Developing an Introductory Computer Science Course for Pre-service Teachers

Computational thinking (CT) involves breaking a problem into smaller components and solving it using algorithmic thinking and abstraction. CT is no longer exclusively for computer scientists but for everyone. While CT does not necessarily require programming, learning programming to enhance CT skills at a young age can help shape the next generation of children with knowledge that can help them succeed in our technological world. In order to produce teachers who are able to incorporate programming and CT into their future classrooms, we created an introductory Computer Science course (CS0) targeting future K-8 STEM teachers yet open to any student to enroll and learn computer science. We used a mixed-methods approach, examining both quantitative and qualitative data based on self-reported surveys, classroom artifacts, and focus groups from four semesters of data. We found that after taking the course, students' self-efficacy in CT increased and while education students initially had lower confidence in their computing abilities than computer science students in the course, by the end of the semester there were no differences in their perceived and actual coding abilities when compared with computer science students. Furthermore, education students had many ideas on how to incorporate similar projects into their own future classrooms.

Keywords: computational thinking; coding; CS0; pre-service teachers; computer science for all

1 Introduction

Technology is ubiquitous with 81% of Americans reporting to be online daily, and 28% of those online almost constantly (Perrin & Kumar, 2019). By providing students with a foundation in Computer Science (CS), we can better prepare them for the future. However, in the United States, females report a lower interest in CS than males, minority students have less access to computers at home, and African American/Black students have fewer CS classes in schools (Google Inc. & Gallup Inc., 2016a). In order

to address the lack of diversity and the gender gap with regard to CS, we need to engage all students with foundational CS skills early on in their classrooms. However, schools report a lack of qualified teachers in CS, and due to priorities with other classes in the curriculum, principals and superintendents would like CS education merged with other subjects (Google Inc. & Gallup Inc., 2016b). In order for teachers to be able to integrate computer science concepts in their classrooms we need to provide training for in-service and pre-service teachers and think about what is CS and how to train teachers to be able to use it in their classrooms.

Computational Thinking (CT) is a central component of CS. It is a type of problem-solving using decomposition, abstraction, and algorithmic thinking. In the United States, CT is now a principal part of the K-12 Computer Science Framework (2016) and included as a required practice in the Next Generation of Science Standards (NGSS), the K-12 science content standards in the United States. Wing (2006) argues that CT skills should no longer be only for computer scientists, but for everyone. In order to ensure that CT is truly available for all, Wing (2008) stresses that these skills need to be taught to children early on. The recent "Computer Science for All" initiative encourages exposing children in K-12 schools to CS and CT skills.

Yadav, Stephenson, and Hong (2017) point out that while in-service teachers are receiving professional development training opportunities in CT, there is less preparation happening at the pre-service teacher level. Therefore, our goal is to provide future teachers with the training necessary in CT and coding in order for them to be able to use these skills in their future classrooms. Our university has an undergraduate math and science content preparation program for pre-service elementary and middle school teachers. In order to prepare these future teachers to be able to use CT and coding in

their future courses we created an introductory Computer Science course, called Computer Science for All, that will be required for students in this program.

To improve the design of the course, a CS university professor teamed up with a K-8 CS teacher who is also a CS integration specialist and instructional coach for her school district. This collaboration provided a way to start the process on integrating current pedagogical CS content into the course with real examples on how future teachers can use what they are learning in their classrooms.

To design the course, we used a CS0 framework. CS0 is a course typically targeted for early Computer Science majors who are not ready for CS1, the first programming course. Rather than having computer science students register for a programming language like C++ or Java, a CS0 course can be a good precursor to better prepare them for programming courses and improve retention rates in CS1 (Doyle, 2005; M Rizvi & Humphries, 2012). We created a CS0 course that, while housed in the CS department, will become a required course for students in our pre-service STEM K-8 teacher program. We allowed students in other disciplines to register for it as well to demonstrate that computer science is for all. Everyone can benefit from having a basic understanding of CS, and it is a skill particularly useful in pre-service teacher programs in order for future teachers to use CS skills in their classrooms.

2 Background

2.1 Incorporating CT in Pre-service and In-service Classrooms

Shute et al. (2017) conducted a thorough review of CT in the literature, which they define as a "conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts." In the United Kingdom, a guide for teachers was

produced to understand CT based on five concepts: algorithmic thinking, decomposition, generalization, abstraction, and evaluation (Csizmadia et al., 2015). Using CT, students can break complex problems into simpler ones that are easier to solve through algorithmic thinking, decomposition, abstraction, and finding patterns. Wing (2006) argues that CT is becoming a fundamental skill as important as reading, writing, and arithmetic. Mishra et al. (2013) reason that using CT moves children from being consumers of technology to creative problem solvers with skills they can use later to enhance their chosen field.

Shute et al. (2017) found when reviewing the CT literature that Scratch and LEGO robotics were often used to increase CT skills. Scratch is easily accessible through the Web, allows students to "remix" existing projects, and exposes students to common programming concepts, such as sequences, loops, parallelism, events, conditionals, operators, and data (Brennan & Resnick, 2012). Scratch is a popular educational programming language used in elementary and middle schools and uses blocks that connect together to create programs (Resnick et al., 2009). Similarly, LEGO robotics use pieces that easily snap together and allow students to create new structures and ideas (Resnick et al., 2009). It also uses a visual programming platform, where students can drag-and-drop blocks of code to the screen, but instead of seeing output on the screen they can view the output of their code as the robot itself responds to the commands, providing a physical representation of the code. This physical representation of code through robotics can help foster students' understanding of abstraction (a key component of CT), which elementary students often struggle with.

