Advances in Engineering Education

FALL 2019

Description and Mixed Methods Evaluation of a Novel Hardware-Based Introductory Programming Course

STEPHEN SECULES
Florida International University
Miami, FL

AND

WESLEY LAWSON
University of Maryland
College Park, MD

ABSTRACT

This paper compares an innovative approach to teaching a required introductory C programming course to a traditional C programming course for electrical engineering (EE) students. The novel course utilizes hardware-based projects to motivate students to master language syntax and implement key programming concepts and best practices. In a mixed methods research evaluation, we compare the attitudes and self-perceptions of the students in each of the introductory courses, as well as success rates for each cohort in their subsequent intermediate programming class and progress toward their degrees. Performance and retention outcomes were similar for the two courses. After students took the novel course, they were more likely to feel that they fit in as electrical engineers and less likely to believe that programming was "not real engineering." Qualitative data and pedagogical descriptions of the two courses help connect these quantitative findings to key features of the courses.

Key words: electrical engineering, active learning, project based learning, mixed methods research, course evaluation

INTRODUCTION

Introductory engineering courses are often a subject of concern and site of pedagogical intervention for universities. Large enrollment and faculty demands mean these classes are often structured as large lectures with homework review sessions led by teaching assistants. These introductory lecture courses can be challenging, impersonal, and disconnected from the discipline. They are also a source of stress and attrition for students, particularly those from under-represented groups [1].

Within introductory engineering course sequences, programming courses have been noted to be particularly difficult, with a high potential for impact on student affect, identity, and persistence as an engineering major [2].

A growing reform movement has promoted active learning pedagogies as an alternative to the lecture format. These courses are structured around opportunities for students to construct knowledge, individually and collaboratively, and they promote learning through engaging in authentic tasks [3]. In the world of introductory engineering design courses, active learning courses have been proven to improve learning outcomes, increase interest, and improve retention in the major [4]–[9]. In the shift from passive learning environments to active-learning and project-driven freshman design courses, new technologies have been introduced to enhance students' knowledge, skills, and abilities. For example, universities have incorporated computer-aided design, LEGO Mindstorms [10], and visual / graphical programming languages [7], [9] to enhance their instruction of design.

For over a decade, Arduino electronics platforms have been available for such courses and offer considerable flexibility and programming power [7]–[9]. In addition, the Arduino is inexpensive and has a large following in the hobbyist community. As such, there is a lot of Arduino-compatible hardware available and a lot of online documentation that can be used to augment the learning process. Although Arduinos can be an enjoyable introduction to a version of C programming language in a course primarily focused on engineering design, there are multiple reasons why instruction on an Arduino may not serve as an adequate introduction to ANSI C programming in a course whose main learning objectives involve programming knowledge and efficacy for all students in the course:

- First, Arduino C has some departures from standard ANSI C, which can make portability an issue. Arduino C is a subset of C++, and as such freely mixes C and C++ concepts, like using a string class as if it were an intrinsic data type [29]. Also, in place of the ANSI C main{} function, Arduino C requires setup and loop functions, which are useful for indefinite hardware monitoring, but are clumsy for many programs with well-defined start and end points. Input and output handling is typically quite different on an Arduino, where many devices have limited memory and use a serial monitor (serial.print) instead of the printf function, so that Arduino C users will have no experience with file input/output (I/O) and related concepts (like FILE pointers) when they move to another platform. Once a beginning student has sufficient experience with either ANSI C or Arduino C, it is not difficult to learn the differences and transition from one to the other, but we feel it is most effective to teach ANSI C on a computer that has an ANSI C compiler.
- Second, as programming instruction nested within a course on design, the Arduino portion is often an instructional module lasting only a few weeks alongside other design activities [9]. This is not enough time to cover in sufficient depth all aspects of C programming. As an example, at the University of Maryland, College Park, the freshman engineering design course

spends 2½ weeks on "Electronic Analysis and Arduino programming," so less than 15% of the semester is spent on Arduino instruction for all students in the course. There is a general introduction that presents basic syntax, program flow and a handful of C keywords, but most of the instruction focuses on standard library functions to control and monitor hardware connected to the I/O pins. Arrays, file input/output, advanced program selection, data types and data structure concepts are glossed over, and best programming practices, code reliability and maintainability, and other important programming principles are typically not discussed at all.

- Finally, in an introductory design course there is typically insufficient assessment to guarantee that all students master the required programming learning objectives, often because the multi-dimensional team-based projects can involve a distribution of roles in which only a few take on the role of programming on a team.

By comparison, courses focusing directly on introducing students to programming have tended to remain traditional in format. The language taught over the past 50 years has changed from FORTRAN to C, C++, Java, Python, etc. and the days of punched cards and batch jobs are long-gone, but the course content still tends to emphasize syntax and software applications, which may create a disconnect with engineering students' senses of the engineering discipline. Likewise, the lecture format assumes that students can passively consume large amounts of information about programming and complete their homework assignments individually at home or seek out extra help at review sessions and office hours.

Active learning classrooms instead make the assumption that students will need to apply and co-create knowledge in order to understand and retain it. Prominent attempts to integrate active learning pedagogy into programming instruction have appeared in the computer science education literature [11]–[14]. Sometimes these courses incorporate active learning elements in a relatively minor supplemental laboratory component for students to seek help they complete their software-based (e.g., a computer game) programming problems. The "traditional" programming class that we describe in our paper is such a course. Other iterations of active learning in introductory program have emphasized lab activities with extended time for group collaboration on ill-structured problems [12], [15]. In general, while important pedagogical advances have emerged from computer science education community, they also differ from engineering educational contexts which benefit from the authenticity of integrating electrical engineering hardware rather than purely software projects.

Finally, the paradigm of hardware-based and project-based or problem-based learning (PBL) has been incorporated in introductory courses. A number of these PBL efforts utilized the Arduino platforms [16], [17], including one on hardware and the Internet of Things. Some of these efforts have featured the Raspberry Pi as the vehicle for PBL instruction [18], [19]. A key difference between the Arduino [20] and the Raspberry Pi [21] and similar single-board computers like the BeagleBone [22], is that they are stand-alone computers on which you can have ANSI standard versions of C,

C++, Python, etc. These and several other microcomputers, including the Handy Board [23], Arduino, BeagleBone and Raspberry Pi, have been demonstrated as valuable platforms for PBL programming instruction [19], [24].

This paper describes an NSF-funded novel introductory C programming course incorporating active learning, hardware, and project-driven pedagogy in small lab sections. Students use a Raspberry Pi microcomputer to complete authentic engineering challenges incorporating software programming and hardware circuitry design. The paper reports on the 4-semester pedagogical intervention and the outcomes of a mixed methods research evaluation effort. Qualitative interviews and ethnographic observations were conducted with students in traditional and novel courses. Pre-, post-, and subsequent semester surveys were implemented for each cohort of students who took the course. We build on two prior papers on this topic discussed survey results [25], [26], to present a comprehensive description and evaluation of the course design effort. Our project is also unique in that we compare using the Raspberry Pi as the vehicle to teach an introductory C programming course with a traditional introductory C programming course. Student outcomes assessed include effectiveness to achieve the programming learning outcomes and self-perceptions and attitudes towards both programming and electrical engineering.

This paper takes the following approach to presenting the pedagogical intervention: First, it combines instructor perspectives and qualitative data to characterize aspects of course structures, lecture content and approach, and lab activities as differences between the traditional and the novel course. Then it discusses quantitative measures of performance and affective outcomes between the two courses. Finally, it concludes with a description of key course design challenges and the instructional team's iterative process to resolve them.

COURSE COMPARISON FOR LEARNING AND ACHIEVEMENT

This pedagogical intervention took place within the Clark School of Engineering at the University of Maryland. We offer two versions of our introductory C programming course for electrical engineers. The first is a traditional course where students are given individual software-only programming assignments that do not involve any hardware besides the computer itself and its peripheral devices. In a second novel course offering, students attend a lab that involves mostly partner-based, programming assignments emphasizing computer-controlled, hardware-driven projects and final multi-week, group and individual projects. The Raspberry Pi (RPi) microcomputer is used for these hardware-based assignments. RPi kits for students and all necessary hardware were supplied from the NSF grant for all four novel course offerings.

