IJCSES

Volume 5, No: 1
August 2021
ISSN 2513-8359

International Journal of Computer Science Education In Schools

Editors

Dr Filiz Kalelioglu

Dr Yasemin Allsop

www.ijcses.org

International Journal of Computer Science Education in Schools

August 2021, Vol 5, No 1

DOI: 10.21585/ijcses.v5i1

Table of Contents

International Journal of Computer Science Education in Schools, August 2021, Vol.5, No.1

ISSN 2513-8359

Examination of the Transitions between Modal Representations in Coding

Mustafa Serkan Abdüsselam[1]
Ebru Turan-Güntepe[1]

[1]Giresun University

DOI: 10.21585/ijcses.v5i1.125

Abstract

This study aims to determine the perceptions of undergraduates, who are receiving coding in a faculty of education, on modal representations employed in the teaching process and identify their transition skills between representations. The research used the quantitative research method, non-experimental design, and descriptive search models, calculating the obtained data frequencies by numerical analysis. The study was carried out with the participation of 58 undergraduates in the Computer and Instructional Technology Department of an education faculty in the 2018-2019 academic year. The representational skill-testing used in the study consists of 12 open-ended questions developed by the researchers. The reliability of the test was calculated as .96 with the Pearson product-moment correlation coefficient value. Transitions between the representation of mathematics, verbal, flowchart, and code were rankly listed in the test, which was applied in a single session. The obtained data were scored with a grading key and undergraduate achievement was assessed according to the transition between representations. The analysis has revealed that representation transition skills may differ from each other and that coding teaching, which takes into account these transition skills, should be carried out with flow chart, verbal, mathematical and ultimately code representations, respectively.

Keywords: coding, modal representation, perception, representational skill testing.

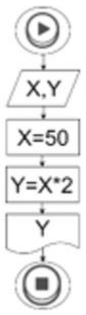
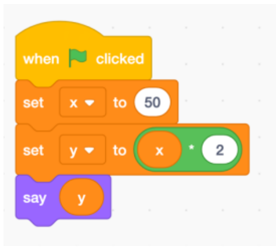
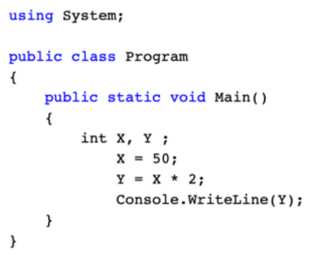
1. Introduction

The rising demand for coding teaching, combined with new skills required by our age, has made the lessons related to coding teaching both in schools (Sterritt, Hanna & Campbell, 2015; Williams et al., 2020) and on online learning platforms (Lau & Yuen, 2011; Çakıroğlu et al., 2016; Zinovieva et al., 2021) the focus of attention. Besides, coding has been introduced to curriculums from the preliminary education levels to accelerate countries' transition from consumer to producer through programs and software (Saeli et al., 2011; Demirer & Sak, 2016). Considering that computer programs do not exist without algorithms, algorithms in the learning process have become essential with the spread of computer applications (Levitin, 2012). This teaching process requires some essential skills such as program algorithm design and coding with a specific assembly code (Skiena, 2020). Moreover, the analysis skills of an algorithm will also help identify ways to make the most of the programming environment (Sedgewick & Flajolet, 2013; Gülbahar & Karal, 2018). During this process, learners are also expected to acquire basic skills such as "designing, writing, testing, debugging, and maintaining a source code of computer" program (Shadiev et al., 2014). Furthermore, it is key to coding learners to know the assembly code rules, and above all, to be able to predict solutions when encountered any problem (Renumol et al., 2009).

Any problem-solving process in coding should consists of understanding the problem, designing the solution frame, presenting the solution through algorithms prepared with cascaded schemes, implementing the planned solution, then testing the solution, fixing it if necessary (Eker, 2005; Levitin, 2012). Learners may use various modal representations during this process. Representations can be defined as graphics, visuals, diagrams, maps, tables, written and verbal languages (Pineda & Garza, 2002). Representations are also employed in coding teaching since they are effective for learners to experience an active learning process and to achieve substantial learning level (Seeger, 1996; Günel et al., 2009). Representations used in various fields may differ according to disciplines (Meij & Jong, 2006; Owens & Clements, 1998). Therefore, diverse representations like algorithms and flow charts are preferred in coding teaching and coding process, respectively. The use of flowcharts is vital to track the problem flow, and especially beginners in coding are recommended to prepare the program algorithm, subsequently writing the software (Eker, 2005; Kim & Lee, 2021). The last stage is to write software using assembly codes, according to the created algorithm (Ensmenger, 2016). Table 1 shows several examples of representations used by undergraduates in the algorithm and coding process during coding teaching.