Grover et al. (2015) included an introductory computer science course in a seven-week curriculum for middle school students and included an assessment to evaluate students' ability to transfer what they learned in a block-based Scratch

environment to a more text-based programming language. They found that after explaining the syntax to them, students were able to understand text-based code snippets.

Maloney et al. (2008) examined 536 Scratch projects in an after school club and found that students were engaged with programming in Scratch and used many programming concepts such as user interaction, loops, conditionals, and communication and synchronization. Similarly, Meerbaum-Salant, Armoni, and Ben-Ari (2010) found that while middle school students had some difficulties with initialization, variables, and concurrency, overall most students were able to understand computer science concepts. Weese and Feldhausen (2017) used Scratch in a summer STEM institute for 5th-9th graders and found students had increased self-efficacy in CT after the program.

Scratch has also had promising results when used in pre-service teacher classrooms. In a science methods course for pre-service teachers, Adler and Kim (2018) found that pre-service teachers were able to model science concepts using Scratch and had ideas for incorporating it as a tool in their own future classrooms. Cetin (2016) compared Scratch-based instruction with the C programming language for pre-service teachers and found that participants using Scratch had better programming performance. Scratch can be an effective way to introduce pre-service teachers to programming concepts in a visually intuitive and user-friendly way before using a text-based language.

Just as Scratch can increase students' CT skills, many educational programming robots have been used to enhance CT skills for pre-service teachers (Jaipal-Jamani & Angeli, 2017) and in K-12 classrooms (Berland & Wilensky, 2015; Bers, Flannery, Kazakoff, & Sullivan, 2014; Papadakis & Orfanakis, 2016). Ioannou and Makridou (2018) reviewed articles pertaining to educational robotics with regard to CT and found

that educational robots can enhance students' cognitive and social skills. Jaipal-Jamani and Angeli (2017) modified a science methods course for pre-service elementary teachers to include robotics using the LEGO WeDo robots, which are designed for second through fourth grade students, and found that there was an increase in pre-service teachers' interest in learning about robotics, self-efficacy to use robotics when teaching science, and in their computational thinking skills. Kim et al. (2018) used robotics as a medium to examine debugging methods used by pre-service teachers. They found that locating bugs was difficult for pre-service teachers and that they could be taught more effective strategies for debugging.

Yadav et al. (2017) argue for the exposure of pre-service teachers to CT in their education technology courses in order for them to have a foundation in CT which they can later learn to integrate into the content through a methods course. We argue for a separate new course for pre-service teachers, based on CS0 courses, which prepare early computer science majors with coding and CT skills. Rather than including a couple of modules in an existing course to increase CT skills, we created a 15-week computer science course that will be required for educators using a CS0 framework.

2.2 CS0: An Introductory Computer Science Course

Many computer science programs often face declining retention rates and low performance in CS1, the first programming course. To improve student retention and performance, many universities have been incorporating a CS0 course that computer science students can take before CS1 (e.g., Doyle, 2005; Rizvi & Humphries, 2012). CS0 courses often include engaging and intuitive drag-and-drop programming languages to introduce students to programming concepts, such as Scratch, Alice, MIT App Inventor, and Robotics (e.g., Anewalt, 2008; Powers, Ecott, & Hirshfield, 2007; M Rizvi & Humphries, 2012; Turbak, Sherman, Martin, Wolber, & Pokress, 2014;

Uludag, Karakus, & Turner, 2011). These drag-and-drop programming languages are considered visual programming languages and can be simpler for those new to coding, as they eliminate the need to worry about syntax, such as including braces or a semicolon, or remembering the name of a keywords or functions, due to the drag-and-drop nature of these languages. Rather than typing, these environments rely on block-based coding.

Moskal, Lurie, and Cooper (2004) developed an introductory computer science course with Alice, which has a 3D environment, and found that for high risk students, the course significantly improved their grades in CS1 as well as their retention in computer science. Powers et al. (2007), however, found that while the object-oriented approach of Alice makes it a good choice at the university level, it can lead to misconceptions on how object-oriented languages work.

Malan and Leitner (2007) propose using Scratch in earlier CS courses before students have been introduced to a first language like Java. They found that using Scratch excited students in addition to introducing them to programming. Rizvi et al. (2011) created a CS0 course with Scratch and found that students in the course had a high self-efficacy in programming, and performance in CS1 for students who took CS0 was greater than in previous years. In a later study, Rizvi and Humphries (2012) found retention rates improved since introducing the CS0 course and this pattern continued the following year.

Robotics are also used in CS0 courses through visual programming environments. Grabowski and Brazier (2011) used LEGO Mindstorms robots in CS0 to help with recruitment and retention and found that students were interested and engaged in the course and that the retention rate of students who completed the course was 100%. Pearce and Nakazawa (2008) created three CS0 courses at their university on

Web, Introduction to Robotics, and Storytelling with Alice. They found this approach increased enrollment in CS1 by 77%, with Alice and robotics helping retain more students in CS1 than the traditional Web CS0.

Turbak et al. (2014) used the block-based programming environment, MIT App Inventor, to introduce events in a CS0 course and found many challenges and ways in which introducing events with App Inventor could be improved. Rather than relying on one programming environment, Uludag et al. (2011) used a unique combination of Scratch, LEGO Mindstorms robotics, and App Inventor in a single CS0 course, which could attract both majors and non-majors.

