Traditional versus Hardware-driven Introductory Programming Courses: a Comparison of Student Identity, Efficacy and Success

Prof. Wesley G. Lawson, University of Maryland, College Park

Prof. Lawson has earned five degrees from the University of Maryland, including a Ph,D, in Electrical Engineering in 1985. In his professional career at College Park, where he has been a full professor since 1997, he has worked on high-power microwave devices, medical devices, and engineering education. He is an author or coauthor on 5 books and over 70 refereed journal articles and 200 conference presentations and publications.

Dr. Stephen Secules, University of Maryland, College Park

Stephen received a PhD in education at the University of Maryland researching engineering education. He has a prior academic and professional background in engineering, having worked professionally as an acoustical engineer. He has taught an introduction to engineering to undergraduate engineers and to practicing K-12 teachers. Stephen's research interests include equity, culture, and the sociocultural dimensions of engineering education.

Prof. Shuvra Bhattacharyya, University of Maryland, USA, and Tampere University of Technology, Finland

Shuvra S. Bhattacharyya is a Professor in the Department of Electrical and Computer Engineering at the University of Maryland, College Park. He holds a joint appointment in the University of Maryland Institute for Advanced Computer Studies (UMIACS), and is a member of the Maryland Cybersecurity Center (MC2), and the University of Maryland Energy Research Center (UMERC). He is also a part time visiting professor in the Department of Pervasive Computing at the Tampere University of Technology, Finland, as part of the Finland Distinguished Professor Programme (FiDiPro). He is an author of six books, and over 250 papers in the areas of signal processing, embedded systems, electronic design automation, wireless communication, and wireless sensor networks. He received the B.S. degree from the University of Wisconsin at Madison, and the Ph.D. degree from the University of California at Berkeley. He has previously held industrial positions as a Researcher at the Hitachi America Semiconductor Research Laboratory (San Jose, California), and Compiler Developer at Kuck & Associates (Champaign, Illinois). He has held a visiting research position at the US Air Force Research Laboratory (Rome, New York). He is a Fellow of the IEEE. He has been a Nokia Distinguished Lecturer (Finland) and Fulbright Specialist (Austria and Germany). He has received the NSF Career Award (USA).

Andrew Elby, University of Maryland, College Park

Andrew Elby's work focuses on student and teacher epistemologies and how they couple to other cognitive machinery and help to drive behavior in learning environments. His academic training was in Physics and Philosophy before he turned to science (particularly physics) education research. More recently, he has started exploring engineering students' entangled identities and epistemologies.

William Hawkins, University of Maryland
Prof. Tudor Dumitras, University of Maryland, College Park

Tudor Dumitra is an Assistant Professor in the Electrical & Computer Engineering Department at the University of Maryland, College Park. His research focuses on Big Data approaches to problems in system security and dependability. In his previous role at Symantec Research Labs he built the Worldwide Intelligence Network Environment (WINE) - a platform for experimenting with Big Data techniques. He received an Honorable Mention in the NSA competition for the Best Scientific Cybersecurity Paper of 2012. He also received the 2011 A. G. Jordan Award from the ECE Department at Carnegie Mellon University, the 2009 John Vlissides Award from ACM SIGPLAN, and the Best Paper Award at ASP-DAC'03. Tudor holds a Ph.D. degree from Carnegie Mellon University.

Mr. Neruh Ramirez, University of Maryland, Department of Electrical & Computer Engineering

Traditional versus Hardware-driven Introductory Programming Courses: a Comparison of Student Identity, Efficacy and Success

Abstract

This paper compares an innovative approach to teaching an introductory C programming course to a traditional C programming course for electrical engineering students. Students who pass either course must subsequently take a traditional intermediate C programming course. The novel course utilizes hardware-based projects to motivate students to master language syntax and implement key programming concepts and best practices. In addition to comparing the attitudes and self-perceptions of the students in each of the introductory courses, we also look at success rates for each cohort in the intermediate programming class as well as their progress toward their degrees. The electrical engineering students who took either introductory class on average had identical GPAs. However, students who took the novel introductory C course did somewhat better than the other cohort in the intermediate traditional class. Furthermore, 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." This increase spanned a number of subgroups within the course, including students from underserved populations. Additional results, a synopsis of the two introductory courses, and a description of a technology-driven intermediate programming course are presented and discussed in this paper.

