Computer Science in Early Education

Jaydon Smith
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
smit9025@morris.umn.edu

ABSTRACT

Recently, research in the field of computer science education has been gaining attention. Computer science is being taught more at the college level than in high schools and middle schools, but should expand more within the latter. Compared to the mathematics and science field in high schools, there isn't a similar amount of coverage for computer science. A few issues come with trying to bring it to younger students: whether younger students can comprehend computer science at their level, what methods exist to teach younger students computer science, and the availability of teachers with the proper knowledge to teach these students. Recent research to test computer science comprehension with younger students has surfaced. I examine two research methods used on a younger audience ranged from five to twelve years old. One uses ScratchJr, and the other uses Scratch and metaphors to teach computer science concepts. Based on the results gathered from these methods, it can be inferred that there was a positive effect on the knowledge of the students who took part in these studies. For the issue of teachers having proper knowledge, a possible solution found was having a training camp that teaches more advanced material. The results from this study showed an increase in confidence toward the material for the teachers that participated.

Keywords

Computer Science, Computer Science Education, K12 Education, Scratch, Limitations of Teaching

1. INTRODUCTION

We currently live in the digital age that is moving on from the industrial age. This has led to a high demand in workers who are able to use computers and develop technology related to computers. More demand for computer science related workers requires more students to be taught computer science. In the United States, computer science is being taught more at the college and university level than at the high school level, unlike fields such as science and math. Early teaching of computer science would allow students to be better equipped for future courses in college and in their careers.

Senior Seminar Paper, October 2020 Morris, MN.

In Burgiel et al. [2], students who were able to take computer science classes in earlier education had higher grades in college computer science education. In order to teach computer science in early education, one must overcome several limitations. These limitations are lack of knowing what methods we can use to teach younger students computer science, and having available teachers who have the proper knowledge to teach these classes. Two methods will be introduced that have been tested and show promising results for computer science concepts comprehension. These methods use a form of the educational program Scratch to teach computer concepts to children. It's been found that through the use of a program like Scratch, children are able to have an easier time moving onto programming languages like Python and Java.

In Pérez-Marín et al. [8], the researchers used metaphors and Scratch to teach elementary students computer science. They collected data through the use of a pretest-posttest design. In Strawhacker and Bers [9], the researchers only used ScratchJr and collected data based on the Solve It tasks that they created in ScratchJr. Both methods found positive results for showing that the children participating in their experiments were able to learn and retain very basic computer science concepts.

For methods like these to be implemented into schools with a set curriculum, there needs to be more teachers who are able to confidently teach computer science. In Leyzberg and Moretti [6], a study was conducted that had current computer science teachers attend a week long professional development training camp for teaching them more advanced computer science knowledge. The results from this study found that the professional development had a positive impact on the teachers and that they felt more prepared to teach their students.

Section 2 goes over some background information of what Scratch and ScratchJr is, and how it is used to teach computer programming concepts. Section 3 details measuring student comprehension of computer science. Section 4 expands on the advantages that early computer science comprehension has for students and their future.

2. BACKGROUND

2.1 Scratch and ScratchJr

A crucial part of the methods explored in this paper is the use of the educational product Scratch and ScratchJr. Scratch and ScratchJr are visual programming languages that were created to help children learn how to code as seen

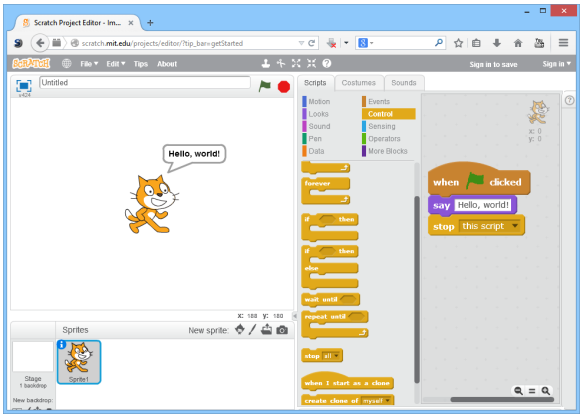

Figure 1: The Scratch interface.