In addition to computer science majors, CS0 courses have been used to introduce computer science concepts to non-computer science majors. Cliburn (2006) created an introductory course for non-majors and found that the course led to more students enrolling and succeeding in programming courses.

While CS0s often use visual programming languages to introduce computer science concepts, some CS0 courses focus on incorporating text-based languages, such as Python. Agarwal and Agarwal (2006) discuss how Python is a good choice for CS0 as the syntax is simpler than other high-level languages which will minimize students' difficulties with learning a first language.

We decided to create a CS0 course for educators that, while open to any major, will be required for our math and science pre-service elementary and middle school teachers and focuses on teaching computational thinking and programming concepts, using two visual-based programming platforms, Scratch and LEGO Mindstorms Education EV3 robots, and concluding with a text-based programming language. We chose Scratch and LEGO robots due their success being used in CS0 courses (Grabowski & Brazier, 2011; Malan & Leitner, 2007; M Rizvi & Humphries, 2012;

Mona Rizvi et al., 2011; Uludag et al., 2011) as well as in K-8 (Grover et al., 2015; Ioannou & Makridou, 2018; Maloney et al., 2008; Resnick et al., 2009; Weese & Feldhausen, 2017) and pre-service teacher classrooms (Adler & Kim, 2018; Cetin, 2016; Jaipal-Jamani & Angeli, 2017). We chose the LEGO Mindstorms, rather than the LEGO WeDo, as the Mindstorms are designed for children ages 10 and up and more of the pre-service teachers in our program were future middle school teachers and we did not have time to cover both in a single course. Furthermore, after learning with the LEGO Mindstorms, pre-service teachers who want to use the WeDo sets in their future classrooms should be able to easily transfer what they learned to the LEGO WeDo, which is designed for younger children.

The goal of this new course is to introduce future K-8 teachers to environments that they can later use in their own teaching. One of our fundamental learning objectives is to integrate CS within an interdisciplinary context, relating concepts to other disciplines such as science, mathematics, and the social sciences.

Our research questions are as follows:

(1) Can a CS0 course significantly improve pre-service teachers' self-efficacy in CT?
(2) How will education students compare to CS students in the same course?
(3) After completing this course, will pre-service teachers be prepared to incorporate CT and coding into their future classrooms?

3 Methodology

In order to assess the effectiveness of the new Computer Science CS0 course, Computer Science for All, we used a mixed-methods approach, where we collected both quantitative and qualitative data in a single study (Johnson & Onwuegbuzie, 2004). In addition to collecting and analyzing artifacts and grades from the course projects, we collected data from a pre- and post-survey of students enrolled in the course each semester. In order to gather qualitative data and triangulate the quantitative results, we also conducted focus groups which was limited to the education students enrolled in the course.

Data collection occurred after we obtained approval from our university's institutional review board. The pre- and post-surveys as well as the focus groups were conducted by our grant external evaluator and were optional and anonymous. This minimized potential research ethics difficulties with regard to the power relationship between the instructor and students.

3.1 Participants

Participants included students enrolled in the Computer Science for All course in all four semesters that it was offered. To recruit students for this new course, we posted flyers around campus, particularly targeting STEM and education areas. In addition, advisors for both computer science and the STEM elementary and middle school education program mentioned the course to students they spoke to. While this course will become a requirement for those education students, at the time it was offered it was not yet a requirement.

Sixty eight students (42 male and 26 female) enrolled in the course (20 in Spring 2018, 17 in Fall 2018, 15 in Spring 2019, and 16 in Fall 2019). Of those 68 students, 18 (26%) were from education, 25 (37%) were computer science students, 3

interdisciplinary studies, 2 in graphic design, 2 in biology, 2 in accounting, 2 in management, 1 mathematics, 1 marketing,1 human resource development, 1 justice studies, 1 communication, and 9 undeclared majors.

Of the 25 Computer Science students, many were early on in their majors: six had not yet taken CS1 (the first programming course), eight enrolled in the course alongside CS1, and four took it along with the second programming course. Seven of the CS students were advanced majors taking the course as it fulfilled a general education requirement. For the early CS students, particularly those who had not yet taken programming 1 or were currently taking it alongside programming 1, they registered due to a recommendation from a faculty member or advisor. The education students who registered for the course did so based on a recommendation from their program advisor who told them about the course. Note that two of the education students were current teachers. One was a CS teacher who took the course to help prepare her for CS endorsement courses, and the other was a mathematics teacher who wanted to learn more about CS to potentially teach it in his school. Furthermore, one student listed as education later switched her major to Computer Science after taking the course. Lastly, one Computer Science student is listed twice as he failed both the course and programming 1 and took them together again the following year.

3.2 Implementation of Computer Science for All

In the first week of the course, students were introduced to the Computer Science for All initiative and they were required to read "Computational Thinking for Teacher Education" (Yadav et al., 2017) in order to understand why CS is important in pre-service teacher curricula. Class discussions included how CS fits into almost every discipline and helps with real world situations such as robotics in healthcare and predicting natural disasters.

In the second week of the course, we described why computational thinking is important in K-12, and how it is now one of the Science and Engineering Practices in the Next Generation of Science Standards (NGSS) in the United States. We used hands-on computational thinking exercises that were not on a computer. For example, students created algorithms for brushing their teeth, helping their classmate create a peanut butter and jelly sandwich, and allowing a hypothetical robot to walk in a square. They also learned to develop their own flowcharts for everyday activities. Examples of student flowcharts include getting ready for school in the morning or taking care of a pet cat's needs. They were also provided with additional computational thinking activities, such as using abstraction and decomposition in an activity from Google's computational thinking course to figure out how many guesses it would take to determine which species on Earth someone was thinking of. In another example, we used a sorting activity where students form groups and sort a bag of material anyway they like but must justify the method they used to sort the material. In the third week, students were introduced to coding concepts using pseudocode and then visual-based programming exercises through the Hour of Code (https://studio.code.org/courses).