Table 1. Algorithm and coding steps in the coding process

Algorithm			Coding	
Simple English	Flowchart	Pseudocode	Block-based	Text-based

Begin
Assign 50 to X
Multiply X by 2 and
Assign to Y
Write Y
End
		Begin
		X=50
		Y=X*2
		Write Y
		End
				`using System;`
				`public class Program`
				`{`
				`    public static void Main()`
				`    {`
				`        int X, Y ;`
				`            X = 50;`
				`            Y = X * 2;`
				`            Console.WriteLine(Y);`
				`    }`
				`}`

Representations have different content (Table 1) and each can be used to present information distinctively (Ainsworth, 2006). Using various representations during the learning process facilitates undergraduates' learning, summarizing content, increasing the retention, and recalling of any edited display (Bodur, 2010). Undergraduates may display and manipulate contents through texts, flowcharts, shapes, mathematical expressions, or codes during the coding teaching. The role and contribution of flowcharts among these representations in coding teaching have been previously reviewed (Shneiderman et al., 1977). These charts have been regarded as a tool in coding teaching (Nickerson, 1995) and have played a key role in solving the problems encountered during the code learning-teaching (Robins et al., 2003). Coding teaching is still perceived as a difficult process for students (Porter & Kalder, 2004; Baist & Pamungkas, 2017), while continuing to pose a major challenge for instructors (Barr & Guzdial, 2015; Sáez-López et al., 2016). The effectiveness of the used representations should be examined to overcome the difficulties encountered in coding teaching processes such as learners' adaptation to programming, creating a mental model of programming (Salleh et al.2018), understanding syntax and code structure (Salleh et al.2018), debugging (Sheard et al., 2009), grasping the program structure (Lahtinen et al., 2005), and building loops (Ginat, 2004). The coding teaching process, which is structured according to the effects of these representations, is believed to contribute to the learners' overcoming the difficulties they face and becoming successful in the process.

Studies in the literature suggest the use of different teaching approaches in coding teaching. One of these approaches is the coding approach with seven steps: understand the problem, devise a plan, compare the strategies, devise an algorithm, code the algorithm, identify and correct an error in a different code and prepare and code new algorithms

(Erümit et al., 2019). Additionally, it is known that the use of self-regulation strategies (Çakıroğlu et al., 2018) in the problem-solving process (Han & Kim, 2016; Yağcı, 2018; Abdüsselam et al., 2021) is also an essential part of the coding process. Although the literature studies on representations that we frequently utilize in the coding teaching process are limited, we have observed that existing studies focus on only a single representation, and that they are mostly related to flowcharts, and code representation (Gajewski, 2018; Kuljis & Baldwin, 2000). In fact, the coding teaching process requires learners to express the software they designed with flow charts, verbal expressions, and pseudo-codes in the program they use, as well as mathematical representations (Sedgewick & Flajolet, 2013). Again, the use of different modal representations and the creation of different schemas of the same information indicate the existence of transitions between modal representations (Stern et al., 2003; Marton & Tsui, 2004). These transitions also show that a single modal representation cannot be sufficient to represent some information or situations (Lemke, 1998). Also, the positive changes in the learning of students who prefer a method suitable for different learning styles (Safari & Hejazi, 2017) feature a detailed examination of the representation types used in coding teaching. The literature review has not yielded any positive result regarding which representations should be used in each step of the coding teaching process (Skiena, 2020). In this context, it is believed that the study will contribute a lot to the education of prospective teachers, who are expected to become programmers or coding teachers in the future, revealing the role and effectiveness of using different representations in coding teaching. Against this background, we aimed to determine undergraduates' (i.e., teacher candidates') perception of the representations in the coding teaching process and their transition skills between representations. To this end, the study sought to answer the following questions:

- How are the teacher candidates' transition skills regarding mathematical expressions' transition into code, flowchart, and verbal expression?
- How are the teacher candidates' transition skills regarding verbal expressions' transition into code, mathematical expressions, and flowchart?
- How are the teacher candidates' transition skills regarding flowchart's transition into a verbal, mathematical expression, and code?
- How are the teacher candidates' transition skills regarding codes' transition into a flowchart, verbal, and mathematical expression?
- How is the teacher candidates' ability to achieve transitions between representations?

2. Method

2.1 Research Desing

The research has been done by using quantitative research method, design that is non-experimental, and descriptive research and the data obtained from it have been analyzed by the researchers (McMillan & Schumacher, 2006). Frequency analysis was used in the numerical analysis of the data and the results were presented quantitatively.

2.2 Participant

This research was conducted with 58 sophomores at the Computer Education and Instructional Technology (CEIT) Department (in the Faculty of Education) of a state university in the Eastern Black Sea Region in the spring semester of the 2018-2019 academic year. These undergraduates, who have successfully completed the Programming Languages-I course and enrolled in the Programming Languages-II, have a basic knowledge of C# programming language. Participant undergraduates who take the Programming Languages-I course in the first semester of the same academic year have sufficient knowledge of programming concepts like variables, loops, and functions. During the Programming Languages-II course, they are expected to develop software through the C# programming fundamentals they have acquired.