Figure 2: The ScratchJr interface.

in Figures 1 and 2. Scratch has been used to allow students to work creatively, and be collaborative [4]. One of the key aspects that makes Scratch popular for educational purposes is its versatility. Students have the ability to create games, tutorials, greeting cards, music videos, and other projects through the use of Scratch.

Scratch works through piecing together command blocks to control sprites that move on a background (called the stage). These command blocks are simplified versions of code commands such as "move x steps" which is the same as changing the x coordinate of a sprite. The sprites are objects that store variables and scripts. There exist many different blocks that mirror programming language such as loop blocks that can be used within the program to repeat commands just like loops in programming languages. These "programmed" sprites allow the user the freedom to create what they want to make such as a game or music video. Scratch runs in a way that requires no compilation step or run mode, this allows users to click on any command fragment and see its action. Users then stay more engaged during the process of creating their projects. Through the use of this program users are able to visually learn computer science programming skills that are translatable to actual programming languages [7].

ScratchJr is an even more simplified version of Scratch that is aimed at children aged five to seven. This version of Scratch features less options and a simpler user interface. The command blocks have been replaced with image blocks, as seen in Figure 2 [3].

Both versions support problem-solving in a way that a goal can be identified and a solution can be created step-by-step and tried out. Not only can users explore problem-solving through their own creativity, puzzle projects can also be created for the users to solve. This is explored in Strawhacker and Bers's method using Solve Its.

3. COMPREHENSION

3.1 Computational Thinking

According to Pérez-Marín et al. [8], computational thinking is defined as "the skill of solving problems, designing systems, and understanding human behavior based on computer science concepts." By finding positive evidence for this skill in children, computer science education should be able to be brought to younger students in the same way that advanced math and science concepts have been brought to younger students. In order to test the possibility of computational thinking in younger students, multiple different programs and methods have been implemented in various studies. The program Scratch will be focused on.

Pérez-Marín et al. [8] designed an experiment that took 132 Spanish Elementary Students. These students were nine to twelve years old and lived in Spain. The researchers had them follow a methodology called MECOPROG. This methodology uses metaphors on top of programming blocks in Scratch to teach basic programming concepts seen in Figure 3. The researchers turned these concepts into metaphors: Loops with a hand mixer, conditionals with an intelligent fridge, input and output with mouth and rectum, and program, sequence, memory, and variables with recipe, pantry, and box.

This allows a lower abstraction level and helps students to learn through benefits found by using metaphors. The researchers covered a variety of concepts, practices, and perspectives using MECOPROG. The students where first introduced to a concept through metaphor, and then practiced with Scratch to learn the concept. The researchers developed a drag-and-drop visual interface companion app for the study that was called CompThink App. This app was used for the improvement of the children's computational thinking, and covered seven aspects: loops, algorithms, patterns, conditionals, steps, instructions, and automation. This app was used occasionally while working on the three blocks of MECOPROG. The students went through a six week course using the MECOPROG method.

In order to measure the students computational thinking, the researchers used a pretest-posttest design. Three tests where used called ROMT, CONT, and PCNT. Unfortunately, what these names stand for couldn't be found in the study. ROMT is a validated test for measuring children over age 10's computational thinking considered the ROM variable. CONT is a concept test that was created ad-hoc for the experiment that tests knowledge of programming, sequence, memory and variables, input and output, conditionals, and loops, considered the CON variable. PCNT is a test for measuring computational thinking similar to CONT, but for children under 10 years old, considered the PCN variable. An example of what a question from one of these tests would be "Which directions should a character

Figure 3: The three MECOPROG blocks. The top row is the concepts and the second row is sample script in Scratch relating to the concepts.

take to reach the goal?", with the directions listed individually in a sequence for the answer. The students took these tests before the six week course, participated in the course, then took the tests again. During the pre-tests, questions from the students weren't answered to avoid giving solutions to the post-test.

