-making skills, and using computer-aided design (CAD) to communicate graphically.

Within this course, algorithmic thinking and programming with C++ is introduced along with the basic use of microcontrollers. Procedural programming using functions is covered in order to facilitate the use of Arduino based micro-controllers and basic components such as LEDs, RGB LEDs, Potentiometers, Piezoelectric speakers, servos and motor controllers, LCD screens, RFID and various sensors such as those used to measure temperature, pressure and force. All these elements are taught to help facilitate the solution to the design problem at hand.

Cornerstone 2 continues with finishing elements of C++, namely data structures, and continues problem solving and programming with Mathworks' MATLAB and the further use of Arduino based microcontrollers. In addition, value sensitive design and ethical theories are introduced on the design side along with 3-D design software. This study does not include students taking Cornerstone 2.

In the first-year program prior to 2014, instruction in the design elements was separate from the programming and microcontrollers elements. The first course in this series, General Engineering and Design, focused on design, ethics and graphical communication only. In the second course, Problem Solving and Computation, students developed their algorithmic thinking skills using the software tools of Mathworks' MATLAB, C++ programming language and Arduino-based microcontrollers. This two-course sequence has all of the same learning objectives and topics as the Cornerstone 1 and 2 sequence and is still in use today because of the need to accommodate transfer students who may have programming or design experience but not both, students who are undecided in college and are exploring various disciplines, and as part of an Engineering Minor for non-engineers. Both these courses offer a project and teams-based experience for our students and cover all the same material as the Cornerstone sequence.

In this study two student cohorts were used, one from Cornerstone 1 and one from Problem Solving and Computation. In both courses, only the C++ programming and microcontroller experiences are compared, and in both courses these subjects are taught at the beginning of the course outline with no other topics introduced. The sequencing of content in each of these courses is summarized in the next two sections. The subject matter and electronic components are the same in both. The approach used to introduce the subject matter is different and is the focus of this study.

Cornerstone 1 – Software-first Instruction

The Cornerstone 1 course is centered around a robotics theme where students are tasked to design an autonomous Sumo Wrestling Battlebot. In order to facilitate the design students must become familiar with the operation of motors and sensors needed to control the robot. They are also introduced to the design process, engineering ethics and graphical communications tools needed to document the design as the semester progresses. At the beginning of the course, students first begin to learn how to program in C++ so they can then begin to learn how to wire components to a breadboard and program the Arduino microcontrollers to build their robot designs. The process starts with an introduction to programming, data types and variables and basic input and output over the course of one week. Students move to problem solving and writing codes that require decisions in the next week. Week three in programming introduces students to the concept of loops. Finally, in week four students learn about user written functions along with random number generators. At this point students have now practiced programming and begin to use a specialty

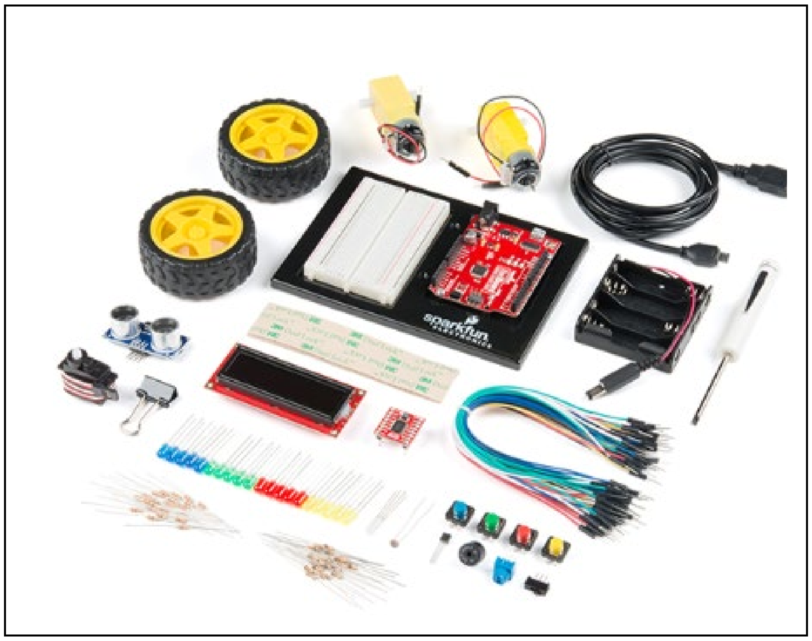
Figure 1. Sparkfun Inventors Kit.

kit of electronic components provided by Sparkfun Electronics called an Inventors Kit as shown in Figure 1. The main kit components include an Arduino based microprocessor called a Redboard, a motor driver, gear motors, servo motor, ultrasonic distance sensor, TMP36 temperature sensor, photocell, Tricolor LCD, assorted color LCDs, buttons, power switch, piezoelectric speaker, resistors, LED display, various wires and wheels. Students begin to learn about basic circuits, breadboards, programmable microcontrollers and the use of the Arduino IDE. In addition, a robot chassis is provided along with reflective sensors, ultrasonic distance sensors and servo motors that are used as an initial platform in the robot builds.