The majority of the semester was devoted to using two visual programming languages, Scratch and LEGO Mindstorms EV3 Education robotics, due to their graphical approach to programming that can be simpler for those new to coding. These environments have "low floor, high ceiling" capabilities, which Grover and Pea (2013) describe as having a simple learning curve for beginners, but more functionality and options for advanced learners. In addition, these skills can be easily transferred into their own future elementary and middle school classrooms.

Using Scratch and robotics, students were introduced to programming with loops, conditionals, and variables through programming in visual programming

environments. While Scratch allows students to use programming concepts and view what they code through characters, stories, and games on their computers, robotics allows them to demonstrate similar programming skills while also learning about hardware, actuators, and sensors. At the end of the Scratch unit students were given an assignment to create an educational game or story using Scratch. Some examples of student work included lessons and games using Scratch to teach mathematics, history, geography, and even British slang. One student created a mathematics race, in which the faster you answer math questions correctly, the faster your sprite reaches the finish line. As the levels go on, the opponent's speed increases, prompting the player to answer the questions at an even quicker speed.

In the robotics weeks of the course, students first built the robot and then engaged in programming challenges, such as creating a mini-version of a driverless car, where the robot can spot and react to the changing colors on a 'traffic light' in addition to detecting and responding to obstacles.

We also wanted to introduce text-based programming to the students, as most programming languages are text-based and we did not want them to later fear or shy away from text-based code. In our first semester offering the course, we used VPython as it is based on the Python programming language, which is a good first programming language to learn, and also has a 3D interactive modeling component where students can visualize what they are coding in three dimensions, and it can be useful when demonstrating physics concepts. In subsequent semesters, we used Python rather than VPython to expose students to text-based programming with similar concepts they already used in Scratch and robotics, without the need to teach additional concepts as it was many of their first time exposed to text-based programming.

In the subsequent three semesters, we also required students to read some sections of the book "Stuck in the Shallow End," which discusses why few minorities enter into CS and is also required reading for CS teachers at the public high schools in our area. This led to discussions on equity, and our female and minority students shared how they identified with the assigned reading as they were directed away from STEM in their own classroom experiences as children and provided with other activities instead.

While class exercises generally consisted of multidisciplinary teams of students, a final project of the semester grouped students in groups of 2-4 with others from similar disciplines and required each group to choose one of the platforms that we covered (primarily Scratch and Robotics), a lesson/topic of their choice, and a grade level if applicable. Students in education fields were intentionally grouped together in order for them to be able to focus on creating lesson plans that could be included in their own future classrooms. We also grouped CS students together, particularly the advanced ones, as we did not want advanced CS students to take over and do the coding if placed with students newer to programming.

Some of the education team projects included: presenting a lesson teaching distributive, commutative, and associative number properties with Scratch targeting 6th grade, creating an interactive quiz/game teaching algebra to middle school students with Scratch, and creating velocity scenarios with questions which their future 8th grade students would need to answer through hands-on programming and data logging with the robots. With the computer science groups, some example of projects included creating a project on teaching loops with Scratch, playing Tic Tac Toe using artificial intelligence through Scratch, and a garbage collector robot with LEGO Mindstorms.

4 Results

4.1 Survey Results

We conducted pre- and post-surveys at the beginning and end of each semester. Out of the 68 students who took the course, 48 students completed both a pre- and a post-survey (14 in the first semester, 13 in the second, 12 in the third semester, and 9 in the fourth semester). Of those 48 students, their majors included: 13 (27%) Education students, 20 (42%) Computer Science students, 2 (4%) Biology students, and 4 (8%) undecided majors. The remaining 9 students were in Accounting, Business Management, Communication, Economics, Graphic Design, Human Resource Development, Interdisciplinary Studies, Marketing, and Mathematics. Note that one education student who completed the survey was a current teacher while the rest were pre-service teachers.

Using a 5-point Likert scale ranging from 1 = 'Strongly Disagree' to 5 = 'Strongly Agree', we asked students questions pertaining to their confidence in computing and coding skills. Students' perception of their ability to write code and their understanding of how computer scientists approach problems significantly increased from the beginning and end of the semester (see Table 1). While their confidence in learning computer science concepts did not increase significantly, the mean values for the pre- and post-survey were high. Furthermore, education students' understanding of computing concepts well enough to incorporate them into their future classrooms increased from the beginning to end of the course.

Table 1. Pre- and Post-test Means for Students in CS0 (n=48)

CT Statement	Paired t (df)	Mean Pre (S. D.)	Mean Post (S. D.)
Significant			
I have the ability to write code for a computer program[a]	t(46)=-4.40***	3.26 (1.34)	4.19 (0.96)
I understand how computer scientists approach problems	t(47)=-2.40*	3.83 (1.02)	4.23 (0.93)
I understand computing or technology concepts well enough to incorporate them in my future classroom[b]	t(12)= -3.49**	3.00 (1.15)	3.92 (0.76)
Not Significant			
I am confident that I can learn computer science concepts	t(47)=-1.07	4.31 (0.62)	4.46 (0.80)

*p<.05; **p<.01; ***p<.0001

[a] The wording on this question changed from the first semester to subsequent semesters. Furthermore, one student left this statement blank.