The traditional course is two credit hours and enrolls at least 90 students per year in two or three sections. The once-weekly lecture lasts 75 minutes and has a maximum enrollment of approximately 60 students. There are bi-weekly, 50-minute discussion sessions which are run by teaching assistants and limited to 12 students.

The novel course is three credits to allow time for shorter (50 minute) lectures twice a week and longer (3-hour) once-weekly lab sections for active learning hardware applications. The novel course was limited to at most 30 students per semester, with at most 10 students in each lab section. The total course enrollment was 77 students over four semesters.

For comparison purposes, a common textbook is used for both courses (*The C Programming Language, 2nd edition* by Brian Kernighan and Dennis Ritchie), but a complete set of online lecture notes are more commonly referenced by students in the novel course. Those notes address both software (syntax, best practices, sample codes, etc.) and hardware (Raspberry Pi's, sensors, circuit theory, etc.). Since all students take the subsequent semester intermediate course, there is a common content expectation for completion of the introductory semester course. The two courses are similar in relative weightings for course grade assessment. The intermediate course is structured similarly to the traditional course in terms of lecture and discussion format.

In summary, the two introductory courses are similar in structure and content. The key differences for the two introductory courses then are (1) the collaborative and active learning format for novel course labs and final project and (2) the introduction and use of electronic hardware. The following two sections assess the novel course structure in terms of fostering learning and student success in the subsequent programming course.

Assessing Contribution to Learning

The novel course structure was broadly successful in fostering learning. Students in the novel course structure were asked in a survey to "Rate the following course components on how useful they were to your learning: lectures, discussion section / lab, homework assignments, and projects." The results, shown in Figure 1, show that the novel course lab was consistently perceived as a valuable course component for student learning, whereas most other course components received more mixed reviews across both courses.

Novel course students were prompted to explain this answer further, and their free responses provide additional clarity into the meaning of these results:

- *I thought that the programming in lab was really good. The only thing I would have wanted more of is in-class programming. I know there is not much time but merely going over different programs in lecture was not enough and I would have liked some in-class programming.*

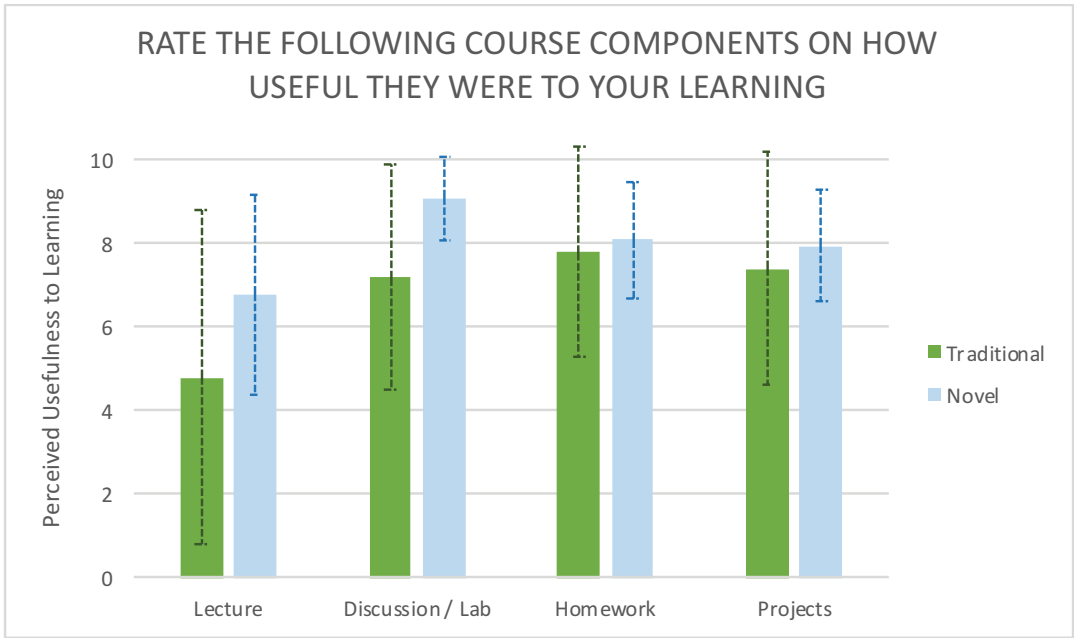

Figure 1. Average student ratings for how useful course components were to their learning (1- not at all useful, 10- extremely useful), compared between traditional and novel course. Error bars are standard deviation of rating scores.

- *The lectures were definitely helpful, but an extreme majority of my learning was a byproduct of actually writing code either in lab or on homework.*
- *I found the lectures difficult to listen to but learned a great deal from actively coding in lab.*
- *Labs and homework assignments were much more helpful than lectures, though the slides were beneficial.*

These perspectives are consistent with student impressions that lectures were valuable resources but the bulk of their learning took place through their active learning work on lab assignments. Since traditional lecture students were not experiencing an active learning setting, their responses represent a partial control—in the absence of an active learning lab experience each traditional course component received middling reviews with a wide standard deviation for how useful they were perceived to learning.

Assessing Student Success Rates

The novel course was taught four times in the past three years. Student success comparisons were made between the students who took the novel course and the students who took the traditional

course in the same time frame. The students who took the traditional class and the novel class were not randomly assigned, but rather self-selected for each course, and unknown selection effects could have occurred if different types of students enrolled in the novel versus the traditional course. We attempted to account for this bias by noting the cumulative grade point averages of students in the two classes, as a measure of overall academic success. This comparison revealed similarities between the two student populations, in that their average cumulative grade point averages to date were both 3.3. However, students who took the novel introductory programming course did somewhat *better* in the intermediate traditional class, in spite of greater similarity in course structures and pedagogy between the two traditional courses versus the novel course. Those students passed the intermediate class with an average GPA of 3.2 ± 1.0 whereas students in the traditional introductory course passed the intermediate course with an average GPA of 3.1 ± 1.0. Although this is not a statistically significant improvement, it indicates a positive or neutral overall trend.

A total of one student failed the novel course over the four semesters and no students withdrew from the course (after the initial schedule adjustment period), so 98% of the students passed the novel course. By contrast, 95% of the students who enrolled in the traditional introductory course completed the course during the same time frame. Most of the students who did not complete the course withdrew from it. Given that taking either the traditional or novel course is a prerequisite for a requirement of the EE major, these withdrawals likely represent students leaving the department, though they could have retaken the course elsewhere or after this study was completed.

The traditional course was restricted to EE majors, though some exceptions were made for students who were not yet enrolled in the limited-enrollment major. The novel course, in comparison, welcomed students who were not yet EE majors, but who wanted to enter the major. Most of the students eventually entered the EE program but a handful did not. Of the students who were already in EE or who eventually were admitted to EE, over 94% are still EE majors and the remainder have transferred to other engineering disciplines. Of the students who took the traditional course during the same time frame, approximately 87% are still in EE. 8% have moved to other engineering disciplines and the remaining 5% students are now computer engineering majors. Thus, for this time period, retention of the novel course students is several percent higher than for the traditional course, and the novel students are doing just as well as the other cohort both in the intermediate programming class and in their discipline.

Having assessed the course structures and overall success, the subsequent sections will use mixed methods empirical findings to 1) characterize the lecture and lab class differences between the two courses, and 2) evaluate them with respect to gains for student affect and identity.

CHARACTERIZING COURSE DIFFERENCES

As noted, both courses incorporated an instructor-of-record-led lecture component which introduced the content required to complete homework, labs, projects, quizzes, midterms, and final exams. The lecture content shared several similarities since the two parallel courses were required to cover the same basic material in an electrical engineering course sequence. The lectures differed somewhat in presentation and character. This section presents similarities and differences in lecture between the two courses.