The process of learning the basics of Arduino is accomplished by completing 3 mini projects which are outlined as follows. In project 1, students begin with blinking a LED and the use of digital ports on the Arduino board, then integrate a potentiometer to control the blink rate, a photocell and the Serial Monitor to learn how analogue ports work. From there they are introduced to RGB LEDs. The final assessment has the students complete a challenge project of making a holiday lights string that can blink at varying rates and colors and turn on and off when it is dark or light out. In project 2, students are introduced to a piezoelectric speaker, button use, an LCD and a temperature sensor. In the final assessment students work to create a thermostat with feedback messages. In project 3, motor controllers, motors, servos and an ultrasonic distance sensor are introduced. The final assessment has the students complete a challenge project to design a fill level controller with various lights, buttons and status messages. This process takes about 2 weeks and students then begin the basic sumo robot build.

The robot competition has two phases. The initial build phase where students focus on getting all the hardware to work and look to develop a working algorithm to give their robot a competitive edge. In this phase all robots use only the base robot kit and look the same but will have widely different programming strategies. The robots will use reflectance sensors, ultrasonic distance sensors and servo motors to control how they operate in the sumo ring. After this first competition the second phase begins, and the focus is to integrate new components and redesign the robot from the ground up to gain a competitive advantage. Here students get to redesign all systems and build a new robot to a set of constraints defined only on size and weight. The students then have a final competition with their new robot at the end of the term.

Problem Solving and Computation – Hardware-first Instruction

The Problem Solving and Computation course is centered around an espionage theme where students are tasked each week to work on a homework problem (a "Mission") that ties that week's learning objective to a spy/espionage storyline. As an example, students must write a program that includes encryption and decryption functions (using one-time pad cryptography) during the week where C++ strings and functions are the learning objectives.

The first four weeks of class are used to teach code fundamentals through the Arduino platform. The goal of this approach is to make coding concepts, which can seem abstract to many students, more tactile and easier to conceptualize through physical input and output. This approach also allowed students to create useful, physical, code-enabled objects very quickly in their learning and so reinforces the practical utility of what they're learning. This process begins at the second class of the semester, when students learn breadboard and circuitry basics. By the end of the class, students can wire an LED circuit on the breadboard and power it from a 5 V source.

The focus of the next three classes is to learn the basics of what code is and how it works in Arduino C++: minimal required code structure, code editor vs. compiler, basic Arduino functions (pinMode, digitalWrite, and delay), comments, and declaring and using integer variables. All of these are taught using an LED circuit on the breadboard as the only required circuitry. The first homework asks students to integrate all of these ideas together and create a box with 2 LEDS that transmit the student's initials in visual Morse code. After this, the next class focuses on Arduino output functions beyond digitalWrite so that the LED can have variable brightness and so that sound can be output. Learning these helps keep students engaged at this point, since simply blinking lights starts to lose interest.

The next two classes focus on Boolean logic and branching through if and if/else structures. These topics are taught very naturally using a button, so the requisite circuitry and Arduino functions for this are also taught. The second homework asks students to create a lockbox of sorts, with three buttons, two LEDs, and a piezo speaker on the breadboard. Students must connect all of these components to the Arduino and write code that tracks how many times the buttons have been pushed by the user. Different lights and sounds are output when the buttons are pushed the correct or incorrect number of times.

The following week—the fourth of the semester—rounds out the hardware-based introduction to coding. An analog sensor, a photoresistor implemented with a voltage divider circuit is taught. The resulting numerical values are used in the code within range-based if/else-if/else logic to create a kind of night light and "theremin-like" musical instrument. The Serial Monitor is also taught so that these objects can be calibrated to a particular setting. A final homework asks students to integrate all the learned concepts into a smart device that responds to its environment. The device is a “magic” candy box themed to Halloween (in the Fall semester) or Valentine’s Day (in the Spring semester) where sounds and lights change as the box is opened.

At this point in the semester, students have used the Arduino ecosystem to learn not just basic circuitry but also nearly all of the fundamentals of coding: basic structure, comments, variables, branching using if-statement structures, integer arithmetic. We then spend the next four weeks getting deeper into C++ writing programs that run on the computer. Students are quickly able to apply their Arduino learning to create sophisticated programs. Additional topics we add these weeks are: floating-point values; looping with while, do/while, and for; pseudorandom numbers; functions; strings; and arrays. Finally, in the last third of the class, students learn MATLAB as an alternative programming language and problem-solving system which is not part of this study.

A final project requires students to use their Arduinos to create a security system for a chosen scenario. Students propose what hazards they want to protect against, such as fire, intruder, loss-of-power, or theft of the device, as well as how the relevant information is output from the device, such as text on an LCD screen, LED strips, or sounds. The projects are required to use at least two electronic components not learned in class. Most students work on these projects during the last two to three weeks of class.

Methodology

The main data collection for this study was in the form of student surveys. All students were asked to complete surveys near the start and again near the end of semester. Surveys were collected via the class learning management system. Students who responded to both surveys had their responses brought together, and all responses were then anonymized before analysis. Pre-course surveys asked students about their prior experiences and self-efficacy with coding and with microelectronics and circuitry. Post-course surveys asked similar questions and added questions about how students think a hardware or software first approach would affect student learning and student frustration. The surveys also asked for student gender and year of study (first-year student, second-year student, etc.) to see if these were confounding factors in their survey responses. The survey questions for the pre-survey and post-survey can be found in Appendix A and Appendix B, respectively.

The surveys were not graded or strictly required, but response rates were nevertheless high. Response rates to the pre-survey and post-survey were 89% and 79%, respectively. The final net response rate of students who completed both surveys was 71% (24 / 34) for students in the hardware-first class and 79% (44 / 56) for students in the software-first class. The results presented below consider survey responses from only those students who completed both surveys.