[b] Only educators' responses are reported here (n=13).

In addition to reporting on the gains of the students, we compared computer science and education students as the majority of our students were from these two disciplines. A multivariate Wilk's Lambda test was conducted examining the three measures for education and computer science students, and these measures were found to be significantly different in the pre-test (Wilk's Lambda=0.584, F(3, 29)=6.89, p=0.0012) though not significantly different in the post-test (Wilk's Lambda=0.933, F(3, 29)=0.69, p=0.5663). We therefore conducted univariate t-tests (see Table 2) to compare each of the measures for computer science and education students and found that there were significant differences between computer science students and education students in their ratings for each item in the pre-survey. Education students rated each of these items significantly lower than computer science students in the beginning of the semester. However, in the post-survey there were no significant differences. This implies that by the end of the semester, education students felt as comfortable with coding and computing as the computer science majors did.

Table 2. Comparison of Education (n=13) vs. Computer Science Students (n=20)

CT Statement	**Major**	**Pre-Test**		**Post-Test**	
		Mean (S. D.)	***t* values**	**Mean (S. D.)**	***t* values**
I have the ability to write code for a computer program	CS	4.20 (1.11)	*t(31)=4.59***	4.25 (0.97)	t(31)=-0.19
	ED	2.38 (1.12)		4.31 (0.63)	
I understand how computer scientists approach problems	CS	4.40 (0.88)	*t(31)= 3.12**	4.35 (0.99)	t(31)=-0.36
	ED	3.46 (0.78)		4.46 (0.66)	
I am confident that I can learn computer science concepts	CS	4.60 (0.60)	*t(31)= 2.85**	4.60 (0.94)	t(31)=0.69
	ED	4.00 (0.58)		4.38 (0.77)	

*p<.01; **p<.0001

After the first semester of the course, we created a Computational Thinking scale used in subsequent semesters of the course for determining students' self-efficacy in CT.

- I am able to break a complex problem into smaller, more manageable parts or components so that it can be solved using a computer.
- I am able to manipulate a system's variables or components to achieve a desired result.
- I am able to modify existing computer code to complete small tasks in subject areas I am familiar with.
- I am able to create computer code to complete small tasks in subject areas I am familiar with.
- I can analyze or interpret a program's output or data.
- I understand how computational skills/tools could be applied to a variety of topics.
- I understand how computers can be programmed to develop solutions to problems.
- I am confident in my ability to use computational thinking to understand or analyze problems.

We conducted a factor analysis which showed the items loading as a single factor with factor loadings greater than .70 (three items were removed as they had factor loadings lower than .70). This new CT scale had a high level of reliability with a Cronbach's Alpha of .94 on the pre-test and .96 on the post-test.

Using a paired t-test on the CT scale comparing students' perceived computational thinking skills at the beginning and end of the semester, we found that student's self-efficacy in CT improved from the beginning ($M=3.44$, $SD=0.96$) to end ($M=4.03$, $SD=0.88$) of the course, $t(31)=-3.32$, $p=0.002$.

4.2 Focus Group Results

We also conducted focus groups both during Spring 2018 and Fall 2019 semesters. The spring focus group contained four out of six of the education students enrolled in the course and who were able to attend, and the Fall focus group had all four educations students enrolled in the course at that time (note that one student later changed majors to CS). The transcribed data revealed that the students were happy they took the course and felt block-based coding helped them understand CT. While some of the education students had been introduced to CT modules in other courses in their education program, they felt that exposure to this foundational course helped bring into perspective what they saw in the other courses. In addition, they reported that having more than one programming environment (Scratch and Robotics) helped build their confidence by showing them that the fundamental algorithms are still the same even though the programming environment may look different.

After taking the course, one student mentioned starting a robotics club in her future school or creating a computer science class for girls only. A second student said that this course motivated the student to be a better teacher since it provided ideas beyond using chalk and a board. Another student said that while s/he does not want to

teach computer science, this class taught him/her how to be a better science and math teacher through the incorporation of computer science into those classes.

4.3 Final Project Results

Similar to our survey results, which showed no differences in students confidence in their computing abilities when compared with CS students at the end of the semester, there were no significant differences found in the final project grades for education (M=93.14, SD=8.07) and computer science (M=85.59, SD=16.72) students; $t(41)= -1.77$, $p=0.0843$. While the overall average grade for student projects was 85.42 out of 100, the average for education students was above that at 93.14. By the end of the course, education students were able to master the material.

Each student had to answer questions individually based on their project as well. While variables were required in the project, a couple of education students used the term variable incorrectly. For example, when asked "What variables did you use in your program?" one student wrote about the 19 sprites she used, another education student wrote about creating voice recordings, and another education group did not use variables in their LEGO Mindstorms project.

In their project write-ups, students were also asked about different problems or ideas they had for the future. While many of the education students did have ideas for projects for math, science, English, and history, a couple of students mentioned how they as teachers would create Scratch projects for the students to use, but they did not mention requiring their future students to create similar programs.

5 Discussion

While previous studies modified existing courses for pre-service teachers to include coding and computational thinking (Adler & Kim, 2018; Jaipal-Jamani & Angeli, 2017; Kim et al., 2018; Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014), Mason and Rich (2019) recommend a discrete course to prepare teachers in CT and coding or for it to be integrated across the curriculum. Our work is part of a larger work that does both. In this paper we describe the inclusion of a new 15-week foundational computer science course for pre-service elementary and middle school STEM teachers. Similar to CS0 courses (Cliburn, 2006; M Rizvi & Humphries, 2012; Uludag et al., 2011), which are generally used for early computer science students, we designed a computer science course which focused on two visual programming environments, Scratch and LEGO Mindstorms, in order to prepare our pre-service teachers to use computational thinking in their future classrooms. Using two platforms in the same course allowed preservice-teachers to see that though a new programming environment may look different, the concepts are still the same.

