The Impacts of a New Programming Course in a First-Year Engineering Experience

By
Jessica Garcia Ramirez

A THESIS

submitted to
Oregon State University
Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented June 3, 2022
Commencement June 2022
AN ABSTRACT OF THE THESIS OF

Jessica Garcia Ramirez for the degree of Honors Baccalaureate of Science in Computer Science presented on June 3, 2022. Title: The Impacts of a New Programming Course in a First-Year Engineering Experience

Abstract approved:

Jennifer Parham-Mocello

This research paper describes the impacts of a new programming class on first-year, non-CS engineering students. In the spring of 2021, the College of Engineering at Oregon State University piloted a new programming course that will be required of all engineering majors in the following academic years. We used the pilot with non-CS engineering students to determine the interests and impacts of incorporating a required programming course in a new first-year engineering experience. We hypothesized that engineering students in a programming course would report an increase their problem-solving skills, while gaining confidence and enjoyment in programming, and we believed the programming language used in the class, prior programming experience, and other demographics played a role.

Out of the remaining 147 students who consented to take the post-survey at the end of the class, we found that only about 7% of the students did not think all engineering students should be required to learn to program, and only about 7% of the students did not think the class helped them with their problem-solving skills. As you might expect after the first programming class, the majority of the students felt only somewhat confident in the programming language they learned, but surprisingly, only 5% of the students said they did not enjoy the programming language. We found no significant difference in students’ feelings toward programming and the language used or prior programming experience, but students identifying as white male were more likely to report higher confidence and enjoying the language.

The results from this pilot suggested that the majority of students thought a programming class should be required for all engineering majors, and students thought programming helped with their problem-solving skills. We did not see evidence that teaching the course in C++ or Python impacted non-CS, engineering students’ feelings about programming. More research needs to be completed to determine if using a different instructional method without a design-first approach and sustainability focused assignments would have this same impact on first-year, non-CS engineering majors.
Key Words: computer science, first-year engineering, programming

Corresponding e-mail address: garciaj3@oregonstate.edu
The Impacts of a New Programming Course in a First-Year Engineering Experience

By
Jessica Garcia Ramirez

A THESIS

submitted to
Oregon State University
Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented June 3, 2022
Commencement June 2022
Honors Baccalaureate of Science in Computer Science project of Jessica Garcia Ramirez presented on June 3, 2022.

APPROVED:

__________________________
Jennifer Parham-Mocello, Mentor, representing School of Electrical Engineering & Computer Science

__________________________
Mike Bailey, Committee Member, representing School of Electrical Engineering & Computer Science

__________________________
Natasha Mallette, Committee Member, representing School of Nuclear Science & Engineering

__________________________
Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University Honors College. My signature below authorizes release of my project to any reader upon request.

__________________________
Jessica Garcia Ramirez, Author
## Contents

1 Introduction ........................................... 2

2 Related Work ........................................ 2
   2.1 Programming Requirement for Engineering Majors .......... 2
   2.2 Programming Requirement for Non-CS Majors .............. 3

3 Course Background ................................... 4
   3.1 Course Structure .................................... 4
   3.2 Course Motivation .................................... 6
   3.3 Research Method ..................................... 7

4 Results and Discussion ............................... 7
   4.1 Do students think the programming course increases their problem-solving skills? ................................. 8
   4.2 Do students think all engineering students should be required to learn to program? Why or why not? ......................... 9
   4.3 How confident are students in programming after taking the class? .................................................. 9
   4.4 How much do students enjoy programming? ................... 9
   4.5 Does the programming class impact students’ choice of major? If so, how? ....................................... 10
   4.6 Do the impacts/outcomes differ among students from different demographics: prior programming experience, the programming language learned, gender, ethnicity, and race? .................. 11

5 Conclusions ............................................ 11
1 Introduction

This research paper describes the impacts of a new programming class on first-year engineering students. In the spring of 2021, Oregon State University’s College of Engineering piloted a new programming course with approximately 150 first-year, non-CS majors. In the following academic years, this class will be required of all engineering majors as the last class in their new yearlong Engineering+ experience.