Course Learning Goals

The student learning objectives on the official syllabi of the two introductory courses are compared in Table I. The first novel course learning objective is basically the same as the first two learning objectives of the traditional course. While the novel course does not specifically mention sorting and searching, as a minimum, a bubble sort and a numerical key binary search are introduced in the class. The third and fourth objectives for both classes are basically the same. The key difference of the third objective is that the novel course teaches students to work in an integrated development environment (IDE) whereas the traditional course teaches the UNIX environment for code development and execution. The traditional course also emphasizes good programming practices and maintainable code even though these are not stated in the list of objectives. The novel course focuses more on solving engineering problems, but developing and implementing algorithms is

Table I. A comparison of the stated learning objectives for the two introductory courses.

Novel course learning objectives	Traditional course learning objectives
1. Operational familiarity with elementary programming concepts: program flow, data types, arrays and memory, logic and arithmetic operations, and functions	1. Elementary programming concepts (e.g. program selection, repetition, and functions)
2. Appreciation for the enabling role of programmable devices in technological systems and applications	2. Fundamental concepts in data structure (e.g. data type, array, string, search, and sort)
3. Ability to use an IDE to write, debug, load and run code to solve engineering problems and to perform basic calculations, input and output	3. Ability to use UNIX as the operating system for text editing, file management, and programming
4. Ability to utilize good programming practices to write efficient, clear, and maintainable code	4. Ability to write a code to implement algorithms or solve problems
5. Understanding of the operation of basic electronic components, sensors and actuators	5. Ability to analyze a given code, debug it, and predict its output
6. Ability to work effectively in teams	
7. Ability to communicate effectively in written / oral formats.	

generally part of that process. The novel course does not have a stated objective comparable to the final traditional course objective (analyze and debug code), but debugging techniques are taught and analyzing, debugging and predicting output for codes has always been a part of the assessment process on midterm and final exams. The novel course objectives 2, 5, 6, and 7 have no corresponding objectives for the traditional course.

Lecture Content Topics

The novel course lectures are divided into 18 modules of varying lengths (Table II). The nine C programming modules are presented in similar order and cover similar material as the traditional course. The first module touches on most features of the language in a relatively superficial way and the remaining 8 modules explore each topic in greater depth. Since Unix/Linux operating system is a key component of the traditional course and the way homework assignments are submitted in the traditional introductory and intermediate courses, Module 10 is a specific introduction to Unix for the novel course. The eight hardware modules are typically much shorter than the programming modules and are inserted into the lectures as needed for the students to be successful in the laboratory.

Although the two courses held similar programming learning outcomes, it was apparent to several students interviewed that the novel course required additional material to learn compared to the traditional course. This appeared to affect how students self-selected for either course based on reputation. Although the novel course incorporated active learning pedagogies aimed at engaging students and providing an authentic engineering experience for unfamiliar and underrepresented groups, the reputation (and sometimes reality) of the novel course meant that it moved faster to cover more material, and the traditional course sometimes became a safer choice for such groups. Although these student experiential and reputational phenomena appeared pivotal in the self-selection process and student population makeup, they also shifted year-to-year idiosyncratically.

Table II. The C programming course module titles.

1	A crash course in C programming	10	Introduction to Unix
2	Data types	11	The Raspberry Pi and the GPIO
3	Operators	12	Introduction to basic circuit components
4	Program selection	13	Introduction to sensors
5	Repetition	14	Introduction to op-amps, diodes and transistors
6	Functions	15	The SPI interface
7	Arrays	16	Introduction to A/D converters
8	Input / output formatting	17	The I^2C interface
9	File input / output	18	Introduction to mux/demux circuits

Table III. Sample individual homework assignment from novel course.

Write a code that inputs a component value which could be a capacitor, inductor, or resistor. First the numerical value is entered, then a modifier MAY be entered: k for 1000, M for 1,000,000, m for 0.001 and u for 0.000001, Finally, an F is entered for a capacitor, an H is entered for an inductor, and an O is entered for a resistor. Write a code that will read in component values until the end of the file. Output the component type and value. If any incorrect data is entered, an error message must be printed. For example:

Input:	50 kO	Output:	50000 Ohm Resistor
	mF		Error: enter a numeric value first
	50 sO		Error: improper modifier
	4.7 uF		0.0000047 Farad Capacitor

Homework Assignments

Similar software-based homework assignments are given in both the traditional and novel courses. The individual homework assignments are assigned during lecture and done outside of lecture and lab time. For both courses, the software-only codes are typically written individually in computer labs or on the students' computers and up-loaded to the course website, where they are tested and graded by the class instructors. Typical assignments for the novel course are related to concepts and computations needed in sophomore and junior-level ECE courses. An example of a software-only homework assignment is given in Table III. Assignments for the traditional class vary by instructor, typically with some assignments that are related to electrical engineering concepts or terminology and several others that are not (e.g., calculate trajectories of projectiles fired from a cannon, create a Black Jack or Yahtzee game).

Instructional Style and Classroom Discourse Observations

One professor taught the novel course and a different professor taught the traditional course. The C programming lecture content was similar between the two courses and both lectures featured mostly passive learning. But they often differed in examples given, pacing, presentation style, and format preferences between the two professors. The traditional course presentation was fairly systematic and linear. The professor would often lecture by writing out lines of code on a whiteboard, discussing each line, its function, and why it needed to be written. The examples were not generally driven by hardware considerations. The professor's discourse style was not particularly conversational—if asking "any questions?" there would typically not be a long pause or a verbal or visual cue that he was seeking student questions. Often each line of code (or definition, or diagram) would be narrated with a paraphrased explanation and a slight uptick at the end of the statement—an indication of a desire to check in with students and confirm whether they were following. Nevertheless, student questions were rare and often kept to smaller clarifying questions, for example "Can you explain what that term means?" The larger class and a stadium-style lecture

Figure 2. Traditional course layout with whiteboard examples and overhead projector showing completed code, and several rows of students.

seating (see Fig. 2) probably contributed to the physical and interactional distance between the professor and students. With lower expectations for participation and greater anonymity amongst students, some students disengaged.

The novel class took a somewhat less linear approach to covering the same programming content plus the content specific to embedded microprocessors and circuitry. In order to get all students up to speed on basic programming content before they entered the lab, the first two weeks of the course were a fast-paced "crash course in C programming language." In subsequent weeks, lecture content moved between standard programming content, electrical engineering content, next-lab-task specific advice, and responding to student queries. The professor had PowerPoint slides prepared which he walked through, but genuine student questions, comments, and conversation frequently added to the discourse. At times curious/advanced questions threatened to take too much class time of the 50-minute period and the professor had to find strategies to limit these questions (see Section V.C). In general, the smaller class size and lesser physical distance between professor and students (see Fig. 3) supported a more conversational interaction, and prevented more obvious disengagement. Most students tended to take notes, watch the lecture attentively, or engage in question asking and answering. Students were active participants in their learning, although not to the same extent as they were in the lab (see Section IV.C).

In general, the student discourse distinguished the two course experiences: whereas in the traditional course, the most common questions were clarification questions about something the professor had just said or written, in the novel course, the most common questions were points of curiosity from students who already understood the material. More students tended to participate in the novel course, and there was a more conversational interaction than in the traditional course.

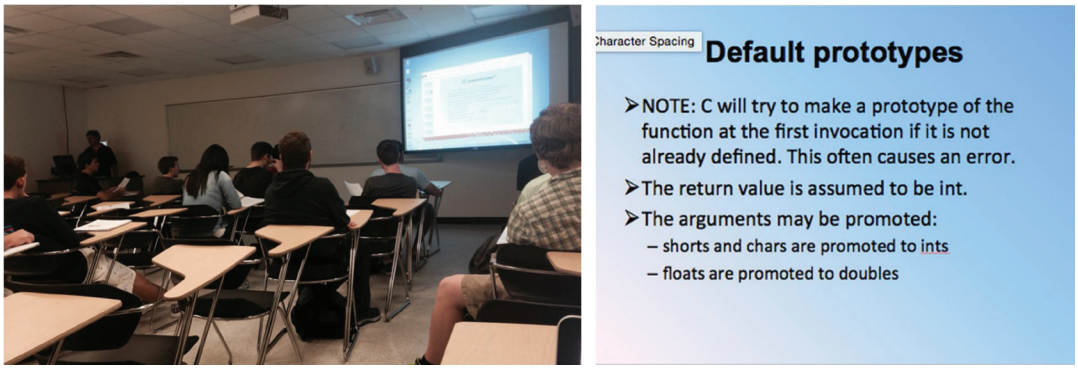

Figure 3. Novel class showing a projected PowerPoint with students in relatively close proximity. An example PowerPoint slide explains prototypes.