Results and Discussion

While the student populations in the two classes were similar in some respects, they were sufficiently different such that it is difficult to draw conclusions by direct comparison of the survey results from the two populations. As shown in Table 1, both the hardware-first and software-first classes were close to even gender balance with no significant differences (*$p=0.63$*). The hardware-first class was 58% women, while the software-first classes were 52% women. No student declined

Results: Demographics *N*=**68**	**Software First**	**Hardware First**	**Total or Weighted Average**
Fully Completed Surveys	44	24	68
Gender: Male	47.73%	41.67%	44.7%
Gender: Female	52.27%	58.33%	55.3%
Gender: Other	0%	0%	0%
Factors & Tests Conducted	**Values & Statistical Significance in Outcomes**		
Gender: M ◆ F ◆ O Expected, χ^2:	45.58 ◆ 54.41 % ◆ 0%		*NSD, p= 0.63*
Year Classification, 1st, 2nd…: ***t-test*:**	1.01	2.37	*SD, p<0.001*

Table 1. Student Demographics.

to report their gender or chose to self-report a gender other than man or woman. The age of the students was significantly different *(p<0.001)* between the hardware-first and software-first classes, so this cannot be ruled out as a confounding factor in the results. The hardware-first class had 0 first-year students, 18 second-year students, 4 third-year students, and 2 fourth-year or later students. The software-first class consisted of 41 first-year students, 2 second-year students, and 1 third-year student. The classification into first-year, second-year, etc. was purely chronological based on when students matriculated as a full-time university student, rather than credits earned or other basis of university standing.

In the hardware-first class, 29% of the students reported having prior experience writing computer programs and 5% (1 student) reported prior experience making circuits. Both of these values are lower than in the software-first class, where 34% of the students report having written computer programs before and 25% reported prior experience making circuits. This result is somewhat surprising, since the hardware-first class was older on average by about 1 year.

Figure 2 shows histograms of student responses to the question about self-efficacy on projects involving coding. In both classes, the largest group of students reported feeling "not at all confident" in their abilities when the semester began (50% hardware-first, 48% software-first). By the end of the semester, however, no students felt this way anymore, and the largest group of students in both classes reported now feeling "moderately" or "very" confident (76% hardware-first, 84% software-first). The post-survey results are very similar in both classes with no significant differences *(pre-survey, p = 0.39, post-survey, p = 0.58)*. Therefore, both the hardware-first and software-first approaches were successful in improving student self-efficacy to desired levels on this primary learning objective.