Table 1 shows the results from the PCNT, CONT, and ROMT tests that were taken by the students and the tests' variables. Using Spearman's rank correlation coefficient, the researchers found a significant correlation between pre and post-test for all variables. The researchers also used the Wilcoxon signed-rank test that is specifically for comparing two related samples, and found statistically significant difference between the pre and post-tests for all variables. For each variable measured in the tests, we can see that the score increased in both the median and mean in post-test. This result shows that the students' computational thinking scores increased from the course that they attended. This confirms that the skill of computational thinking in children exists and can be cultivated through a teaching method directed at improving computational thinking.

The researchers further discuss how they found that the children that participated in the study thought learning to program was engaging. All children payed attention throughout the course. There is basis here that children are already drawn to and enjoy computers, and by allowing them to learn more about the computers they are using, they are allowed to explore a field that interests them.

In Strawhacker and Bers [9], children from kindergarten through 2nd grade in the US took part in a six-week course using ScratchJr. A total of 57 students participated. A post-test was used to collect data based on the children's responses. Lessons introduced foundations ideas of computer science, and comprised of three modules: interactive collage, animated story, and interactive game. During the course the students were tasked to solve Solve It tasks that were developed by the researchers. These Solve It tasks were open-ended tasks that the students had to reverse engineer a program corresponding with a finished project projected on the screen. These tasks measured the ability of observation, memory, and reasoning, and stimulated the development of computational thinking. The researchers recorded errors made on the Solve Its to measure evidence of children's programming strategies. It was found that kindergarteners had an average of 3 errors per question, first graders had an average of 1.8 errors per question, and second graders had an average of 1.4 errors per question. There was another finding through Solve It Task 2 where two characters on screen took turns doing actions. Students in kindergarten found this concept to be too advanced because it hadn't been covered yet. A single solution to this Solve It was to use a Wait block which is similar to parallel programming. Only 13% of first graders used a Wait block, while 40% of second graders used a Wait block. This shows that a trend exists for the higher a grade, the less errors made, as well as students being able to form possible solutions based on new knowledge. With the domain of programming being a new area of study, this finding allows us to see that improvement over increasing grade is a viable pattern within the field of computer science.

It was found that all students where able to master simple Motion commands within ScratchJr. This shows that the children were able to draw on existing knowledge domains. Icon-based thought is a skill that is still developing within early childhood, an example being connecting the letter "A" with its sound. With some prior experience of learning the alphabet, being able to connect Motion blocks with what they would do is easy for the students to accomplish.

	PCN			CON			ROM		
	Mdn	M	SD	Mdn	M	SD	Mdn	M	SD
Pre	8.57	8.37	1.25	2.69	2.77	1.32	4.28	4.23	1.36
Post	9.28	8.99	1.05	5	5.08	1.59	4.64	4.77	1.56

Table 1: Pre and post-test PCN, CON, and ROM median, mean, and standard deviation.

4. ADVANTAGES

4.1 Future Education and Careers

Early programming introduction has been shown to help young children with cognitive skills, visual memory, and language skills [3]. Teaching computational thinking early also has the benefits of allowing students to manage uncertainty, assess problem difficulty, and use modularization [2]. Through early computer science education, students will be able to learn more advanced computer science concepts during their post-secondary education and will have an easier time with their computer science courses. These students will also bring more knowledge into their careers allowing the development of greater technologies.

In Burgiel et al., a study was formed to examine what affect early computer science courses had on student performance in college computer science. Using a sample of 2,871 introductory college computer science students from a wide variety of different colleges and universities in the US, a survey was conducted at the beginning of their introductory college science courses. After the course was over, the instructors entered final course grades. The survey consisted of questions asking about content and pedagogy in computer science courses taken before college. Sample mean grade in college computer science was 85.4 out of 100. Based on data gathered from the survey and course grades, the researchers used a two-level HLM (Hierarchical linear model) to predict

college computer science grades. Table 2 shows the correlation data based on the HLM. Variables observed in the study are listed on the left side of the table such as average parent education, whether students had help at home, if the student was male, other race which means non-white, etc.. The word "innovative" is based on whether a professor described their course as innovative and not on whether the course is actually scientifically innovative. "N" is the amount of classes or students per model. Variables under "Interactions" are based on how to variables interact with each other. Such as with the interaction between getting more help at home and increased frequency of coding practice is correlated with lower college grades.