We have characterized the lecture content and approach in order to account for the broader contexts in which students experienced the novel course pedagogical intervention in lab. Certain practical constraints and instructional styles contributed to the differences we have noted, rather than solely intentional and strategic pedagogical efforts connected to the novel pedagogical intervention.

NOVEL LAB-BASED PEDAGOGICAL INTERVENTION

The lab-based portion of the novel course represents the key pedagogical intervention, and the most significant learning experience for students who took the course. Although the discussion section for the traditional course was similar in allowing access to an undergraduate teaching fellow (UTF) in a smaller setting aimed at homework help, it was not as consistently recognized as valuable for learning by students. The scope of the empirical research did not allow for a side-by-side comparison of discussion sections, however based on ethnographic classroom observations the novel course lab appeared to differ from the traditional course discussion section in the following ways: 1) the extent to which the novel course lab comprised active engagement on programming tasks, versus the traditional discussion section which answered focused questions on areas of trouble for homework assignments largely completed outside, and 2) the incorporating of hardware-based electrical engineering tasks in the novel lab, versus solely software computer programming tasks in the traditional course, and 3) the incorporating of collaborative (pair- and team-based) assignments in the novel lab curriculum, versus solely individual assignments in the traditional course.

The following sections draw on empirical research to characterize the nature of the novel lab programming tasks and show students completing them.

Lab Technology and Resources

The Raspberry Pi microcomputer was the device the students used in the novel course lab for all semesters, but each offering used a new version of the device, culminating with the RPi 3 Model B. Each new model required additional preparation work to teach the labs, but also offered significant hardware upgrades to improve the experience in the lab. Enhancements included faster processing, more USB ports, more general purpose input-output (GPIO) pins, more memory, and finally on-board WiFi. Raspberry Pi's have the necessary capability to design a number of meaningful programming laboratories to give students a glimpse of the meaning of many EE sub-disciplines. In addition to the digital input/output pins, the RPi can communicate via SPI, I2C, and UART interfaces. Multiple pins can easily be configured for pulse-code modulation. While there are no analog inputs, we used the SPI interface to get analog data from an MCP3008, an 8-channel, 10 bit A/D converter.

Each lab was led by a UTF with C programming experience, often having taken the course in a prior semester. The laboratory room was equipped with a station for each student that included a Raspberry Pi and needed hardware (keyboard, mouse, HDMI/DVI monitor) (Figure 4). It also contained

Figure 4. Students working in lab with circuitry, RPi, and desktop computers.

all of the components, hardware, and test and measurement equipment needed to perform the labs. The hardware included resistors, LEDs, photoresistors, acoustic and infrared distance sensors, temperature and magnetic field sensors, gyros and accelerometers (or all together in an inertial measurement unit, or IMU), servos, operational amplifiers, A/D converters, multiplexors, and other miscellaneous components. There are also breadboards, multimeters, soldering irons, rechargeable batteries, and various wires and connectors.

Students were also issued an RPi kit that included the device, case, charger, USB Wi-Fi adapter (before the RPi 3B), cables and electronic components, and an SD card. The SD cards were preloaded with an image that included all of the software needed to succeed in the course. Software included a standard GUI, sample codes that showed how to use many of the GPIO features, the GEANY IDE and the necessary programs and libraries to program in C and control the GPIO, along with documents that discussed a number of helpful hints, including how to automatically execute a code when the RPi boots, and wiring diagrams for a number of key circuits. Students were also provided with tutorials on how to connect their RPis to their laptops. Many students chose this route even in the lab as it provided a very convenient, portable interface to the RPi. Unlike an Arduino, the laptop functions only as an interface, allowing the RPi access to the screen and keyboard and facilitating data transfer between the RPi and an OS/Windows/Chrome machine.

Lab Session Content

Each semester there were nine labs in the course and one group project. For the first course offering, the majority of the labs were individual projects. For the remaining three offerings, based on course feedback, about one-third of the labs were designated individual labs and the other two-thirds were designed to be done in groups of two. Lab content only varied slightly from one semester to the next, based on student feedback. While some of the labs could be finished in three hours, many were to be completed outside of regular lab time, and students carried their SD cards to and from lab for continuity. Lab 9 was optional, but many students completed it. A summary of the lab goals is given in Table IV.

Starting with the third offering, a final individual project was added to the course, in addition to the group project described below. Whereas in Lab 2 students had to write a code that output a sentence in Morse code, for their final individual project they had to detect Morse code from a circuit built by the instructor, and translate that code into English. The instructor's code would randomly select from a list of sample sentences and output in Morse code those sentences on an LED. In conjunction with this new project, the duration and weight of the final paper exam was decreased. It was felt that the new format would allow a more accurate assessment of students' programming skills.

Table IV. The principle goals of the laboratories for the C programming class in example term.

Lab	Content/Goals	Group?
1	Assemble RPi Kit and write simple code to output message	no
2	Generate a code that allows you to type in a sentence and then have an LED blink the sentence in Morse code	yes
3	Generate a code that will turns lights (LEDs) on when lights are off and keep track of where (say, in a house) the lights are on	no
4	Learn to use the MCP3008 A/D converter. Write codes to (a) get data from analog temperature sensors, (b) calibrate an IR distance sensor, and (3) use a calibrated IR distance sensor to measure distances to objects.	yes
5	Generate two codes for a 3-axis analog accelerometer. The first code is used to calibrate the sensor. The second code is to measure and record accelerometer data with a calibrated sensor and attempt to discern velocity and distance.	yes
6	Generate two codes for a 3-axis digital magnetic sensor. The first code is used to null and calibrate the sensor. The second code is to measure and record magnetic field data with a calibrated sensor.	no
7	Generate a code that interprets the data from an acoustic distance sensor to estimate distance to objects and to identify and ignore outliers in the data.	yes
8	Generate a code that utilizes a servo motor and a magnetic sensor to track a moving permanent magnet	yes
9	Write a code to use two digital magnetic sensors and a mux/demux chip to make a magnetic gradiometer. Write a code to calibrate this device.	yes

The final group project has been to use sensors to follow an obstacle course from start to finish and then turn off the sensors and return to start. The first few semesters we used an RC (remote control) tank as the platform for the project, as shown in Figure 5 (left). The tank was hacked and a custom integrated circuit was designed and used to control the tank motors and to make it

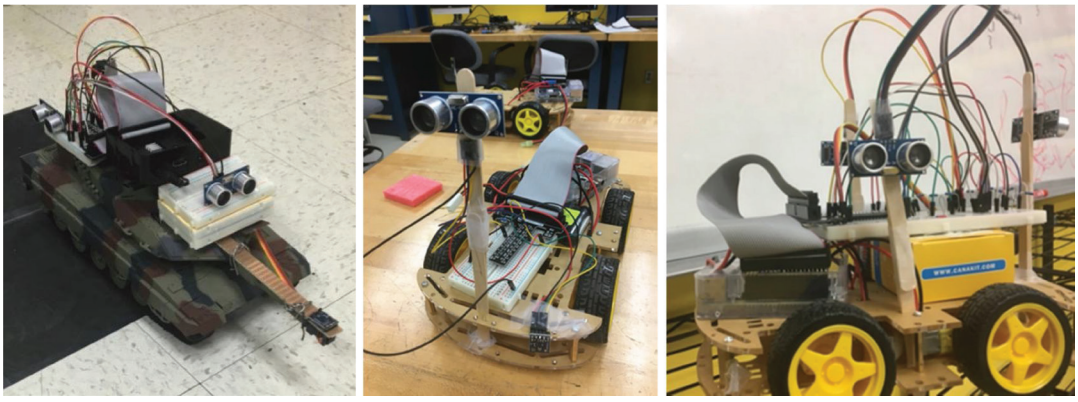

Figure 5. Original vehicle for the final project first three course offerings, hacked RC tank (left), Latest vehicle for the final project first three course offerings, off-the-shelf vehicle and motor shield (middle, right).