Introduction

For many years there has been an increasing emphasis on active-learning in freshman engineering courses to increase interest and improve retention in the discipline.[1] Many of those courses focus on engineering design,[2-6] and programming language instruction in these courses is of secondary importance. In these courses, programming is often taught in a fragmented way by having the students learn the basic rudiments of syntax and then modify examples by trial-and-error. We believe that the active-learning approach can enhance rigorous introductory programming courses and have been developing a proof of this concept in a multi-year NSF-funded study of a novel pedagogical intervention.

For the past few years we have offered two versions of our introductory C programming course. The first is a traditional course where students are given individual paper-based programming assignments that do not involve any hardware besides the computer itself and its peripheral devices. In the other course, students do have some individual programming assignments, but there is 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) 3B computer[7] is currently the device the students use for the hardware-based assignments, though there are many devices off the shelf today that have similar capabilities.[8] Both classes nominally require the same textbook. While the traditional course is two credits, the novel course is three credits to allow time for the hardware instruction. Students from both classes then need to take the same intermediate C programming course that is also traditional in its format.

The novel course has been taught four times for a total enrollment of 77 students. This number has been limited by resources, as we have provided RPi kits for students and all necessary hardware from an NSF grant. The lecture meets twice a week for 50 minutes and has had at most 30 students.

The once-weekly, three-hour computer labs have had at most 10 students. In the traditional course, the lecture lasts 75 minutes and the maximum enrollment is around 60 students. The hands-on, bi-weekly, 50-minute discussion sessions are limited to 12 students. The intermediate class lecture has about 50 students and meets twice weekly for 75 minutes. That class also has a once-weekly, hands-on 50-minute discussion section.

We have been using mixed research methods, including student surveys, classroom observation, and student interviews, to compare the impact of these courses on student beliefs about programming, the electrical engineering profession, and their own abilities. The surveys have concentrated on pre-course and post-course identity and efficacy beliefs of the students. Preliminary findings[9]suggest that students in the novel course find their course more collaborative and more like "real-world" engineering than students in the traditional course did. Students in the novel course also had greater self-efficacy and identity gains, particularly regarding fitness as an engineer, as compared to students in the traditional course.

In the following sections we summarize the differences in the content and pedagogy of the traditional and novel introductory courses, the success rates for both cohorts of students in the intermediate C programming course, the final results of our studies regarding student identity and efficacy beliefs from the two introductory courses, including for students from underserved populations, and our plans to develop a technology-driven version of the intermediate class.

Pedagogical differences between the two introductory C courses

Table I.A comparison of assessment weights.

Assessment	Novel course grade (%)	Traditional course grade (%)
Homework	15	20
Quizzes	15	10
Midterm	10	10
Projects	15	35
Labs	20	-
Final Exam	25	25

The two introductory programming courses are quite similar in programming language content and assessment. The weighting system for student assessment in the most recent semester is shown in Table I. While there are slight individual differences, the total weight for homework and quizzes is 30% for both classes. Both classes have one midterm at 10% and a final at 25%. The traditional class has three individual projects and they count slightly more than one-third of the final grade. The total weight for the novel course's labs and final project is the same 35%. Most of this 35% represents a group grade, with only about 1/3 of the labs being an individual effort. For the traditional course the entire grade is an individual effort.

The student learning objectives on the official syllabi of the two introductory courses are compared in Table II. 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. In both courses the students learn to work in an integrated development environment (IDE), although the traditional course does not mention this explicitly in the learning objectives; Additionally, the traditional course teaches the use of the UNIX environment for compiling, executing and submitting programming assignments. The traditional course also emphasizes good programming practices

Table II. 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 and oral formats.	