On the top of the table the three models in the HLM are listed. Model 1 contains all significant control variables observed. Significant variables having a p-value that is less than .05. The p-value measures the probability that an observed difference could have occurred by random chance, a low p-value can tell us that the results are significant to us. Model 2 consists of all pedagogical variables of interest excluding all non-significant variables of interest. Model 3 is based on significant interactions. These models show the variance in student grades with model 1 being 9% of the variance, model 2 being 10% of the variance, and model 3 being 11% of the variance. The numbers to the right of the variables on the left is the data calculated based on the three models.

We can see that higher math SAT scores are correlated with better grades according to a 0.74 to 1.16 increase in grade points over 0. More coding practice in high school computer science courses correlated with higher grades in introductory college computer science, from 0.88 to 1.05 increase over 0. Higher frequencies of non-coding computer use corresponded to lower grades in college computer science from -0.67 to -0.65 decrease under 0.

	Model 1: Controls		Model 2: Main Effects		Model 3: Interactions	
	b	(se)	b	(se)	b	(se)
Intercept	77.55***	(1.53)	78.48***	(1.56)	80.57***	(1.71)
Student-level variables						
Avg. parent education	0.39	(0.22)	0.34	(0.22)	0.33	(0.22)
Help at home	0.38	(0.65)	0.33	(0.65)	0.46	(0.65)
Vocabulary	−0.04	(0.21)	−0.11	(0.21)	0.23	(0.25)
Male	−1.06*	(0.50)	−1.19*	(0.50)	−1.22*	(0.49)
Hispanic	−1.08	(0.73)	−1.11	(0.73)	−1.02	(0.73)
Black	−5.01***	(0.87)	−4.95***	(0.87)	−4.82***	(0.87)
Asian	−0.67	(0.57)	−0.60	(0.57)	−0.68	(0.56)
Other race	−1.59*	(0.75)	−1.47*	(0.74)	−1.61*	(0.74)
SAT Math/100	1.16***	(0.24)	1.07***	(0.24)	0.74**	(0.27)
Freshman	1.41**	(0.47)	1.13*	(0.47)	1.09*	(0.47)
Coding			0.88***	(0.24)	1.05***	(0.25)
Non-coding computer use			−0.67**	(0.22)	−0.65**	(0.22)
Class-level variable						
Innovative college CS	1.75	(0.91)	1.81*	(0.91)	−7.28*	(3.04)
Interactions						
Help at home X coding					−1.85**	(0.67)
Innovative X SAT math/100					1.46**	(0.47)
Innovative X vocabulary					−1.22**	(0.46)
N (classes)		177		177		177
N (students)		2871		2871		2871
Pseudo-R^2		9.0%		10.0%		11.0%

Note: Significant *p*-values indicated by * ($p \leq .05$), ** ($p \leq .01$), and *** ($p \leq .001$). Parentheses indicate standard errors.

Table 2: Table of Models Predicting College Computer Science Grade

Drawing from this, we can infer that having computer science classes within high school has the potential for students to do better in college computer science. Having more coding based classes has the potential for the students to do better in college computer science than they would just concept based classes in high school. The table also shows that there are some race and gender correlations that need to be further explored.