For this study, we used data from non-CS, engineering students in the pilot to determine the impacts of incorporating a required programming course in a first-year engineering experience. We hypothesized that engineering students in the programming course would think that the course was valuable to their education and increased their problem-solving skills, while they gained confidence and enjoyment in programming, and we believed the programming language in the class, prior programming experience, gender, and race played a role on the impacts.

This paper focuses on answering the following research questions:

1. Do students think the programming course increases their problem-solving skills?
2. Do students think all engineering students should be required to learn to program? Why or why not?
3. How confident are students in programming after taking the class?
4. How much do students enjoy programming?
5. Does the programming class impact students’ choice of major? If so, how?
6. Do the impacts/outcomes differ among students from different demographics: prior programming experience, the programming language learned, gender, and race?

The rest of the paper is structured as follows. First, we describe the motivation and related work behind requiring a programming course of all engineering students Section 2. Then, we provide details on the programming course in Section 3, and we report on how student data was collected in Section 3.3. In Section 4, we provide an overview of the programming course’s impacts on students’, and finally, we present conclusions and directions for future work in Section 5.

2 Related Work

2.1 Programming Requirement for Engineering Majors

Most academic programs require students to learn some form of programming, usually using a domain-specific language, such as MATLAB [1] or Excel [2]. Very rarely does
an engineering degree require learning a general purpose language, such as Python or C++, which are typically learned in elective courses.

Even if an engineering degree requires students to learn programming, most engineering degrees do not require their students take the programming class in their first year. In fact, most programming classes for engineers are numbered at the Sophomore level.

In engineering degrees where programming is required, MATLAB [1] is a common choice of programming language in many academic programs due to its easy syntax and computing environment [3]. The simplicity allows for engineers of all backgrounds to learn and adapt the language.

Researchers [3] analyzed a first-year engineering program designed to guide all engineering students to think about code with the goal of all engineering students being able to transfer and apply their knowledge and skill set to any engineering discipline. Students were able to solve real problems computationally which allowed them to increase their problem-solving skills which were applicable to multiple aspects of engineering beyond the course [3].

The engineering program focused on teaching students MATLAB. When surveying students about what skills they believed helped them be able to read code snippets from 3 languages and provide comments for each of them, researchers found a common theme: syntactic constructs. Among these languages were C++, Python, and Java, which have similar flow [3]. The study described students using their syntactic constructs skill to comment on “how to create specific coding elements or what the elements do, such as plotting, creating vectors, functions, variables, loops, conditionals, and MATLAB syntax.” Familiarity with syntactic constructs also helped students identify similarities and differences between the languages from the 3 code snippets provided and better understand code they had not written.

There was very little research on the impacts, but some studies included looking at ways to help students acquire the necessary skills and tools to achieve an engineering bachelor’s degree [4]. A study at Michigan Technological University focused on the university’s common first-year engineering program, which all incoming engineering students (approximately 1,000 students) had to take [4]. The program emphasized the importance of a near-peer mentoring programming, where students were put in groups of 24 students and assigned to a teaching assistant [4]. Participation in these groups was mandatory and counted as attendance and participation points [4]. The study found that students showed more engagement in the course overall as a result of being a part of these groups which helped them deepen their understanding of the concepts [4].

2.2 Programming Requirement for Non-CS Majors

One study showed that programming should not be taught in the first CS class to non-CS majors [5], and another study showed that application matters when teaching non-
CS majors [6]. At Georgia Institute of Technology, all incoming students are required to take a computing course [6]. At the time of the university-wide requirement, there was only one CS course that these incoming students could take [6]. The issue was that the CS course was an introductory course for CS majors [6]. The university saw low success rates among these non-engineering students taking the course, which drove students as well as faculty members to push for change. As a result, the university created a new computational course for students of all backgrounds [6].

The main goals of the new course were to 1) better satisfy the needs of students with various interests and 2) have a long-lasting impact on students and their computational skills [6]. The course was built to have assignments that ranged from text manipulation, HTML, to fetching index pages of a news website. These assignments built up in complexity throughout the course which allowed students to see growth in their understanding of computation and reach a level of satisfaction [6].