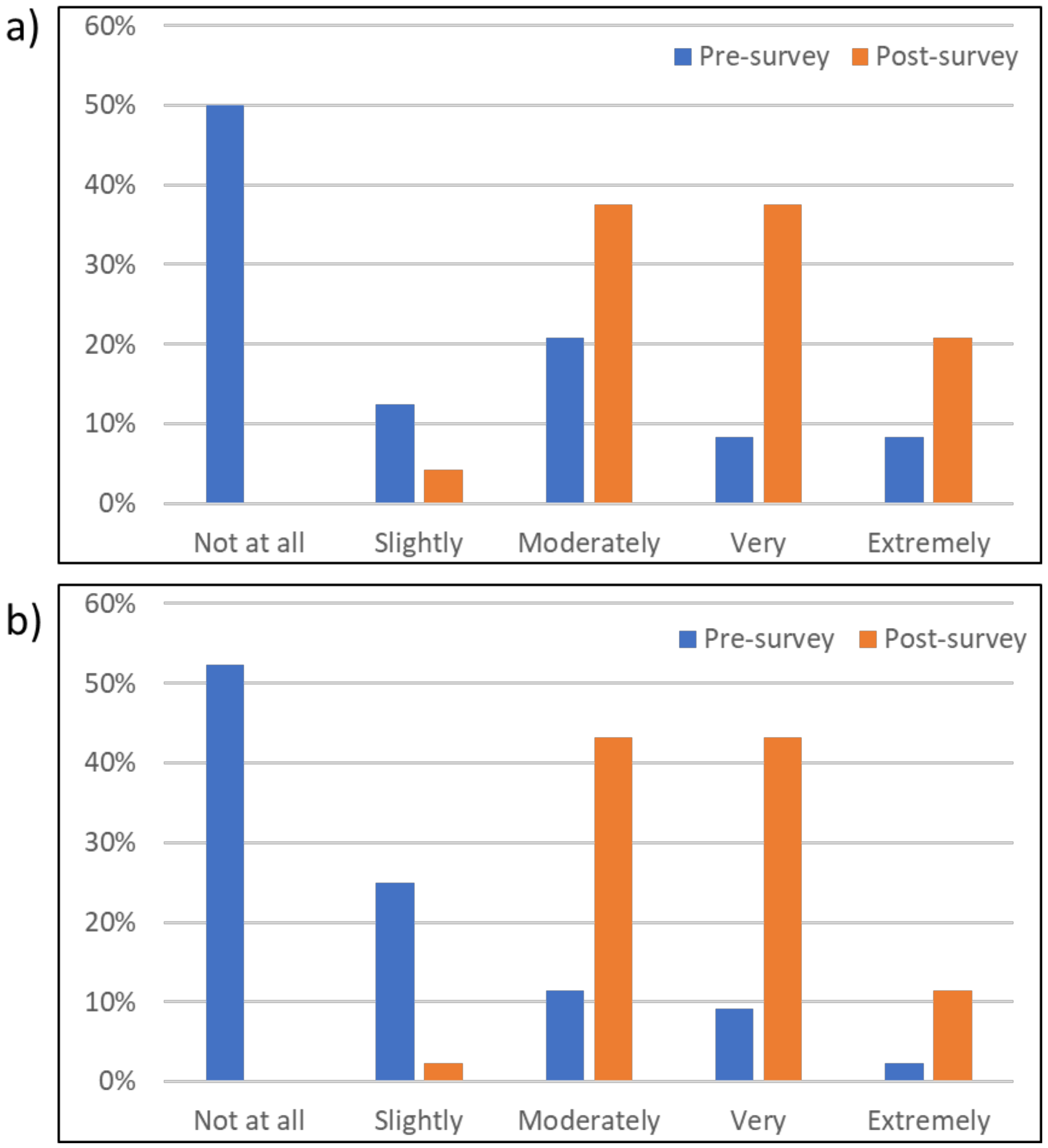

Figure 2. A histogram of responses from students in the (a) hardware-first class and (b) software-first class to the question about how confident they would be to start working on a graded project that involves computer programming in the pre- and post-surveys.

Students also increased in confidence in their ability to create circuitry. Figure 3 shows histograms for self-efficacy on projects that involve circuitry. Unlike the previous question, the result here shows a perceptible difference, however not significant *(pre-survey, $p = 0.25$, post-survey, $p = 0.17$)* between the hardware-first and software-first classes. Once again, students in both classes report low confidence in the pre-survey, with the largest group of students in both classes feeling “not at all confident”; however, the proportion of students reporting no confidence is larger in the hardware-first class, 63%, relative to the software-first class, 48%.

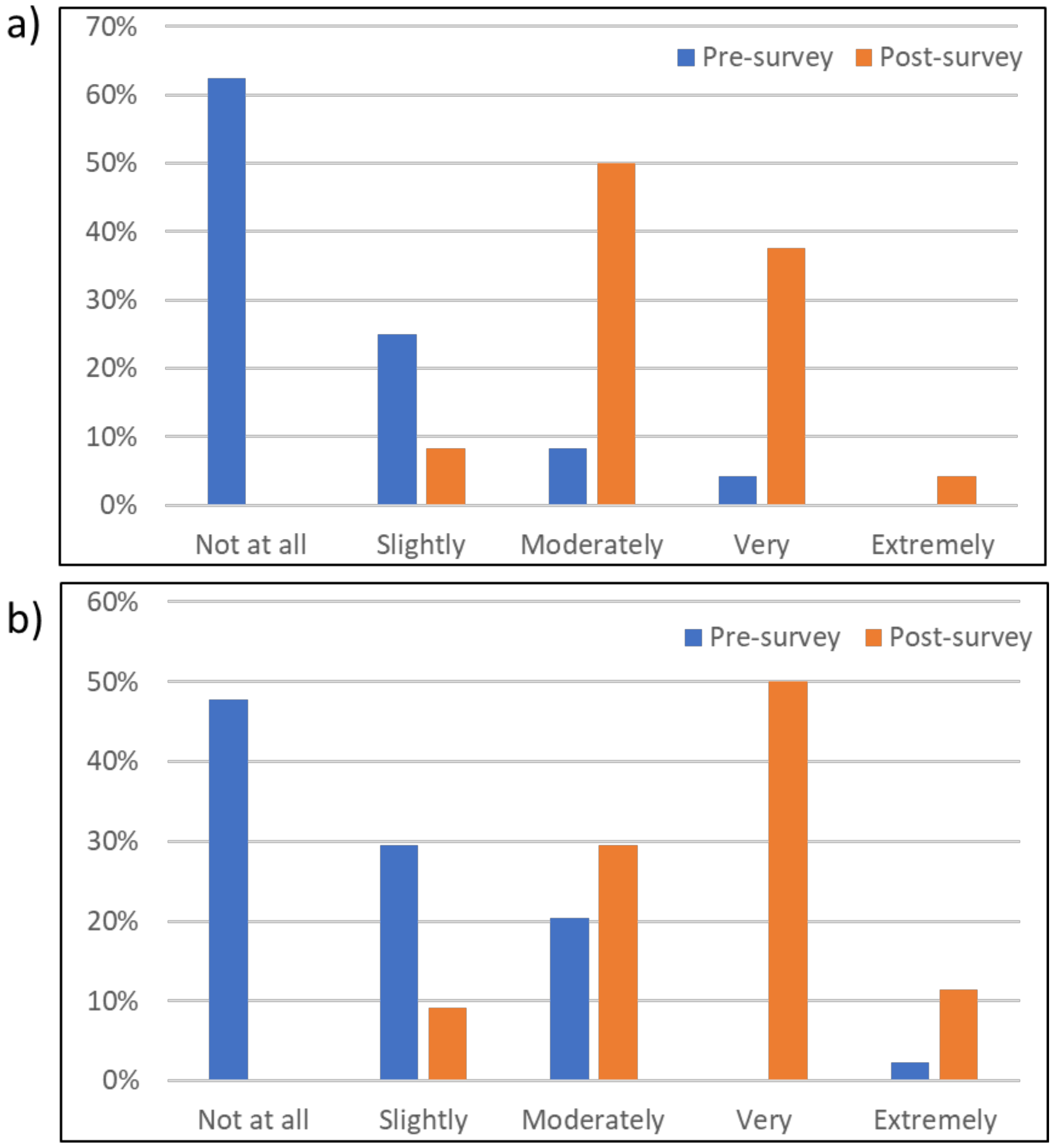

Figure 3. A histogram of responses from students in the (a) hardware-first class and (b) software-first class to the question about how confident they would be to start working on a graded project that involves circuitry in the pre- and post-surveys.