5. LIMITATIONS

5.1 Teachers

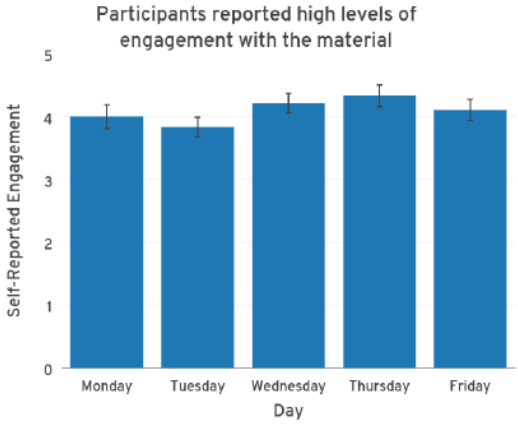

Figure 4: Levels of engagement with the material

A challenge that comes up with teaching early education students about computer science is having teachers available who are confident enough to teach these students. With this field of education being relatively new, there is a high demand for computer science teachers and not a large supply that is available [10]. Schools often address this problem by taking teachers who aren't knowledgeable about computer science and having them go through a training camp to learn what they need to know to teach computer science to students. This leads to teachers not feeling confident enough in their knowledge in the field to teach students.

In Leyzberg and Moretti [6] a study was conducted on teachers in the US who were first learning computer science to teach students. The researchers developed a week-long summer professional development workshop in order to further teach these teachers and help build their confidence in the subject. The workshop curriculum focused on material beyond the depth of AP curricula for computer science. This curriculum included recursion, analysis, machine language, and theoretical computer science topics. Twenty-five teachers participated in the workshop. What the researchers found is that the participants had very positive feedback about the workshop overall and that they came out of it feeling more confident about the material. There was a high amount of engagement from all participants that didn't decrease during the workshop according to figure 4. This can then show that in order for the demand to be met for teachers to teach early education computer science, workshops can be made to teach these teachers the material that they need in order to teach their students.

In Yadav et al. [10], the researchers discuss that with the push for computer science education around the globe such as CS for All in the US and Computing at School in the United Kingdom, there is a critical need for training a large amount of computing teachers. Currently within the United States, the computer science teacher field is flawed, this is causing a challenge for training and retaining computer sci-

ence teachers. A limited number of programs offer computer science teacher certifications as primary licensure areas. Those responsible for creating these programs don't have an understanding of what constitutes computer science. This then leaves the teachers who do go through these programs with little actual knowledge of the material that these teachers should be teaching. [5]

57% of teachers that teach computer science also teach other content areas. It is also shown that these teachers reported that computer science is being taught through the mathematics, business, or technology department in their schools. This requires that teachers who want to teach computer science have to meet certification requirements in another field.

The researchers created a study that observed twenty-four high school computer science teachers within the United States. This study had these teachers complete an online background questionnaire for collecting demographic information. They then held an interview with the researchers such that the researchers can better understand just what challenges these computer science teachers faced in the classroom. The interview was semi-structured, and was pilot tested on two computer science teachers for clarity and modification of the interview. The pilot interviews were not included in the data.

The researchers used a qualitative analysis software called Dedoose to analyze the data they collected. Dedoose is a web application for mixed methods research. This allowed the researchers to find eight conceptual themes related to the teachers' experiences in computer science classrooms. These themes were then categorized into three categories.

The first category being challenges of teaching in the classroom. Within the category, three themes are shown; content challenges, pedagogical challenges, and assessment challenges. The content challenge talks of how the teachers struggled with their limited content knowledge, and how they would be learning concepts alongside trying to teach their students. The pedagogical challenge talks of teachers concern of keeping students engaged and focused in the course, and meeting students' needs. The assessment challenge talks of how teachers have concern on evaluating student learning when not having proper tools or methods to evaluate student learning.

The second category is compounding factors that influence teaching. The first theme is lack of computer science teacher preparation. Teachers within the study discussed that there was a lack of preparation programs in their own background and that they learned to teach computer science mostly on their own. The second theme is isolation. The teachers talked about how unlike other content areas, computer science teachers don't have much of a support group of other computer science teachers with how few there are. The third theme was information technology challenges. Teachers discussed how they needed reliable support from the IT teams in their school district, and that they don't always have the resources for the latest technology.

The final category is supporting computer science teachers. The first theme being organized repository. The teachers discussed a need for better organization of online resources so that they can have an easier time finding what they need to be effective in the classroom. The second theme being community of practice. The teachers discussed a need for a community of computer science teachers in order to solve an issue of isolation. This would allow more computer science teachers to share ideas, tools, and approaches.