impossible to accidently short and damage those motors. The final semester an off-the-shelf vehicle was used as shown in Figure 5 (middle and right), after being modified by the students) We also bought an off-the-shelf motor shield and voltage regulator so that the teams could use a single 6-9V rechargeable battery to power their entire device. Both devices worked quite well, but the off-the-shelf device should be easier for other departments to replicate. We used a Pololu DRV8835 Dual Motor Driver Kit and a S7V7F5 step-up/step-down regulator, but many other hardware solutions exist. More details of the course content can be found online [27].

Lab Discourse Example

Lab activities were videorecorded intermittently as part of the research evaluation efforts. The following discourse example comes from students working on lab 3, "Generate a code that will turn lights (LEDs) on when lights are off and keep track of where (say, in a house) the lights are on." In practical terms, this activity involved programming a Raspberry Pi circuit where a photo-resistor sensor would sense classroom light or dark (simulated by covering the sensor with finger) and lighting up an LED elsewhere on the circuit board when the sensor noted darkness. Although the lab was individual, several students in close physical proximity would help one another. The following passage is between Jillian (J) and Nick (N), who have been sitting next to each other in lab for a couple of weeks and choose to remain lab partners in subsequent weeks due to their good working relationship. Jillian and Nick are about 30 minutes into a 3 hour lab session at the time of the clip (Figure 6).

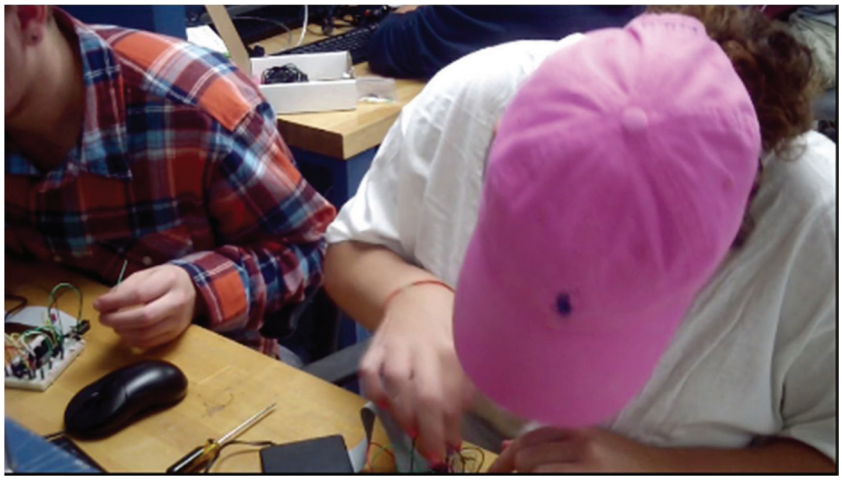

Figure 6. Videorecorded example of students working on lab task.

N (to himself): So 12, and then plugged in 13.

[N + J leaned over their circuits adding wires]

J: Ok. What did you connect these lights to? What pins...

N: Um, you gotta create your own um

J: (So)

N: ...input and outputs so...

J: (It doesn't matter)

N: Well look at this graph and anything that has GPIO you can use

[Nick gesturing at Raspberry Pi pin-out diagram on his desktop computer monitor]

J: (Uhhuh, you can use)

N: So I just used 11, 13, 15, and 16. and 12.

J: Why do you need so many?

N: Cause you need 3 outputs and 2 inputs.

[Pause, Jillian glancing down at her circuit then up at Nick's diagram]

J: Ok so, GPIO 27 which is pin 13.

N (to himself): Gonna try to put this in...

[Nick maneuvering his Raspberry Pi back into position on the circuit board]

J: Oh mine's backwards compared to this. Ok that's why.

J: So, 23.

[Pause, Jillian adding wires, Nick looking at desktop computer monitor]

J (to herself): Why is this not going in

[Jillian grabs a new wire, glances at diagram]

N: Jumped the gun

J: You did?

N: Yeah, I thought I was done but I made an if-loop with nothing in it.

[Pause, Jillian adds another wire, then looks at her circuit scratching her head]

J: Ok, and then so you connect to these lights with something and then we need one more output right, so one more on the other side.

[Jillian glances at Nick, who is preoccupied, pulls out wire to continue working]

N (to himself): But when you do both you don't have to...space

[Pause, Jillian adding wires, Nick looking at monitor]

N: Do you ever notice how in the code [Instructor] adds way too many exclamation marks?

J: Yeah. He just gets excited.

[Pause, adds a wire]

J: Ok I think it works. So now I just need to add to this right

N: Uh yeah...

In this discourse clip we see Nick and Jillian pursuing their individual projects but sharing out loud about their progress and challenges in a friendly and helpful way. The conversation easily slips between working silently, thinking out loud, asking for and offering help, making casual joking conversation, and recognizing successes and difficulties. Nick is slightly ahead of Jillian at this stage (this pattern was not stable throughout this task or other subsequent tasks, and Nick recognized Jillian as often leading him in a post-semester interview), and he is comfortable being interrupted to explain his process and thinking. Although the sharing of information might be construed as cheating in another course, it is encouraged as productive collaboration in this course—indeed although the lab is officially an individual task, conversations like these are continually going on in the background of the clip. Importantly, Nick is sharing not only the surface features of his circuit-building (e.g., pins 11, 13, 15, 16, and 12) but the rationale (choosing the pins which say GPIO) and the broader context for the choice (the need for a certain number of Inputs and Outputs). As a cognitive apprentice [28] to Nick for this portion of the lab task, this helps Jillian scaffold towards more expert reasoning and participation in her next steps. In this participation mode, Nick and Jillian both receive instructional benefits—Nick to expound on his reasoning in a way which helps him formalize and abstract, Jillian gets to move forward in implementation while walking through the cognitive steps Nick took to get there. Her quiet comments while reasoning about her own diagram suggest her glances at Nick's diagram and circuit are not about a shortcut to completion, but about assessing the differences between the circuits to access a deeper conceptual understanding. Although Nick is a few steps ahead of Jillian in this moment, he keeps a casual and humble attitude towards the process of explaining his steps which resist creating a status hierarchy between the two of them, even casually self-deprecating about his own mistakes ("jumped the gun"). The complex and open-ended tasks meant that many types of intellectual contributions were valued and (at least in this pairing) students shifted back and forth in who was taking intellectual leadership, rather than stabilizing firm identities as expert and apprentice.

It is worth noting the unique features of this classroom setting. The collaborative student interaction afforded in this lab is not present in traditional lecture settings, and is likely not a focus of homework-help discussion sections aimed at receiving help from the UTF. This peer interaction provides for a cognitive apprenticeship which can move students from novice-like into more expert-like participation. This interaction also likely has positive impacts on students' sense of community and belongingness. In addition, the lab tasks are incorporating embedded electronics which are framed as close corollaries to real life scenarios (e.g., turning lights on and off in response to ambient light level), thus creating a greater integration of programming with authentic engineering tasks. These key features of the lab intervention likely have an impact on the student identity and affective survey measures noted in the subsequent section.

Table V. Self-reported prior programming background for students in 2016 novel programming course for student respondents in sample, and total course cohort size. PB = programming background.