Though all students reported increased confidence by the end of the semester, students in the hardware-first class remained somewhat lower relative to the software-first students. "Moderately confident" was the most often selected choice in the hardware-first class at 50%, whereas "Very confident" was most often selected in the software-first class at 50%. In the "Extremely" category the results had a similar pattern with 11% for students in the software-first class and only 4% for those in the hardware-first class.

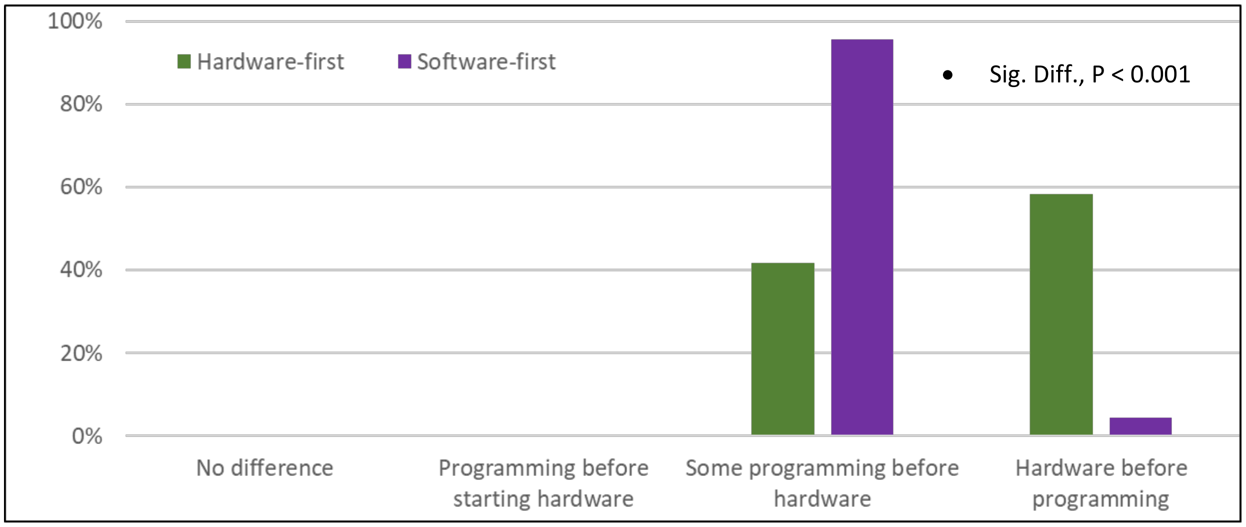

Figure 4. Student responses to a post-survey question about which approach would lead to more learning.

In the post-survey, students in both classes were asked their opinion about whether a software-first, hardware-first, or hybrid approach would help students learn more. No student in either class thought that the choice of which approach to use would be inconsequential, and no student thought that it would be better to teach all the programming before starting any of the hardware. In choosing between the remaining two options, hybrid or hardware-first, there were significant differences *($p<0.001$)* in responses between the two classes. As can be seen in figure 4, more students in the hardware-first class thought the hardware-first approach led to better learning outcomes (59% to 3%), although 40% of the students thought learning some programming before moving on to the hardware would help. In the software-first class, nearly all students thought that learning some programming before the hardware would lead to the best learning outcomes. To summarize these results, students taught with a hardware-first approach mostly thought a hardware-first approach provides greater learning, and students taught with a software-first approach mostly thought a hybrid software-first approach provides greater learning. The students in the hardware-first class, though, were more equivocal in their opinions about this.

When asked whether a change in approach might have led to less frustration, most students in both courses—71% of the students in the hardware-first course and 61% of the students in the software-first course—said they felt no frustration with the approach used with no significant difference *($p = 0.66$)* as shown in figure 5. Of the remaining students, more students in each course suggested that the approach used in their class is the one that would, in fact, have led to less frustration. It is hard to know how to interpret this result. In the case of the "hardware-first" class, only one day was spent purely on circuitry. After this, the remainder of the first four weeks taught programming on the Arduino, *i.e.*, programming instruction was simultaneous with instruction in hardware and circuitry. Therefore, one interpretation for these students is that they would have preferred additional circuitry-only instruction before learning any code. In the case of the "software-first"