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 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 the midterm and final exams. The novel course objectives 2, 5, 6, and 7 have no corresponding objectives for the traditional course.

The selected topics for the two courses are compared in Table III. One can see from the topics list that the programming topics are virtually identical for the two courses. As mentioned before, both courses use IDEs (CLion in the traditional course, Geany in the new course), but the traditional course emphasizes UNIX in more detail. The novel course has one lecture on UNIX and one homework assignment that requires the students to use Linux to compile, debug, and run. It is hoped that the UNIX lecture will help the novel course students transition more smoothly to the intermediate class that generally also favors a UNIX environment. The electrical engineering (EE) concepts (*in italics*) are taught only in the novel course. The novel course has 25 minutes of extra lecture time each week. That is roughly the time needed to introduce the EE topics.

In summary, the two courses spend about the same amount of time covering the same basic topics in a “passive” lecture format. The two courses both have a hands-on phase for the course with a similar ceiling on the maximum number of students in a section (10 vs. 12). The hands-on novel lab is an hour longer than the traditional course’s computer lab, because the novel course students spend a significant amount of time assembling and debugging hardware, in addition to programming during the lab.

The key differences for the two courses then are (1) the collaborative nature of the novel course labs and final project and (2) the introduction and use of electronic hardware. Sample labs include photoresistor-controlled lights, Morse Code generation, distance sensors and outlier identification,

velocity estimation from accelerometer data, magnetic sensing and magnet following. 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 Fig 1. The tank was hacked and a custom integrated circuit was designed and used to control the tank motors and to make it impossible to accidently short and damage those motors.

The final semester we used an off-the-shelf vehicle as shown in Figs. 2 and 3 (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.

Table III. The selected topics for the introductory programming courses. Y = yes, covered; N = no, not covered; S = some coverage.

Topic	**Novel course**	**Traditional course**
Programming environment in UNIX	S	Y
Integrated development environment (IDE)	Y	S
Problem solving by programming	Y	Y
Data types and variable scopes	Y	Y
Logical and arithmetic operations	Y	Y
Program selection (if, if-else, switch)	Y	Y
Repetition (for, do-while)	Y	Y
Functions	Y	Y
Formatted input/output, file input/output	Y	Y
Arrays	Y	Y
Strings	Y	Y
Basic electric circuits concepts	*Y*	*N*
Sensors and actuators	*Y*	*N*
A/D converters	*Y*	*N*
Communication protocols – SPI and I^2C	*Y*	*N*

All student groups enjoyed some measure of success for their project, with about 1/3 of the groups being able to get their vehicles back to the starting point at least once.

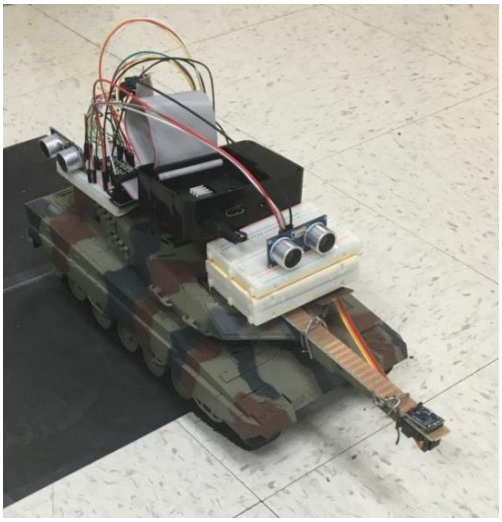

Figure 1. Original vehicle for the final project first three course offerings (hacked RC tank).