While our course was developed for pre-service STEM teachers, we opened the course to all interested students. Students' self-efficacy in CT increased significantly from the beginning to end of the course. While it is not surprising that in the beginning of the semester education students did not feel as confident in their ability to learn computer science and use coding as the computer science students, we found that at the end of the semester there were no longer any differences. This implies that after taking the course, education students felt as comfortable with CT and coding as students who were CS majors. In fact, we had one education student switch majors after the course to CS. Previous studies on multidisciplinary collaboration have had CS majors and non-CS majors work on interdisciplinary projects where CS students are working on the

coding, and in those cases CS students perform better on computing self-assessments than non-CS majors (Pulimood, Pearson, & Bates, 2016; Way & Whidden, 2014), our findings show that when CS majors and education students are studying the same material in the same course non-CS students were able to perform as well in coding. Like Cliburn (2006), we found that CS0 worked well for non-computer science majors, however, unlike Cliburn (2006), whose goal was to recruit students into CS1, our primary goal is for our future teachers to use computer science in their future classrooms. Papadakis and Kalogiannakis (2019), similarly, found promising results when using Scratch in a semester-long course for pre-service Kindergarten teachers.

Therefore, to answer our first two research questions, we found that including a CS0 course into a curriculum for pre-service teachers did improve students' self-efficacy in CT, and education students' confidence in their CT ability was similar to CS students by the end of the course.

Our third research question asked whether education students would be prepared to use CT in their future classrooms after taking the course. We found that while students were more confident in their own coding skills at the end of the semester and had many ideas for implementing it in the future, as we saw from the focus group, we noticed during the final project that some students were not able to articulate how what they were currently learning could be used in their future classrooms. Several students mentioned creating coding projects for their students to use, but they did not discuss their students coding the activities on their own. Therefore, we plan to incorporate more specific examples in future iterations of the course on how the content they are learning can be used or modified to create relevant lessons in their own classrooms. We plan to emphasize that the goal of the final project is not only for them to incorporate

coding into a lesson, but to impart similar methods into their teaching so that their future students can use similar coding skills in their own projects.

Just as CS programs have been incorporating CS0 courses into their curriculum to better prepare computer science students with fundamental skills they will need to be successful in their later programming courses (Doyle, 2005; M Rizvi & Humphries, 2012), we need to provide our future educators with a complete CS course in order for them to have the basic skills they need to be successful at incorporating CT into their future classrooms. We recommend that pre-service teacher programs include an introductory computer science course that introduces future teachers to CT and coding. Future educators should be required to take this course to better prepare them with the foundation they need when they will hopefully come across CT modules in their later content and pedagogy courses, and for them to have the skills to use CT and coding in their future classrooms as educators. This will produce numerous K-8 students who will be exposed to computational thinking and coding at a young age. In this paper, we outline the steps we took to create this course in the hope that more pre-service teacher programs include CS courses in their curricula.

While the Computer Science for All course will become a requirement for our STEM K-8 teacher program, we plan to show this course to advisors in all of our education programs so that they can encourage their students to take this course as well. Pre-service teachers in every area can benefit by using coding and computational thinking in their future classrooms, whether they teach STEM or Social Studies (Güven & Gulbahar, 2020) and Language Arts (Wolz, Stone, Pearson, Pulimood, & Switzer, 2011). Furthermore, we plan to modify the course to include more topics on data, which is relevant for the social sciences. While one of our education teams used the robotics' data logging feature in their final project, we would like to update the course

to have all students work with data. Requiring more learning on data, particularly with practical topics directed for K-8, can enhance our classroom learning. While the Computer Science for All course was developed for pre-service teachers, a couple of in-service teachers took the course as well and future work will examine using Computer Science for All to train in-service teachers.

5.1 Lessons Learned

Below we outline lessons learned with possible solutions for future implementation of a CS0 for Educators:

5.1.1 Difficulty with Text-Based Programming

Our first semester used VPython to introduce text-based programming after students used Scratch and robotics. We noticed that students in the course had the easiest and fastest time learning Scratch and the most difficulty with coding with VPython. Scratch has many tutorials built in for students to easily learn how to accomplish many tasks. Some reasons for their difficulty with VPython may include (1) text-based programming is more difficult than drop-and-drop languages, (2) when teaching VPython we needed to introduce additional elements such as the 3D environment (X, Y, and Z axis) in addition to the inclusion of some physics concepts such as vectors. Therefore, we decided in subsequent semesters to introduce students to Python rather than VPython, thereby focusing only on a text-based language and using the concepts they learned throughout the semester without additional components. Some studies have reported difficulties when transferring concepts between block-based to more text-based approaches (Chetty & Barlow-Jones, 2012; Powers et al., 2007; Weintrop & Wilensky, 2016) and others report that using a visual-based approach, rather than a text-based one, can lead to improved learning and level of interest in future computing

courses (Weintrop & Wilensky, 2017). Therefore, we decided to shorten the length of time we focused on text-based programming in general in order to ensure that the majority of the course dealt with visual-based programming for their first exposure to computer science.