After the new course, students continued to show interest in CS by engaging in conversations with their peers and continuing to write computational programs in languages like Python [6]. Although non-CS students may not use their programming skills directly in their future career, research has shown that it is still important to prepare them to solve a range of complex problems across multiple disciplines [7].

3 Course Background

3.1 Course Structure

During the spring of 2021, the College of Engineering offered two sections of the new programming course to a total of 158 students. One section was taught in C++ with 60 students, and the other section was taught in Python with 98 students. Both sections covered core programming concepts, such as variables, expressions, conditions, control structures, functions, and arrays/lists. Both sections met twice a week for a 50-minute lecture led by the section instructor and once per week for a 110-minute studio led by one or more undergraduate learning assistants (ULAs) for a total of 10 weeks. Due to the COVID-19 pandemic, lectures, studios, and office hours were delivered synchronously via Zoom. Course materials, such as handouts, readings, and lecture slides, were available for students to access through the university’s learning management system called Canvas [8].

The course focused on developing students’ problem-solving skills, algorithmic thinking, debugging practices, and designing programs to solve sustainability problems using universal design principles. The course learning outcomes stated that by the end of the course students would be able to:

- Demonstrate the ability to create a computer program to solve a problem using universal design.
- Demonstrate the use of software to perform engineering problem solving.
• Use critical thinking to identify computational solutions and articulate limitations related to social structural inequities such as: racial, cultural, gender, socioeconomic and accessibility.

• Describe the separate roles of modeling and analysis in engineering practice.

• Acquire and apply new knowledge from external sources in engineering computation.

• Refine goals for academic, personal, and professional achievement by generating a resume, digital portfolio, and/or code repository.

Grades for the C++ section were composed of homework assignments (30%), studio activities (20%), midterm exams (30%), participation credit (5%), and a digital portfolio (15%). Grades for the Python section were composed of homework assignments (25%), studio activities (25%), midterm exams (25%), participation credit (10%), and a digital portfolio (15%). There was only a difference of 5% in all categories except for the digital portfolio.

Students in both sections worked on creating a digital portfolio of their computational work in the class to be used as part of their resume. The grading system used in both sections consisted of twelve grades: A, A−, B+, B, B−, C+, C, C−, D+, D, D−, and F. Students were required to obtain a grade of C or better in the course to count it toward their degree requirements.

Lecture attendance for both course sections were strongly encouraged but not required. However, there was participation credit for attending lectures. Attending studios was required for students to complete the studio report and get checked off by a ULA before the end of the studio session. These weekly studio reports encouraged students to work in a group but also included individual activities.

Studio activities and homework assignments were designed for students to solve sustainability problems using computation. There were a total of six assignments, and each assignment consisted of two parts: design and implementation. The design was due one week after the assignment was assigned, and the implementation was due one week after the design due date. This allowed students to get feedback on their design from the instructor or ULA and make any revisions before moving onto the implementation step.

During Week 1, Students were introduced to George Polya’s steps for solving problems from the book, “How to Solve It” [9]. The steps consisted of 1) understanding the problem, 2) devising a plan, 3) carrying out the plan, and 4) looking back and testing the solution. The first part of the assignment required students to submit a PDF document with a written statement of their understanding of the problem requirements, specifications, assumptions, and any questions they had about the problem statement.

After breaking down the problem, students used their analysis to design a plan using a flowchart or pseudocode to map out their solution to the problem. The first
part also required students to create good, bad, and edge cases to look back on and test their program with after the implementation. The second part of the assignment was the actual implementation either in C++ or Python, depending on the section.

Throughout the term, students incrementally worked on their digital portfolio using basic HTML and CSS. From the very first studio, students in the C++ section learned how to SSH onto the engineering servers and create a session in Linux using the terminal prompt. MacOS or Linux users used the built-in terminal application and Windows users downloaded MobaXterm, a free secure shell (SSH). The studio also covered basic Linux commands to allow students to create, edit, save, compile and run .CPP files. On the other hand, students in the Python section used Visual Studio Code for their coding environment.