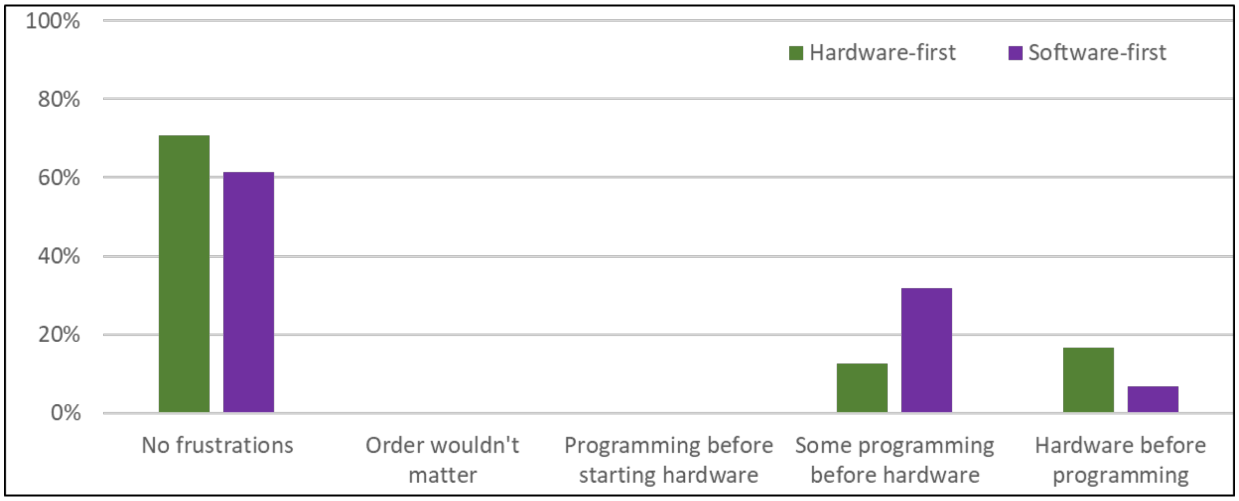

Figure 5. Student responses to a post-survey question about which approach would lead to fewer student frustration.

course, a third of the students total reported that "introducing the hardware basics after some introduction to programing would have helped me be less frustrated." Since the option given in the survey stated "introduction" to programming, these students might be indicating that the hardware instruction might be moved to earlier in the class, after only an introduction to programming. In both classes, roughly 10% of the students stated that the approach taken by the other class would have led to less frustration.

The work described here represents the first time a hardware-first approach was attempted. As is commonly the case when teaching new material or teaching existing material in a new way, the co-author piloting this approach found many small areas where students were confused and would have benefitted from additional scaffolding or adjustments in the content delivery. Thus, the second attempt in teaching with a hardware-first approach will likely see students experience reduced frustration and easier learning outcomes. As an example, integer variables and arithmetic were taught in the same class day. These were taught together this time mostly because they were taught together during the previous times that the instructor taught programming, and these concepts are easily demonstrated together on a text-output program running on a computer. On the other hand, when using the hardware-first approach and so learning to code on the Arduino, physical output (*e.g.*, an LED turning on or off) is the result of the program, and arithmetic results are harder to palpably see. Thus, arithmetic will be delayed to later in the content schedule in future iterations.

The study's results also have led to the second co-author rethinking the software-first curriculum. In previous studies as discussed in the literature review show that adding hands-on elements increased student engagement and motivation. It seems logical then that introducing these elements earlier in the curriculum might lead to better learning outcomes and an increase in student self-efficacy. The author intends to now stagger the programming elements and hardware projects in a hybrid model instead of doing all programming first and hardware second. This approach will require a redesign of some of the projects as any use of Arduino will require the students to at least understand decisions and loops in programming.

Conclusion and Recommendations

In this study, we investigate student perspectives on their experiences in learning basic procedural programming and applying these skills to control hardware via an Arduino based microcontroller. Students from two separate classes were taught the same material but in a different order. In the first class, students first learn all the necessary programming elements before applying these to the hands-on microcontroller projects. In the second class, students learn concurrently both the programming and hands-on microcontroller elements. Responses to a pre and post-course survey are used to analyze students' experiences, self-efficacy with coding and with microelectronics and circuitry. Also, in the post-course survey students are asked about how an approach would affect student learning and influence any student frustration.

Analysis of the pre and post surveys indicate that students taught using either approach had similar improvements in self-efficacy to code and to build projects with basic circuitry. Therefore, both the hardware-first and software-first approaches are viable options to successfully teach software and hardware integration in a first programming course. Regarding a preference, students taught with a hardware-first approach mostly thought a hardware-first approach provides greater learning, and students taught with a software-first approach mostly thought a hybrid software-first approach provides greater learning indicating a bias towards the approach currently being used. This is not a surprising result as our students readily adapt to the learning environment in which they are put in. However, we should still acknowledge the fact that just because they are resilient and adaptable it does not mean the approach we use is the best one and we should always be open-minded to change. Finally, most students expressed little frustration in learning the material using either approach and for those who did more suggested that using the other approach would have led to increased frustration in learning the material

Even though there may be confounding factors of age, programming and microcontroller experiences between the software-first and hardware-first classes, these findings show that there will always be more than one way to approach meeting learning objectives in instructional design. As educators we must always be willing to try new approaches and sometimes after assessment, we will find that we make a sound case for instructional change while at other times it might be inconclusive or a step backwards. Nonetheless, there is value in trying and as in this study we will continue to look to improve our approach to introducing students to both software and hardware applications moving forward.

References

[1] S. F. Freeman *et al.*, "Cranking Up Cornerstone: Lessons Learned from Implementing a Pilot with First-Year Engineering Students," presented at the 2016 ASEE Annual Conference & Exposition, Jun. 2016. Accessed: Jan. 06, 2023. [Online]. Available: https://peer.asee.org/cranking-up-cornerstone-lessons-learned-from-implementing-a-pilot-with-first-year-engineering-students

[2] K. A. Dunnigan, A. Dunford, and J. Bringardner, "From Cornerstone to Capstone: Students' Design Thinking and Problem Solving," presented at the 2020 ASEE Virtual Annual Conference Content Access, Jun. 2020. Accessed: Jan. 27, 2023. [Online]. Available:

https://peer.asee.org/from-cornerstone-to-capstone-students-design-thinking-and-problem-solving