5.1.2 Varying Levels of Student Ability

Despite students being paired up with multidisciplinary teams for in-class exercises, some groups were able to advance and master exercises much more quickly than others. There were two solutions used to address the varying skills of the students in the course. The first was additional challenges placed in each in-class exercise so that those groups who were faster always had additional work they could do while the others were completing the main tasks. The second was the inclusion of a computer science student helper in the course. This student was considered a kind of peer leader, who was an advanced computer science student and attended each hands-on coding class to assist the instructor by also walking around the room and helping answer questions students may have with their specific code. This helped groups that needed additional support catch up by having an extra person available around the classroom at all times to assist students with their code. Studies on in-service teachers and computational thinking exercises have found differentiated assignments and peer mentoring helpful in elementary school classrooms (Israel, Pearson, Tapia, Wherfel, & Reese, 2015). By using these techniques in the classroom, we are demonstrating methods the pre-service teachers can try when integrating computing in their future classrooms.

5.1.3 Difficulty with Programming Terminology

As shown from their final projects, while the education students performed well, they had some confusion regarding understanding programming keywords, such as variables.

We also noticed them struggling with the term conditionals. While CS students were more familiar with terminology from previous programming experience or courses, education students rely on other disciplines such as learning about variables in math and science and conditionals from English Language Arts which causes some confusion at first. Grover et al. (2015) used a CT curriculum which included getting students comfortable with CS vocabulary through a "word wall" where terms and definitions were listed both on a course tab and pasted on the wall. Future CS0 implementation can reinforce programming vocabulary to ensure that the education students can use programming jargon correctly.

6 Conclusion

This study discusses the inclusion of an introductory computer science CS0 course placed in a curriculum for future elementary and middle school science and math teachers. Using visual programming languages, pre-service teachers were able to create programs and develop ideas for inclusion in their own future classrooms. Results demonstrate that students' self-efficacy in their CT skills increased from the beginning to end of the semester in addition to education students' understanding of computing concepts they can incorporate into their future classrooms. Furthermore, while at the beginning of the semester education students rated their coding and confidence in computing skills lower than computer science students in the course, by the end of the semester there were no perceived or actual coding differences. In addition, towards the end of the semester they had many ideas for including coding into their future careers through the creation of a robotic clubs, a female-only computer science course, and incorporating computational thinking in their future science and math classrooms.

Acknowledgements

OMITTED FOR BLIND REVIEW

References

Adler, R. F., & Kim, H. (2018). Enhancing future K-8 teachers' computational thinking skills through modeling and simulations. *Education and Information Technologies*, *23*(4), 1501–1514. https://doi.org/10.1007/s10639-017-9675-1

Agarwal, K. K., & Agarwal, A. (2006). Simply Python for CS0. *J. Comput. Sci. Coll.*, *21*(4), 162–170.

Anewalt, K. (2008). Making CS0 fun: an active learning approach using toys, games and Alice. *J. Comput. Sci. Coll.*, *23*(3), 98–105.

Berland, M., & Wilensky, U. (2015). Comparing Virtual and Physical Robotics Environments for Supporting Complex Systems and Computational Thinking. *Journal of Science Education and Technology*, *24*(5), 628–647. https://doi.org/10.1007/s10956-015-9552-x

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157. https://doi.org/https://doi.org/10.1016/j.compedu.2013.10.020

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *American Education Researcher Association*. Vancouver, Canada. Retrieved from https://dam-prod.media.mit.edu/x/files/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Cetin, I. (2016). Preservice Teachers' Introduction to Computing: Exploring Utilization of Scratch. *Journal of Educational Computing Research*, *54*(7), 997–1021. https://doi.org/10.1177/0735633116642774

Chetty, J., & Barlow-Jones, G. (2012). Bridging the gap: the role of mediated transfer for computer programming.

Cliburn, D. C. (2006). A CS0 course for the liberal arts. *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. Houston, Texas,

USA: ACM. https://doi.org/10.1145/1121341.1121368

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). Computational thinking: a guide for teachers. Retrieved from https://community.computingatschool.org.uk/resources/2324

Doyle, J. K. (2005). Improving performance and retention in CS1. *J. Comput. Sci. Coll.*, *21*(1), 11–18.

Google Inc. & Gallup Inc. (2016a). Diversity Gaps in Computer Science: Exploring the Underrepresentation of Girls, Blacks and Hispanics. Retrieved July 1, 2019, from http://services.google.com/fh/files/misc/diversity-gaps-in-computer-science-report.pdf

Google Inc. & Gallup Inc. (2016b). Trends in the state of computer science in U.S. K-12 schools. Retrieved July 1, 2019, from http://goo.gl/j291E0

Grabowski, L., & Brazier, P. (2011). Robots, recruitment, and retention: Broadening participation through CS0. In *Frontiers in Education Conference*. https://doi.org/10.1109/FIE.2011.6142918

Grover, S., & Pea, R. (2013). Computational Thinking in K–12. *Educational Researcher*, *42*(1), 38–43. https://doi.org/doi:10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Güven, I., & Gulbahar, Y. (2020). Integrating Computational Thinking into Social Studies. *The Social Studies*, 1–15.