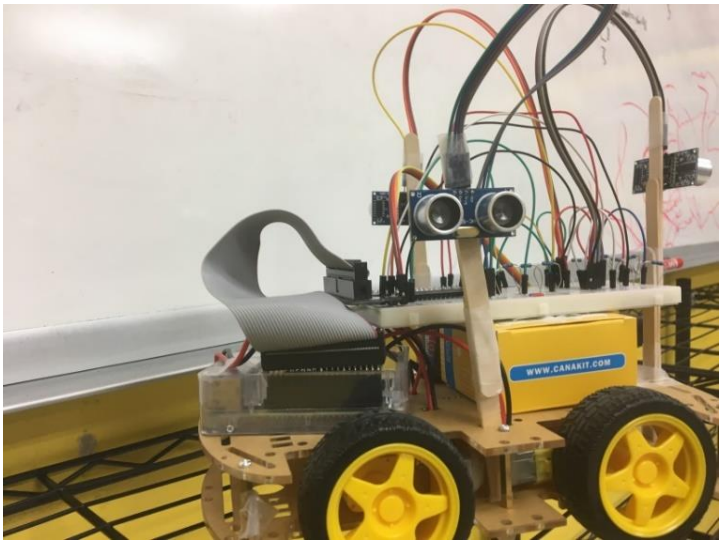

Figure 2. Vehicle for the final project for the most recent semester (off-the-shelf vehicle and motor shield).

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. During that time frame, the electrical engineering students who took those two classes were academically very similar, in that their average cumulative grade point averages to date were both about 3.33 ± 0.01. However, students who took the novel intro C course did somewhat *better* in the intermediate traditional class, in spite of the shift in course pedagogy. Those students passed the intermediate class with an average GPA of 3.22 whereas students in the traditional introductory course passed the intermediate course with an average GPA of 3.11. Since both cohorts had a standard deviation in GPA of about 0.9-1.0, these are not statistically significant improvements but do indicate a neutral or positive effect of the course.

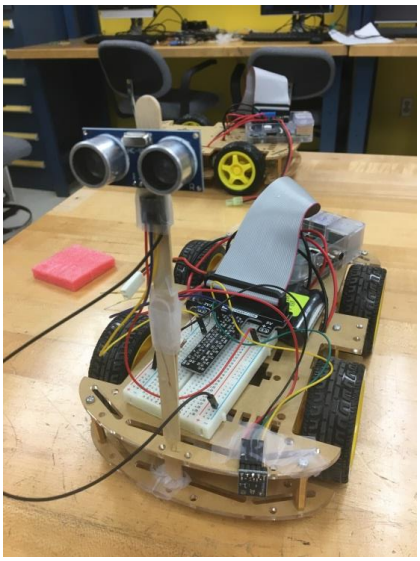

Figure 3. Another view of the off-the shelf vehicle after modified by students.

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, only 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 requirement of the EE major (unless waived by AP or placement exam), these withdrawals likely represent students leaving the department.

The traditional course was restricted to EE majors, though some exceptions were made for students who wanted to enter into our limited-enrollment program. 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, almost 87% are still in EE. A little over 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.

Student identity and efficacy

In a previous paper we presented initial survey data which included shifts towards appreciation of group programming[9]. In this paper we present a culminating analysis across semesters related to efficacy and identity (Tables IV and VI). The numbers in the tables 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, for example, 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).

The change in opinions for all students in a course are shown in Table IV. Responses for the novel course are separated by the year the course was taken. 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. 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 2-3 times larger for students in the novel course.

Table IV. 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

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 ($\alpha = 0.5$, 2-tailed) for feelings of fitting in as an electrical engineer and statistically significant decline in the 2016 cohort ($\alpha = 0.5$, 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."