[3] J. A. Bolognese, R. Whalen, E. D. Cordell, A. P. L. Cilfone, and B. D. Williams, "A First Year Engineering Information Literacy Workshop: Redesigned for Remote Delivery," presented at the 2021 ASEE Virtual Annual Conference Content Access, Jul. 2021. Accessed: Jan. 06, 2023. [Online]. Available: https://peer.asee.org/a-first-year-engineering-information-literacy-workshop-redesigned-for-remote-delivery

[4] B. K. Jaeger, R. Whalen, and S. F. Freeman, "Programming is Invisible – or is it? How to Bring a First-year Programming Course to Life," presented at the 2012 ASEE Annual Conference & Exposition, Jun. 2012, p. 25.1079.1-25.1079.23. Accessed: Jan. 06, 2023. [Online]. Available: https://peer.asee.org/programming-is-invisible-or-is-it-how-to-bring-a-first-year-programming-course-to-life

[5] M. Jalal and H. Anis, "The Impact of Students' Grit & Project Ownership on Students' Learning Outcomes in Maker-based Cornerstone Engineering Design Courses," presented at the 2022 ASEE Annual Conference & Exposition, Aug. 2022. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/the-impact-of-students-grit-project-ownership-on-students-learning-outcomes-in-maker-based-cornerstone-engineering-design-courses

[6] S. C. Ritter and S. G. Bilén, "EDSGN 100: A first-year cornerstone engineering design course," presented at the 2019 FYEE Conference, Jul. 2019. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/edsgn-100-a-first-year-cornerstone-engineering-design-course

[7] E. Kames, B. Morkos, and D. Shah, "A Longitudinal Study Exploring Motivation Factors in Cornerstone and Capstone Design Courses," presented at the 2018 ASEE Annual Conference & Exposition, Jun. 2018. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/a-longitudinal-study-exploring-motivation-factors-in-cornerstone-and-capstone-design-courses

[8] S. John, C. Hanson, J. Lenn, M. Jansons, and J. Potoff, "Implementing Embedded Control into Projects Designed by Students With Little or No Programming Experience," presented at the 2020 First-Year Engineering Experience, Jul. 2020. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/implementing-embedded-control-into-projects-designed-by-students-with-little-or-no-programming-experience

[9] J. R. Haughery and D. R. Raman, "A Systematic Review of Mechatronic-based Projects in Introductory Engineering and Technology Courses," presented at the 2015 ASEE Annual Conference & Exposition, Jun. 2015, p. 26.119.1-26.119.9. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/a-systematic-review-of-mechatronic-based-projects-in-introductory-engineering-and-technology-courses

[10] M. Galaleldin and H. Anis, "The Impact of Integrating Making Activities to Cornerstone Design Courses on Students' Implicit Theories of Making Ability," presented at the 2019 ASEE Annual Conference & Exposition, Jun. 2019. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/the-impact-of-integrating-making-activities-to-cornerstone-design-courses-on-students-implicit-theories-of-making-ability

[11] M. Galaleldin, H. Anis, and P. Dumond, "Board 32: The Impact of Integrating Making Activities Into Cornerstone Design Courses," presented at the 2019 ASEE Annual Conference & Exposition, Jun. 2019. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/board-32-the-impact-of-integrating-making-activities-into-cornerstone-design-courses

[12] W. Zhan, J. Wang, and M. Vanajakumari, “High impact activities to improve student learning,” presented at the 2013 ASEE Annual Conference & Exposition, Jun. 2013, p. 23.661.1-23.661.14. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/high-impact-activities-to-improve-student-learning

[13] K. Calabro, “Work in Progress: A Case Study Exploring Teaching Strategies Employed in a Cornerstone Engineering Design Course,” presented at the 2018 ASEE Annual Conference & Exposition, Jun. 2018. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/work-in-progress-a-case-study-exploring-teaching-strategies-employed-in-a-cornerstone-engineering-design-course

[14] T. Shepard, J. Choi, T. D. Holmes, and B. W. Carlin, “The Effect of Project Constraints and Choice on First-year Microcontroller Projects,” presented at the 2015 ASEE Annual Conference & Exposition, Jun. 2015, p. 26.1522.1-26.1522.12. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/the-effect-of-project-constraints-and-choice-on-first-year-microcontroller-projects

[15] P. Wong and B. Pejcinovic, “Teaching MATLAB and C Programming in First-year Electrical Engineering Courses Using a Data Acquisition Device,” presented at the 2015 ASEE Annual Conference & Exposition, Jun. 2015, p. 26.1480.1-26.1480.11. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/teaching-matlab-and-c-programming-in-first-year-electrical-engineering-courses-using-a-data-acquisition-device

[16] D. J. Frank, K. J. Witt, C. Hartle, J. J. Enders, V. Beiring, and R. J. Freuler, “A Low-Cost Robot Positioning System for a First-Year Engineering Cornerstone Design Project,” presented at the 2016 ASEE Annual Conference & Exposition, Jun. 2016. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/a-low-cost-robot-positioning-system-for-a-first-year-engineering-cornerstone-design-project

[17] R. H. Brooks, “First-Year Engineering Program Curriculum ReDesign,” presented at the ASEE 2021 Gulf-Southwest Annual Conference, Mar. 2021. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/first-year-engineering-program-curriculum-redesign

[18] D. J. Frank *et al.*, “Developing and Improving a Multi-Element First-Year Engineering Cornerstone Autonomous Robotics Design Project,” presented at the 2017 ASEE Annual Conference & Exposition, Jun. 2017. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/developing-and-improving-a-multi-element-first-year-engineering-cornerstone-autonomous-robotics-design-project