Through these three categories and their themes, we can identify the different kind of challenges that computer science teachers in the United States might be experiencing. Albeit, with a low sample size, not all computer science teachers will be experiencing these kinds of problems. With better certification programs, a way for computer science teachers to be able to connect with each other for support, a repository of teaching tools, assessment tools, and material, and professional computer science workshops, computer science teachers will be better able to teach computer science to their students.

6. CONCLUSION

The field of computer science is being explored through many different methods and has been gaining a lot of attention. Through the resources used in this paper, it is shown that young students do have the ability to develop the skill of computational thinking through the use of teaching with the programs Scratch and ScratchJr. With this proof, we should be moving toward a future where more computer science classes are being provided before college in order to allow students to acquire more background that fits into the digital age. Through availability of computer science courses in early education, students will then be able to proceed to post-secondary education with confidence that they will be able to do well in their computer science courses and build upon the knowledge that they have gained. These students would then have a better time getting into their careers in computer science and companies would have more experienced workers to handle current and further development of technology. There is currently a talent gap within the United States for computer science related careers because the growth of these jobs has been larger than the amount of declared computer science majors coming out of college each year [1].

By increasing the exposure of the computer science field to younger students, more interest will develop for the field and there will be growth in computer science majors. If more classes are to be offered before college, then teachers are going to be required to teach these classes. More programs need to be offered that teach upcoming computer science teachers what they need to know for these courses. All current computer science certification programs need to be redone in order to properly accommodate the knowledge needed for computer science teachers to properly teach their classes. There needs to be a push to separate computer science in schools to its own area so that computer science teachers don't have to also meet the requirements of another discipline in order to teach computer science. Through the use of developed workshops catered toward teaching teachers computer science content, these teachers will be available for these new classes while more computer science only teachers become available in the future.

7. ACKNOWLEDGMENTS

I want to thank Hussam Ghunaim, Elena Machkasova, KK Lamberty, and my peers for feedback and advice.

8. REFERENCES

[1] D. D. Bowman. Declining talent in computer related careers. *Journal of Academic Administration in Higher Education*, 14(1):1–4, 2018.

[2] H. Burgiel, P. M. Sadler, and G. Sonnert. The association of high school computer science content and pedagogy with students' success in college computer science. *ACM Transactions on Computing Education (TOCE)*, 20(2):1–21, 2020.

[3] L. P. Flannery, B. Silverman, E. R. Kazakoff, M. U. Bers, P. Bontá, and M. Resnick. Designing ScratchJr: Support for early childhood learning through computer programming. In *proceedings of the 12th international conference on interaction design and children*, pages 1–10, 2013.

[4] K. Hayat, N. A. Al-Shukaili, and K. Sultan. Alice in oman. *Education and Information Technologies*, 22(4):1553–1569, 2017.

[5] K. Lang, R. Galanos, J. Goode, D. Seehorn, F. Trees, P. Phillips, and C. Stephenson. Bugs in the system: Computer science teacher certification in the us. *The Computer Science Teachers Association and The Association for Computing Machinery*, 2013.

[6] D. Leyzberg and C. Moretti. Teaching cs to cs teachers: Addressing the need for advanced content in k-12 professional development. In *Proceedings of the 2017 ACM SIGCSE technical symposium on Computer Science Education*, pages 369–374, 2017.

[7] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–15, 2010.

[8] D. Pérez-Marín, R. Hijón-Neira, A. Bacelo, and C. Pizarro. Can computational thinking be improved by using a methodology based on metaphors and Scratch to teach computer programming to children? *Computers in Human Behavior*, 105:105849, 2020.

[9] A. Strawhacker and M. U. Bers. What they learn when they learn coding: investigating cognitive domains and computer programming knowledge in young children. *Educational Technology Research and Development*, 67(3):541–575, 2019.

[10] A. Yadav, S. Gretter, S. Hambrusch, and P. Sands. Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, 26(4):235–254, 2016.