	No PB	Some (non-C) PB	Some C PB	Substantial C PB	Course cohort size
Traditional 2015	10	3	1	0	60
Traditional 2016	4	10	2	0	60
Novel 2015	5	12	2	5	29
Novel 2016	6	7	2	4	20

Student Identity and Efficacy Assessment

In this section we present a culminating quantitative analysis of identity and efficacy outcomes for various demographic groups of students. Prior programming background was a significant divider of experience in both traditional and novel courses, so in Table V, we indicate student self-identification of prior programming background.

In Table VI, we indicate the low response rates in the traditional course (emailed electronic survey) versus higher ones in the novel course (in-class paper survey). The low traditional course response rates limited our ability to conduct pre-post matched pairs tests for statistically significant changes, and led us instead to aggregate all PRE and POST traditional course responses as an imperfect

Table VI. Survey responses, class cohort size, and response rate for 2015 and 2016 cohorts.

	Survey responses	Class Cohort size	Response Rate
Traditional 2015 PRE	15	60	25%
Traditional 2015 POST	14	60	23%
Traditional 2015 PRE+POST	5	60	8%
Traditional 2016 PRE	14	60	27%
Traditional 2016 POST	16	60	26%
Traditional 2015 PRE+POST	5	60	8%
Novel 2015 PRE	17	29	59%
Novel 2015 POST	22	29	76%
Novel 2015 PRE+POST	14	29	48%
Novel 2016 PRE	19	20	95%
Novel 2016 POST	19	20	95%
Novel 2016 PRE+POST	19	20	95%

"control" population response for that point in the semester. The novel course had a much higher response rate for both surveys and was more appropriate for disaggregated statistical analysis.

Table VII and VIII present a culminating analysis across semesters related to efficacy and identity. The numbers in the table show the shift from the pre-survey at the start of the semester to the post-survey at the end of the semester. A positive number shows how much the average response has increased during the semester on a 7-point scale, where all scores are ranked on a scale from strongly disagree (1) to strongly agree (7). See the Appendix for a sample survey protocol. The change in opinions for all students in a course are shown in Table VII.

Since some questions are worded such that a positive likert answer is a negative course outcome, the difference measures have been shaded and italicized in green for positive shifts and shaded in red without italics for negative shifts. Almost all of the shifts were in the desired direction for students in the novel course in 2015. In 2016 the results were still positive for the first two questions. Students in the new course always felt more like they fit in as EEs after they completed the course.

Table VII. Comparison of overall survey data trends from all traditional courses to two cohorts of novel course.

2015 + 2016 ENEE 140 Data Pre-Post Comparison		I feel like I fit in as an electrical engineer.	Programming is not "real engineering."	I want to take more programming classes beyond this class, even if they aren't required.	I'm excited about the electrical engineering major.	Going into Intermediate Programming, I feel confident that I can learn coding.
Traditional	Pre Mean	5.5	2.4	5.0	6.0	5.4
Traditional	Pre St Dev	0.9	1.3	1.6	0.8	1.4
Traditional	Post Mean	4.9	2.2	4.9	5.7	5.1
Traditional	Post St Dev	1.6	1.4	1.7	1.4	2.0
Traditional	Difference	*–0.6*	*–0.2*	–0.2	–0.3	–0.3
Novel 2015	Pre Mean	5.6	2.5	5.6	6	6.1
Novel 2015	Pre St Dev	1.1	1.3	1.5	1.4	1.1
Novel 2015	Post Mean	6.4	2.1	5.4	6.5	6.5
Novel 2015	Post St Dev	0.5	1.2	1.4	0.5	0.8
Novel 2015	Difference	*0.8**	*–0.4*	–0.2	*0.5*	*0.4*
Novel 2016	Pre Mean	5.3	2.3	5.4	6.4	6.0
Novel 2016	Pre St Dev	1.4	1.4	1.1	0.5	1.5
Novel 2016	Post Mean	5.5	1.7	4.4	6.1	5.1
Novel 2016	Post St Dev	1.3	0.9	1.8	0.7	1.4
Novel 2016	Difference	*0.2*	*–0.6*	–1.0*	–0.3	–0.9

* represents statistically significant results. Difference measures in green italics represent a positive shift, difference measures in red represent a negative shift.

Students in the traditional course on average felt less like they fit in as EEs. Students in both courses felt that programming is more like "real engineering" after taking the course, but the shift in opinion was larger for students in the novel course.

Complete pre-post data sets for the novel programming course enabled a matched pairs t-test from pre-test to post-test responses of each individual, showing a statistically significant improvement the 2015 cohort ($\alpha = 0.05$, 2-tailed) for feelings of fitting in as an electrical engineer and statistically significant decline in the 2016 cohort ($\alpha = 0.05$, 2-tailed) for interest in taking more programming courses. In addition to the relative change pre- to post-test, it is noteworthy that the absolute values of post-survey mean responses are more favorable for the novel course in every case except one (2016 novel course interest in taking more programming courses). The high absolute value for the novel programming course is in part a result of more of these novel course students were rating the maximum response for the pre-test and therefore not having anywhere to "improve." Students with no or some prior programming background contributed the largest portion of the overall decrease in interest and confidence in further programming courses.

A secondary focus of the research was to document the impact and potential for the pedagogy on underrepresented minority (URM) communities. In this study, underrepresented minorities were conceived of as women and non-Asian racial minorities. Data for students from underrepresented communities only is shown in Table VIII.

The overall trends for URM students are consistent with patterns for the overall class. The question "I feel like I fit in as an EE" goes down for the students from the traditional course while the same question gets a large (though not statistically significant) increase for students from the novel course. Finally, the URM students in both courses feel that programming is more like real engineering after taking the course. However, the change is higher for students from the novel course. The remaining three questions have shifts in the undesired direction for both courses, and they are consistent and comparable to the overall class trend. In the prior cohort (with more positive results for the final three questions) data was not separated out for URM groups.

Table VIII. Comparison of shifts in survey data for students from URM communities.

Pre-Post Comparison for students from underserved populations	Number of URMS responding	I feel like I fit in as an electrical engineer.	Programming is not "real engineering."	I want to take more programming classes beyond this class, even if they aren't required.	I'm excited about the electrical engineering major.	Going into Intermediate Programming, I feel confident that I can learn coding.
Traditional course	13	–0.8	*–0.2*	–0.6	–0.4	–0.5
Novel course 2016	9	*0.8*	*–0.7*	–1.1*	–0.4	–1.3

* represents statistically significant results. Green italicized is positive shift, red not italicized is negative.

Free response survey questions reveal a range of novel course student perspectives which contributed to the negative values for interest in further programming courses and confidence in learning:

> "I hear the professor for Intermediate Programming for Electrical Engineers gives outrageous projects. I worry that I will stress out instead of learn."
> - Asian female student with no prior programming background
>
> "Although I did enjoy this class a lot, my schedule is too busy already to take another coding class unnecessarily."
> - White male student with some prior programming background
>
> "It has been a struggle for me to learn coding. I made some great progress, but the difficulty level of Intermediate Programming for Electrical Engineers worries me."
> - White female student with some prior programming background

These quotes appear to show students with a mix of optimism and genuine worry about their future encounters with programming.

CHALLENGES FOR THE NOVEL COURSE DESIGN

Having laid out the novel affordances of the class and suggested how they have benefited students, we also want to lay out some of the challenges encountered. The course curriculum was consistently being adapted and designed, in collaboration and conversation between research and instructional personnel. Revealing this pedagogical design process, then, constitutes an important piece of the intellectual work of the course effort, one which is not visible in the final curriculum and successful survey results.

Introducing New Content Within an Existing Course Sequence

One challenge of the novel course was the need to cover all of the material of the traditional introductory course, in addition to the new material. An additional credit was offered for the additional time and effort spent. This sometimes contributed to a student impression of the novel course as having a lot of content to cover and being the more challenging course option. The need to get students up to speed on programming formed a "crash course" first two week introduction, which was a noted period of stress for students newer to programming.