[19] J. N. Thomas, C. Theriault, C. Duba, L. P. van Ginneken, N. J. Rivera, and B. M. Tugade, “A Project-based Computer Engineering Curriculum,” presented at the 2015 ASEE Annual Conference & Exposition, Jun. 2015, p. 26.90.1-26.90.20. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/a-project-based-computer-engineering-curriculum

[20] M. V. Deshpande, P. K. Waychal, and P. P. Udawant, “Analysis of Improved Pedagogy Applied for Teaching courses related to Computer Programming for First Year Engineering Programs,” presented at the 2015 ASEE International Forum, Jun. 2015, p. 19.3.1-19.3.5. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/analysis-of-improved-pedagogy-applied-for-teaching-courses-related-to-computer-programming-for-first-year-engineering-programs

[21] J. K. Lumpp *et al.*, “Instructional Use of Computers in a Hands-on Programming Course for First-Year Engineering Students,” presented at the 2019 ASEE Annual Conference & Exposition, Jun. 2019. Accessed: Jan. 31, 2023. [Online]. Available:

https://peer.asee.org/instructional-use-of-computers-in-a-hands-on-programming-course-for-first-year-engineering-students

[22] B. M. Lunt and C. R. Helps, "Freshman Cornerstone Course Meets Multiple Educational Goals," presented at the 1998 Annual Conference, Jun. 1998, p. 3.292.1-3.292.6. Accessed: Jan. 27, 2023. [Online]. Available: https://peer.asee.org/freshman-cornerstone-course-meets-multiple-educational-goals

[23] J. E. Lewis, B. S. Robinson, and N. Hawkins, "First-Year Engineering Student Perceptions in Programming Self-Efficacy and the Effectiveness of Associated Pedagogy Delivered via an Introductory, Two-Course Sequence in Engineering," presented at the 2020 ASEE Virtual Annual Conference Content Access, Jun. 2020. Accessed: Jan. 31, 2023. [Online]. Available: https://peer.asee.org/first-year-engineering-student-perceptions-in-programming-self-efficacy-and-the-effectiveness-of-associated-pedagogy-delivered-via-an-introductory-two-course-sequence-in-engineering

Appendix A: Pre-Survey

1. What is your name?
2. What are your pronouns?
3. What was your first year of full-time college study?
4. Prior to this semester, how much experience did you have with writing computer programs?
    - I had never written a computer program.
    - I had very limited prior experience writing computer programs
    - I had written multiple computer programs, but I needed a little help getting started again
    - I had written multiple computer programs, and was able to write short programs again very quickly
5. If you had programming experience prior to this class, where did you first learn to program?
    - AP course in high school
    - Course in school before high school
    - Extra-curricular club or program
    - Non-AP course in high school
    - Not applicable / I had no prior experience
    - Previous college course
    - Self-taught
6. Prior to this semester, how much experience did you have with making electronics (basic circuitry)?
    - I had never worked with making electronics
    - I had very limited prior experience with making electronics
    - I had worked with making electronics multiple times, but I needed a little help getting started again
    - I had worked with making electronics multiple times, and was able to get started again very quickly
7. Thinking back to just before we started this semester, how confident would you have been to start working on a graded project that involves computer programming?

    - Not at all Confident
    - Slightly Confident
    - Moderately Confident
    - Very Confident
    - Extremely Confident
8. Thinking back to just before we started this semester, how confident would you have been to start working on a graded project that involves making electronics (basic circuitry)?
    - Not at all Confident
    - Slightly Confident
    - Moderately Confident
    - Very Confident
    - Extremely Confident
9. Thinking back to just before we started this semester, how confident would you have been to start working on a graded project that involves an unclear path to the solution?
    - Not at all Confident
    - Slightly Confident
    - Moderately Confident
    - Very Confident
    - Extremely Confident

Appendix B: Post-Survey

1. What is your name?
2. What are your pronouns?
3. What was your first year of full-time college study?
4. How confident would you be now to start working on a graded project that involves computer programming?
    - Not at all Confident
    - Slightly Confident
    - Moderately Confident
    - Very Confident
    - Extremely Confident
5. How confident would you be now to start working on a graded project that involves making electronics (basic circuitry)?
    - Not at all Confident
    - Slightly Confident
    - Moderately Confident
    - Very Confident
    - Extremely Confident
6. How confident would you be now to start working on a graded project that involves an unclear path to the solution?
    - Not at all Confident
    - Slightly Confident
    - Moderately Confident
    - Very Confident
    - Extremely Confident

7. After being introduced to both a programming language (C++) and a hardware application of programming (Arduino), please chose the statement you most agree with.
    - o Starting with programming or hardware first would not make a difference in a student's learning
    - o Learning all the programming before any hardware introduction would help students learn more
    - o Introducing the hardware basics after some introduction to programing would help students learn more
    - o Starting with the hardware basics would help students learn more
8. After being introduced to both a programming language (C++) and a hardware application of programming (Arduino) please chose the statement you most agree with in regard to frustrations experienced.
    - o I didn’t feel any particular frustration about learning the hardware or programming
    - o Starting with programming or hardware first would not have made a difference in my level of frustration
    - o Learning all the programming before any hardware introduction would have helped me be less frustrated
    - o Introducing the hardware basics after some introduction to programing would have helped me be less frustrated
    - o Starting with the hardware basics would have helped me be less frustrated