3.2 Course Motivation

Because first-year students often switch majors, and to support the exploratory General Engineering program, the first orientation courses across COE were officially interchangeable for students. While COE did this to support exploration and students who were still deciding on the right major, all orientation-related content/courses were only offered Fall term. This meant that if students did not make the right choice the first time, they were jumping blindly into their second choice for major.

Despite the fact that the Course Learning Outcomes (CLO’s) did not match across the majors, first-year courses were largely treated as interchangeable. As a consequence, no orientation course was a prerequisite for other courses. As orientation courses were controlled by different schools in COE without a mechanism for coordination, these courses continued to diverge in content and focus. We found that our orientation courses were often less about exploration (except within the context of the school or program) and general success skills, and instead, they were often treated as the first course in the major. While understandable from the viewpoint of the pressure on programs to fit more into the curriculum, this was antithetical to the nature of these courses and to supporting students in finding the right major. Because of the large number of COE students switching majors during their first year (approximately 30% of 1200 students), instructors teaching subsequent courses often had to bridge the preparation gap, or students were set up to fail.

For the School of Electrical Engineering and Computer Science (EECS), there was an added barrier. Two first-year programming courses beyond the orientation class were prerequisites for each other and the second-year EECS courses. Even though EECS offers these courses every term, students who switch into ECE and CS were almost certain to have to extend their time at the university in order to graduate.


3.3 Research Method

To conduct the study involving students, approval from the Institutional Review Board (IRB) was obtained. At the end of the course, students were invited to participate in a post-assessment Qualtrics survey. Most questions that we analyzed had three response options, except for one question on the impacts of the class on their major choice with five responses. All responses were numerically quantified using a Likert-scale.

To measure the impacts of the ENGR 103 programming class on non-CS engineering majors, we asked the following questions in the post-survey:

- How much programming experience (in any language) did you have before taking ENGR 103?
- Do you think all Engineering students should be required to learn to program? Why or why not?
- How confident are you in programming in the language learned in ENGR 103?
- How much did you enjoy programming in the language learned in ENGR 103?
- Do you think ENGR 103 helped you with your problem-solving skills?
- Reflect on ENGR 103 this term, please indicate your opinion on the following: This course influenced my major.

4 Results and Discussion

To determine the programming course’s impacts on students, we analyzed students’ post-course questionnaire responses to understand their perspectives and feelings. We used observations of the number of student responses to questions and trend in responses to determine the general impacts of the ENGR 103 programming class on non-CS engineering majors. We used the Wilcoxon rank sum test with continuity correction with a p-value $\leq 0.05$ for the significance level for the questions about impacts based on section, gender, race, ethnicity, underrepresented minority, and programming experience. We also used “underrepresented minority” or “URM” to describe students who were either female or non-white and “non-URM” to describe white male students. We recognize the threat to validity this causes.

At the end of the course, 132 students out of the remaining 147 consented to participate in the survey. Among the 132 consenting participants, 95 learned Python and 53 learned C++, 27 were female (20.45%) and 105 were male (79.55%), 92 were white (69.70%) and 40 identified with a different race (30.30%), 112 were Non-Hispanic and 20 were Hispanic, 76 were Non-URM and 56 were URM, and 61 had programming
experience and 71 did not have programming experience (see Table 1). These demographies were representative of the 2021/2022 university demographics (22.59% female and 29.98% non-white) [10].

<table>
<thead>
<tr>
<th>86 Python</th>
<th>46 C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>105 Male</td>
<td>27 Female</td>
</tr>
<tr>
<td>92 White</td>
<td>40 Non-White</td>
</tr>
<tr>
<td>112 Non-Hispanic</td>
<td>20 Hispanic</td>
</tr>
<tr>
<td>76 Non-URM</td>
<td>56 URM</td>
</tr>
<tr>
<td>61 Programming</td>
<td>71 No Programming</td>
</tr>
</tbody>
</table>

Table 1: Consenting Student Demographics

4.1 Do students think the programming course increases their problem-solving skills?

Only 6.7% of students (9 out of 132) who consented to participate in the survey did not think that the course helped them with their problem-solving skills (see Figure 1. The majority of the class (89 out of 132, 67%) thought the course increased their problem-solving skills. This could be due to teaching Polya’s steps for solving problems in the class and requiring designs before writing the program. Another experiment would need to be conducted without using Polya’s problem solving steps and a design-first approach to programming.