As course reputation and logistics changed over the years of the study, the shifting demographic groups who enrolled in the course appeared to have an impact on the outcomes for students. In particular, interviews and observations with students in the early cohorts revealed that prior

programming background became a significant divider of experience in both traditional and novel courses. In the novel course 2016 cohort, 6 out of 19 students rated themselves as having no prior programming background, and 7 out of 19 students rated themselves as having "some" programming background in a programming language other than C. These students showed similar positive shifts in beliefs (coming to disagree that "Programming is not 'real' engineering"), but contributed the largest portion of the overall decrease in interest and confidence in further programming courses. Since this question was not asked in 2015, we do not have information on the programming experience from the novel course in 2015, nor can we accurately represent the traditional course across multiple cohorts. However, the impression based on research interviews and instructor interactions was that the 2015 cohort had more students with some prior programming experience than the 2016 cohort.

Table V. Self-reported prior programming background for students in 2016 novel programming course.

	No programming background	Some programming background (non-C)	Some C-programming background	Substantial C-programming background
Novel 2016	6	7	2	4

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 VI.

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

2015 + 2016 Pre-Post Comparison for students from underserved populations	Number of URMS	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

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 underserved 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.

Free response survey questions reveal a range of student perspectives which contributed to the negative values in 2016:

> "I hear the professor for Intermediate Engineering gives outrageous projects. I worry that I will stress out instead to 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 ENEE150 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.

A Technology-driven intermediate programming course

Given the success of the hardware-driven introductory class, development of a technology-driven version of the intermediate class has been launched. The key topics in the intermediate programming class are shown in Table VII. All of these topics would be included in the technology -driven version of the class. 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. 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. A sample instrumentation project could be to use an Inertial Measurement Unit (IMU) to detect a small embedded magnet (say imbedded in human tissue for a medical application). There would be three codes that need to be written to complete this project. The first would be to use the 3-axis accelerometer plus the gyros and 3-axis magnetometer to map out the magnetic field in the absence of the small magnetic marker. The second code would be to evaluate the data to eliminate outliers and smooth out the data. The final code would be to localize the embedded magnetic marker using only the 3-axis magnetometer and the smoothed data. Many of the labs would be used to explore specific topics from Table VII and various sensors included in the IMU. One example would be to have a code allocate memory dynamically to store all the sensor data needed to determine the current location of the IMU. A structure would be used to store a full data point plus the pointers needed for a linked list. The final data for the first code would be stored as a graph to facilitate magnetic field data retrieval for the final code.

Table VII. Topics for the intermediate programming class

unit testing
separate compilation
makefiles
Pointers
Dynamic memory allocation
Structures
Linked lists
Graphs and applications
Dynamic data structures
Abstract data types
Object-oriented design

Summary and conclusions

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 more than adequate 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. We hope to not just continue this course in the future, but also to transfer this teaching philosophy to the intermediate programming course.

Bibliography

1. Knight, D. W., L. E. Carlson, and J. F. Sullivan, "Improving Engineering Student Retention through Hands-On, Team Based, First-Year Design Projects," *31st ASEE International Conference on Research in Engineering Education*, June 22 – 24, 2007, Honolulu, HI.
2. Dally, J. W., and G. M. Zhang, "A Freshman Engineering Design Course," *Journal of Engineering Education*, pp. 83-91, April 1993.
3. Meade, J., "Change is in the Wind", ASEE PRISM, 2, May 1993, pp. 20-24.
4. Parker, J., D. Cordes, and J. Richardson, "Engineering design in the freshman year at the University of Alabama-Foundation Coalition program," Frontiers in Education Conference, 1995. Proceedings. 1995 (Atlanta, GA), pp. 4d2.5 - 4d2.8 vol.2.
5. Burton, J. D. and D. M. White, "Selecting a Model for Freshman Engineering Design," *Journal of Engineering Education*, pp. 327-332, July 1999.
6. Recktenwald, Gerald W. and David E. Hall, "Using Arduino as a Platform for Programming, Design and Measurement in a Freshman Engineering Course," ASEE Annual Conference and Exposition. Vancouver, BC. June 26-29, 2011.
7. https://www.raspberrypi.org/
8. http://www.technologyreview.com/view/514036/beaglebone-black-a-makers-dream/
9. AUTHORS. 2016. American Society of Engineering Education Annual Conference, New Orleans.