Ioannou, A., & Makridou, E. (2018). Exploring the potentials of educational robotics in the development of computational thinking: A summary of current research and practical proposal for future work. *Education and Information Technologies*, *23*(6), 2531–2544. https://doi.org/10.1007/s10639-018-9729-z

Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, *82*, 263–279. https://doi.org/https://doi.org/10.1016/j.compedu.2014.11.022

Jaipal-Jamani, K., & Angeli, C. (2017). Effect of Robotics on Elementary Preservice Teachers' Self-Efficacy, Science Learning, and Computational Thinking. *Journal of Science Education and Technology*, *26*(2), 175–192. https://doi.org/10.1007/s10956-016-9663-z

Johnson, R. B., & Onwuegbuzie, A. J. (2004). Mixed Methods Research: A Research Paradigm Whose Time Has Come. *Educational Researcher*, *33*(7), 14–26. Retrieved from http://www.jstor.org/stable/3700093

K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org

Kim, C., Yuan, J., Vasconcelos, L., Shin, M., & Hill, R. B. (2018). Debugging during block-based programming. *Instructional Science*, *46*(5), 767–787. https://doi.org/10.1007/s11251-018-9453-5

Malan, D. J., & Leitner, H. H. (2007). Scratch for Budding Computer Scientists. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (pp. 223–227). New York, NY, USA: ACM. https://doi.org/10.1145/1227310.1227388

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (pp. 367–371). Portland, OR, USA: ACM. https://doi.org/10.1145/1352135.1352260

Mason, S. L., & Rich, P. J. (2019). Preparing elementary school teachers to teach computing, coding, and computational thinking. *Contemporary Issues in Technology and Teacher Education*, *19*(4). Retrieved from https://www.citejournal.org/volume-19/issue-4-19/general/preparing-elementary-school-teachers-to-teach-computing-coding-and-computational-thinking

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2010). Learning computer science concepts with scratch. *Proceedings of the Sixth International Workshop on Computing Education Research*. Aarhus, Denmark: ACM. https://doi.org/10.1145/1839594.1839607

Mishra, P., Yadav, A., Henriksen, D., Kereluik, K., Terry, L., Fahnoe, C., & Terry, C. (2013). Rethinking Technology & Creativity in the 21st Century: Of Art & Algorithms. *TechTrends*, *57*(3), 10–14. https://doi.org/10.1007/s11528-013-0655-z

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*. Norfolk, Virginia, USA: ACM. https://doi.org/10.1145/971300.971328

Papadakis, S., & Kalogiannakis, M. (2019). Evaluating a course for teaching introductory programming with Scratch to pre-service kindergarten teachers. *International Journal of Technology Enhanced Learning*, *11*(3), 231–246.

Papadakis, S., & Orfanakis, V. (2016). The combined use of Lego Mindstorms NXT and App Inventor for teaching novice programmers. In *International Conference EduRobotics 2016* (pp. 193–204). Springer.

Pearce, J., & Nakazawa, M. (2008). The Funnel That Grew Our Cis Major in the Cs Desert. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 503–507). New York, NY, USA: ACM. https://doi.org/10.1145/1352135.1352304

Perrin, A., & Kumar, M. (2019). About three-in-ten US adults say they are 'almost constantly'online. *Pew Research Center*.

Powers, K., Ecott, S., & Hirshfield, L. M. (2007). Through the looking glass: teaching CS0 with Alice. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. Covington, Kentucky, USA: ACM. https://doi.org/10.1145/1227310.1227386

Pulimood, S. M., Pearson, K., & Bates, D. C. (2016). A study on the impact of multidisciplinary collaboration on computational thinking. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 30–35).

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: programming for all. *Commun. ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Rizvi, M, & Humphries, T. (2012). A Scratch-based CS0 course for at-risk computer science majors. In *2012 Frontiers in Education Conference Proceedings* (pp. 1–5). https://doi.org/10.1109/FIE.2012.6462491

Rizvi, Mona, Humphries, T., Major, D., Jones, M., & Lauzun, H. (2011). A CS0 course using Scratch. *J. Comput. Small Coll.*, *26*(3), 19–27.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017, November 1). Demystifying computational thinking. *Educational Research Review*. Elsevier Ltd. https://doi.org/10.1016/j.edurev.2017.09.003

Turbak, F., Sherman, M., Martin, F., Wolber, D., & Pokress, S. C. (2014). Events-first programming in APP inventor. *J. Comput. Sci. Coll.*, *29*(6), 81–89.

Uludag, S., Karakus, M., & Turner, S. W. (2011). Implementing IT0/CS0 with scratch, app inventor forandroid, and lego mindstorms. *Proceedings of the 2011 Conference on Information Technology Education*. West Point, New York, USA: ACM. https://doi.org/10.1145/2047594.2047645

Way, T., & Whidden, S. (2014). A loosely-coupled approach to interdisciplinary computer science education. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)* (p. 1). The Steering Committee of The World Congress in Computer Science, Computer ….

Weese, J., & Feldhausen, R. (2017). STEM Outreach: Assessing Computational Thinking and Problem Solving. In *2017 American Society for Engineering Education Annual Conference & Exposition (ASEE)*. Columbus, Ohio.

Weintrop, D., & Wilensky, U. (2016). Bringing blocks-based programming into high school computer science classrooms. In *Annual Meeting of the American Educational Research Association (AERA). Washington DC, USA*.

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1–25.

Wing, J. M. (2006). Computational thinking. *Commun. ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wolz, U., Stone, M., Pearson, K., Pulimood, S. M., & Switzer, M. (2011). Computational thinking and expository writing in the middle school. *ACM*

Transactions on Computing Education (TOCE), *11*(2), 1–22.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational Thinking in Elementary and Secondary Teacher Education. *ACM Transactions on Computing Education*, *14*(1), 1–16. https://doi.org/10.1145/2576872

Yadav, A., Stephenson, C., & Hong, H. (2017). Computational Thinking for Teacher Education. *Communications of the ACM*, *60*(4), 55–62.