Likewise, the difference in content created stress in the first two weeks for students taking the intermediate course—the introductory course used linux/unix operating system consistently in

order to submit homework assignments. A linux/unix lecture was developed in the novel course in order to remedy this, but it was found not to ease the transition to the intermediate course entirely. Additional instructional redesign efforts will include integrating linux/unix into laboratory assignments, and experimenting with having the first laboratory session before the crash course has been completed. Having seen firsthand the value of students' active learning, we think the laboratory sessions can help resolve students' feelings of firehose content delivery in lecture by integrating some of the content into laboratory instructional activities.

Disparities in Programming Background

Another major challenge to the course was the disparity in prior programming between students. It became clear that many of the students in the introductory programming course had substantial programming experience, while another portion of the students were truly being introduced to programming for the first time. The learning needs of students at the two ends of this experience spectrum were extremely different, and this need for differentiation is an ongoing course design challenge. Although historically, a placement test was put in place to help address this issue, over time the introductory course became an attractive option for programming-experienced students. This appeared to happen for a variety of reasons, including course registration difficulties, lack of knowledge about the placement test, difficulty with self-awareness of how much programming experience is substantial, and the simple attractiveness for programming-experienced students of taking an easier course the first semester. This issue spans both the novel course and traditional course, and some of the remedies such as a compulsory placement test would require broader institutional coordination to implement. Next steps include a collaboration with traditional course instructors around a simple pre-test which can help evaluate student familiarity with programming beyond their own self-reporting. This sort of tracking would help instructors at least know about their students' prior experience. In the long run, it could be developed to help restructure course registration around prior experience.

V.C. Accommodating Advanced Questions in Lecture

Related to disparities in programming background was a difficulty with structuring appropriate participation in lecture. In lecture, a phenomenon where advanced students asked curious and advanced questions ended up taking a substantial amount of time away from beginning content. It also made programming background into a status marker with adverse consequences. A recent instructional intervention related to this phenomenon was creating a "2 strikes" rule where students get to ask two curious/advanced questions per semester, but on the 3rd question they are asked to see the instructor in office hours. This practical remedy encourages advanced students' curiosity without letting it take up significant instructional time or adversely impact other students.

Structuring Participation in Lab Tasks

In a first pilot iteration of lab curriculum, all lab programming tasks were individual assignments. This had an unfortunate consequence that students with less programming background were consistently slower and sometimes got stuck. Undergraduate Teaching Fellows (UTFs) likewise were spread thin amongst assisting many individual students, leaving them less available to help every student. In an iterative redesign, programming tasks were conceived of and implemented as paired assignments. This had an affordance that students were able to collaboratively teach and learn from each other on many issues, it alleviated instructional pressures on the UTF, and it reduced the pressures of being behind for less programming background students. Some individual tasks were kept depending on content needs and a desire for individual accountability. Additional design challenges related to participation on paired tasks include how to structure participation so that advanced students are not doing a bulk of the tasks, and beginning students have equal opportunities for active learning. Although these issues vary based on the individual UTF and student pairs, one strategy may be to encourage teaching assistants to define what appropriate participation looks like, and to make effective participation into a significant graded course component (rather than only evaluating final products).

CONCLUSIONS AND FUTURE WORK

Novel Course Evaluation

The novel course has been taught four times in the past three years, and has evolved gradually during that time based on feedback from the data taken during this research program. The novel course appears to be sufficient preparation for our intermediate programming class and has had a small positive impact on student retention. Students are generally satisfied with the course and leave with an improved self-image regarding their fitness as EE students and an improved understanding of the role of computer programming in their discipline. The novel lab component of the course is regarded by students as particularly valuable for their own learning.

A Hardware-driven intermediate programming course

Given the success of the hardware-driven introductory class, development of a hardware-driven version of the intermediate class has been launched. The key topics in the intermediate programming class include pointers, dynamic memory allocation and data structures, linked lists, and graphs. As with the hardware-driven introductory course, there would be a number of individual homework assignments and group labs. There would also be a multi-week final group project. Unit testing and

separate compilation would be stressed in the group labs and final group project. Projects would rotate from a number of areas including instrumentation, networking, security, image processing, and others. For example, sample networking problems could be: a) implement a small webserver, (b) implement a message passing over network, or c) implement a distributed traffic light control system. We hope to begin teaching the intermediate course next year and make the two PBL-course sequence a permanent option for our students.

ACKNOWLEDGEMENTS

This work was supported by NSF grant DUE- 1245745. We appreciate the significant contributions of Andrew Elby, Ayush Gupta, and Shuvra Bhattacharyya. We would also like to acknowledge the participation of Tudor Dumitras, Neruh Ramirez, Gang Qu, Bryan Quinn, and the many undergraduates who led the course laboratory sections.

REFERENCES

[1] E. Seymour and N. M. Hewitt, *Talking about leaving: Why undergraduates leave the sciences*. Boulder, CO: Westview Press, 2000.

[2] S. E. Walden and C. E. Foor, "'What's to keep you from dropping out?' Student Immigration into and within Engineering," *J. Eng. Educ.*, vol. 97, no. 2, pp. 191–205, 2008 [Online]. Available: https://doi.org/10.1002/j.2168-9830.2008.tb00967.x

[3] M. Prince, "Does Active Learning Work ? A Review of the Research," *J. Eng. Educ.*, vol. 93, no. 3, pp. 223–231, 2004.

[4] K. M. Calabro, "Flipping the Classroom on an Established Introduction to Engineering Design Course," in *5th First Year Engineering Experience (FYEE) Conference*, 2013, p. F1A1-F1A6 [Online]. Available: http://fyee.asee.org/FYEE2013/papers/1015.pdf

[5] R. M. Clark, M. Besterfield-Sacre, D. Budny, K. M. Bursic, W. W. Clark, B. A. Norman, R. S. Parker, J. F. Patzer Ii, and W. S. Slaughter, "Flipping Engineering Courses: A School Wide Initiative," *Adv. Eng. Educ.*, no. Fall, pp. 1–39, 2016 [Online]. Available: https://advances.asee.org/publication/flipping-engineering-courses-a-school-wide-initiative/. [Accessed: 07-Mar-2017]

[6] J. W. Dally and G. M. Zhang, "A Freshman Engineering Design Course," *J. Eng. Educ.*, vol. 82, no. 2, pp. 83–91, 1993 [Online]. Available: https://doi.org/10.1002/j.2168-9830.1993.tb00081.x

[7] C. A. Berry, D. Chang, and C. Miller, "From LEGO to Arduino: Enhancement of ECE Freshman Design with Practical Applications From LEGO," in *American Society for Engineering Education Annual Conference*, 2016 [Online]. Available: https://peer.asee.org/26972

[8] G. W. Reckenwald and D. E. Hall, "Using Arduino as a Platform for Programming, Design, and Measurement in a Freshman Engineering Course," in *American Society for Engineering Education Annual Conference*, 2011 [Online]. Available: https://peer.asee.org/18720

[9] K. M. Calabro, K. T. Kiger, W. Lawson, and G. Zhang, "New directions in freshman Engineering Design at the University of Maryland," in *Proceedings - Frontiers in Education Conference, FIE*, 2008, p. T2D6-T2D11 [Online]. Available: https://ieeexplore.ieee.org/document/4720298

[10] Lego, "Mindstorms." [Online]. Available: https://www.lego.com/en-us/mindstorms/learn-to-pro

[11] A. Soares, "Problem Based Learning in Introduction to Programming Courses," *J. Comput. Sci. Coll.*, vol. 27, no. 1, p. 36, 2011 [Online]. Available: http://dl.acm.org/citation.cfm?id=2037151.2037161

[12] M. Barg, A. Fekete, T. Greening, O. Hollands, J. Kay, J. H. Kingston, and K. Crawford, "Problem-Based Learning for Foundation Computer Science Courses" [Online]. Available: http://www.cs.usyd.edu.au/~judy/Teach/cse_pbl99.pdf. [Accessed: 03-Jul-2017]

[13] E. Nuutila, S. Törmä, and L. Malmi, "PBL and Computer Programming — The Seven Steps Method with Adaptations," *Comput. Sci. Educ.*, vol. 15, no. 2, pp. 123–142, 2005 [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/08993400500150788