Figure 1: ENGR 103 Increased Problem-Solving Skills
4.2 Do students think all engineering students should be required to learn to program? Why or why not?

Only 6.7% of students do not think that all engineering students should be required to learn to program (see Figure 2). Surprisingly, 63% of non-CS engineering students (83 of 132) thought that all engineering majors should have to learn to program. Students believed that as technology becomes a bigger part of our lives, understanding the basics of programming could help them in any engineering path they chose.

![All ENGR Students Learn to Program](image)

Figure 2: All Learn to Program

4.3 How confident are students in programming after taking the class?

We found that 81% of the students (107 out of 132) were only somewhat confident in programming in the language they learned in their course, while <10% were not confident at all or extremely confident (see Figure 3). This could be due to the 10-week quarter, instead of a 15-week semester. In any case, 10 weeks was not enough time to make students feel confident in programming.

4.4 How much do students enjoy programming?

Even though only 4.5% of students (6 out of 132) said that they did not enjoy programming in the language learned in their class, 36% of the students (48 out of 132) only enjoyed programming somewhat (see Figure 4). However, over half of the non-CS engineering students (78 out of 132) reported enjoying programming a great deal.
4.5 Does the programming class impact students’ choice of major? If so, how?

About half of students (62 out of 132) were either neutral or did not think the class influenced their decision in choosing a major, even though the overwhelming majority of students think the class helped them with their problem-solving skills and all engineers should take the class (see Figure 5). However, the other half of the participants in the study (70 out of 132) said that the class either somewhat or strongly influenced their choice in major. Those who did feel the class influenced their choice in major reported that the course gave them insight on career paths that involved computation, but that does not mean that they necessarily wanted to be a CS major.
4.6 Do the impacts/outcomes differ among students from different demographics: prior programming experience, the programming language learned, gender, ethnicity, and race?

For the last question, we used a t-test to compare the means and a Wilcoxon rank sum test to compare the medians between the two populations for each demographic in question. The impacts and outcomes did not differ significantly among any demographic, except for a higher confidence and enjoyment in programming among students identifying as white male versus an URM.

For confidence in their programming, white male students reported more confidence than students identifying as a URM with a p-value=.02 (see Figure 6). For enjoyment of programming, more white male students reported enjoying programming a great deal than those identifying as a URM with a p-value=.04 (see Figure 7). While it was only a small number of students who reported that they did not enjoy programming at all, we found it interesting that a higher percentage of white male students reported not enjoying programming at all compared to who were not white male.

5 Conclusions

The results from this pilot suggested that the majority of students thought a programming class should be required for engineering majors, and students thought programming helped with their problem-solving skills. We did not see evidence that teaching the course in C++ or Python impacts non-CS, engineering students’ feelings
Figure 6: Confidence Based on Underrepresented Minorities

Figure 7: Enjoyment Based on Underrepresented Minorities
about programming. However, the language used may impact students’ DWF or final grade. More research should be completed to determine if using a different instructional method without a design-first approach and sustainability focused assignments would have this same impact on non-CS, engineering majors.

From this experience, we learned that students only felt somewhat confident in programming after a 10-week course. Students also recognized that learning to program allowed them to build critical thinking and computational skills, which will be essential for students’ to solve complex problems in the 21st century.

We were surprised by the wide range of positive students’ feelings toward learning a programming language in a synchronous, online class environment during the COVID-19 pandemic. As with any experience report or research study, the evaluation of these results were only from the consenting students in the class and might not be generalizable; other students might have different experiences. Therefore, there may be other factors, other than the programming languages used, demographics, and course structure, that played a role in the outcomes. However, this study gave some insights into how non-CS engineering students felt about being made to learn to program in their first year.

In the future, all engineering students’ will be required to enroll in the new programming course. The university will offer on-campus courses and fully online courses. In the future, the online courses will use asynchronous learning, which might impact the first-year engineering students differently than an online synchronous class.
References