[14] J. Kay, M. Barg, A. Fekete, T. Greening, O. Hollands, J. H. Kingston, and K. Crawford, "Problem-Based Learning for Foundation Computer Science Courses," *Comput. Sci. Educ.*, vol. 10, no. 2, pp. 109–128, 2000 [Online]. Available: https://doi.org/10.1076/0899-3408(200008)10:2;1-C;FT109

[15] N. Titterton, C. M. Lewis, and M. J. Clancy, "Experiences with lab-centric instruction," *Comput. Sci. Educ.*, vol. 20, no. 2, pp. 79–102, 2010 [Online]. Available: https://doi.org/10.1080/08993408.2010.486256

[16] C. Carlson and D. Day, "Transformation of an Introductory Computer Engineering Course Utilizing Microprocessors and a Focus on Hardware Limitations," in *American Society for Engineering Education Annual Conference*, 2017 [Online]. Available: https://peer.asee.org/29041

[17] M. A. Rubio, R. Romero-Zaliz, C. Manoso, and A. P. De Madrid, "Enhancing an introductory programming course with physical computing modules," *Proc. - Front. Educ. Conf. FIE*, vol. 2015–Febru, no. February, 2015 [Online]. Available: https://ieeexplore.ieee.org/document/7044153

[18] X. Zhong and Y. Liang, "Raspberry Pi: An Effective Vehicle in Teaching the Internet of Things in Computer Science and Engineering," *Electronics*, vol. 5, no. 3, p. 56, 2016 [Online]. Available: http://www.mdpi.com/2079-9292/5/3/56

[19] J. D. Steinmeyer, "Project-Based Learning with Single-Board Computers," in *American Society for Engineering Education Annual Conference*, 2015 [Online]. Available: https://peer.asee.org/24609

[20] "Arduino." [Online]. Available: https://www.arduino.cc/

[21] "adafruit." [Online]. Available: https://www.adafruit.com/

[22] "BeagleBone." [Online]. Available: http://beagleboard.org/bone

[23] "The Handy Board and the Super Cricket." [Online]. Available: http://www.handyboard.com/

[24] P. Jamieson and J. Herdtner, "More missing the Boat - Arduino, Raspberry Pi, and small prototyping boards and engineering education needs them," *Proc. - Front. Educ. Conf. FIE*, vol. 2014, 2015 [Online]. Available: https://ieeexplore.ieee.org/document/7344259

[25] W. G. Lawson, S. Secules, S. Bhattacharyya, A. Elby, W. Hawkins, T. Dumitras, and N. Ramirez, "Traditional versus Hardware-driven Introductory Programming Courses: a Comparison of Student Identity, Efficacy and Success," in *American Society for Engineering Education Annual Conference*, 2017 [Online]. Available: https://peer.asee.org/27939

[26] W. G. Lawson, S. Secules, S. Bhattacharyya, and A. Gupta, "An Application-based Learning Approach to C Programming Concepts and Methods for Engineers," in *American Society for Engineering Education Annual Conference*, 2016 [Online]. Available: https://peer.asee.org/26567

[27] W. Lawson, "ENEE 148." [Online]. Available: https://terpconnect.umd.edu/~lawson/enee148.html

[28] B. Rogoff, R. Paradise, R. M. Arauz, M. Correa-Chávez, and C. Angelillo, "Firsthand Learning through Intent Participation," *Annu. Rev. Psychol.*, vol. 54, pp. 175–203, 2003 [Online]. Available: https://doi.org/10.1146/annurev.psych.54.101601.145118

AUTHORS

Stephen Secules is an Assistant Professor in the School of Universal Computing, Construction, and Engineering Education (SUCCEED) at Florida International University. He had a prior 5-year career as an acoustical engineering consultant, and completed his PhD in Curriculum and Instruction from the University of Maryland in 2017. His primary research interests are diversity, equity, and culture in undergraduate engineering education.

Professor Wesley Lawson earned five degrees from the University of Maryland, College Park, including a Ph,D, in Electrical Engineering in 1985. In his professional career at College Park, where he has been a professor since 1988, he has worked on high-power microwave devices, medical devices, and engineering education. His current interests include integrating engineering design and other STEM concepts with computational thinking and computer language acquisition for pre- and in-service teachers, undergraduate engineering students, and middle-school and high-school students. He is an author or coauthor on 5 books and over 70 refereed journal articles and 200 conference presentations and publications.

APPENDIX – PRE-SURVEY QUESTIONS FOR FALL 2015

Name: _____

Q1 Please indicate your preference for working in groups or alone for the following scenarios.

	1 Working Entirely Alone	2 Working Mostly Alone	3 Working More often Alone than in Groups	4 Working Equally Alone and in Groups	5 Working More often in Groups than Alone	6 Working Mostly in Groups	7 Working Entirely in Groups	No Opinion
I think I would enjoy learning to program by:	❍	❍	❍	❍	❍	❍	❍	❍
I think the most productive way to learn to program would be:	❍	❍	❍	❍	❍	❍	❍	❍
I expect this class to consist of:	❍	❍	❍	❍	❍	❍	❍	❍
If I encounter programming in my professional life, I expect it to be:	❍	❍	❍	❍	❍	❍	❍	❍

Q2 Please explain one or more of these answers in your own words.

Q3 Please rate the following skills based from 1 - least helpful, to 10 most helpful. I expect these programming skills to be most helpful in this course:

_____ logical thinking

_____ specifics of the programming language (syntax)

_____ commonly used algorithms

_____ debugging skills

_____ properly formatting a code/program

_____ problem solving

_____ strategic planning to solve a problem

_____ breaking a bigger problem into smaller chunks

Q4 Please rate the following skills from 1 - least helpful, to 10 most helpful. I expect these programming skills to be most helpful in my future engineering courses:

_____ logical thinking

_____ specifics of the programming language (syntax)

_____ commonly used algorithms

_____ debugging skills

_____ properly formatting a code/program

_____ problem solving

_____ strategic planning to solve a problem

_____ breaking a bigger problem into smaller chunks

Q5 Please rate the following skills from 1 - least helpful, to 10 most helpful. I expect these programming skills to be most helpful as a professional engineer:

_____ logical thinking

_____ specifics of the programming language (syntax)

_____ commonly used algorithms

_____ debugging skills

_____ properly formatting a code/program

_____ problem solving

_____ strategic planning to solve a problem

_____ breaking a bigger problem into smaller chunks

Q6 To do well in this course, how important (1 – least important, 10 most important) do you think it will be to:

_____ get the right answer

_____ understand programming concepts deeply

_____ write code that another student could understand

Q7 Please explain one or more of these answers in your own words.

Q8 Please indicate to what degree you agree/disagree with the following statements

	Strongly Disagree	Disagree	Slightly Disagree	Neutral	Slightly Agree	Agree	Strongly Agree
I feel like I fit in as an electrical engineer.	❍	❍	❍	❍	❍	❍	❍
Programming is not "real engineering."	❍	❍	❍	❍	❍	❍	❍
I would want to take more programming classes beyond this class, even if they weren't required.	❍	❍	❍	❍	❍	❍	❍
I'm excited about the electrical engineering major.	❍	❍	❍	❍	❍	❍	❍
Coming into this class, I feel confident that I can learn coding.	❍	❍	❍	❍	❍	❍	❍

Q9 Please explain one or more of these answers in your own words.

Q10 What was your programming background prior to ENEE 140 / ENEE 148?

- ❑ No prior programming background
- ❑ Some programming experience with a different language (e.g. Java, Arduino)
- ❑ Substantial programming experience with a different language (e.g. Java, Arduino)
- ❑ Some programming experience in C++
- ❑ Substantial programming experience in C++

Q11 Regarding gender, I identify as:

Q12 Regarding race and ethnicity, I identify as:

Q13 Is there anything else about you that you think is significant in the way you will experience this class? Feel free to make any comments and explain any of your prior answers further.