2.3 Data Collection Tools

A representational skill testing (RST) with 12 open-ended questions was created during the development of the data collection tool in the research. These questions were selected from the questions contained in the documents that can be used in the teaching process of the C# programming language course of the CEIT department and among those that undergraduates described as easy, medium, and difficult during five years. Afterward, feedback on the questions were collected from three programming teaching experts and a representation expert. Based on the feedback from the experts, required modifications were made for the sake of the clarity, comprehensibility, and quality of the questions. To ensure the content validity of this test, the researchers consulted two experts on instructional technologies from the university. At the same time, they collected the opinion of an expert on measurement and evaluation to determine the test conformity. The pilot test was also carried out with volunteer junior undergraduates (N=6) who completed the programming courses of the CEIT department. The test with modal representations was refined and finalized according to the feedbacks from the experts' and undergraduates' observations during the pilot study. The data obtained from RST in the main study were scored independently by two researchers. The Pearson product-moment correlation coefficient value was calculated for the sake of the consistency between the two scorings to determine the reliability of the study. Pearson product-moment correlation coefficient value was calculated as .96 ($r = 0.963$, $p < 0.01$), and it was assessed that this value represents a significant, positive, and high-level relationship between the two scorings. We can thus assume that the obtained scoring is sufficiently reliable.

The developed test is planned to be applied in 100 minutes. The questions in the test were selected and designed in a way that requires analyzing a program based on a problem with different modal representations. The main purpose is to examine undergraduates' perceptions and use of various modal descriptions to solve a problem in the text-based coding process. Having been graded as easy, medium, and difficult, the RST questions cover mathematical, verbal, flowchart, and code representations. The questions' themes in modal representations, from easy to the complex are as follows: in math questions; basic calculus, arithmetic-geometric sequence, basic permutations, combinations & probability, in verbal questions; knowledge level, comprehension level, and analysis levels, in flowchart and code questions; simple flows, conditional flows, and loop structures. The implementation process of the research is schematized in Figure 1 accordingly.

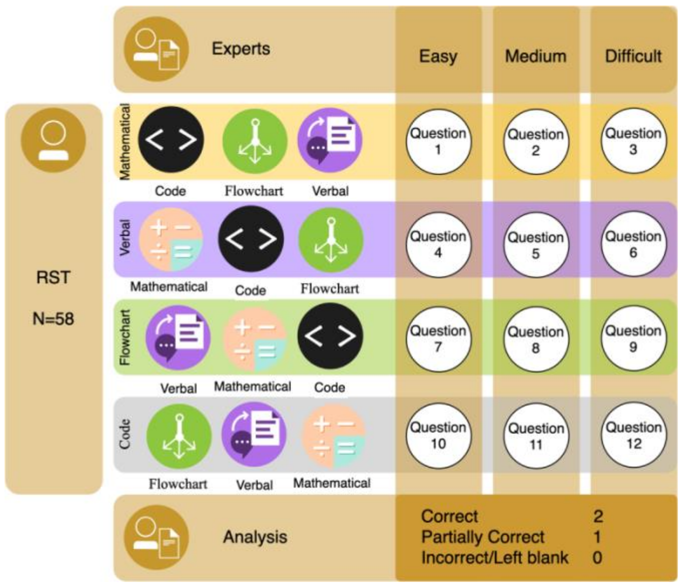
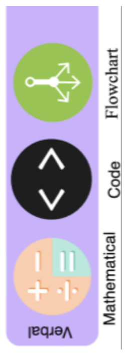


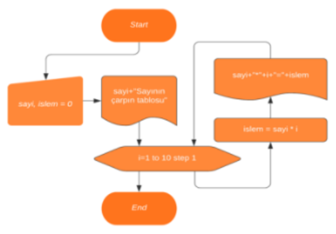
Figure 1. The research's implementation process and the grading key

RST consists of 12 questions as shown in Figure 1. RST consists of four-dimensional question representations: mathematical, verbal, flowchart, and code. In each dimension, a problem was defined with 3 questions classified from easy to difficult. The undergraduates were asked to define the algorithm and coding they will employ in the solution by using other representations. An example of RST questions is shown in Table 2.

Table 2. Examples of RST modal representation types and questions

Type	Sample Question
	Create the multiplication table of a number to be entered externally. Finalize the relevant question with C # assembly codes, flowchart, and mathematical representations.

Mathematical	Code	Flowchart

$\sum_{i=1}^{10} sayi * i$	`private void button1_Click(object sender, EventArgs e)`
	`{`
	`int sayi, islem;`
	`sayi = Convert.ToInt32(textBox1.Text);`
	`listBox1.Items.Add(sayi+" Sayısının çarpın tablosu");`
	`for (int i = 1; i <= 10; i++)`
	`{`
	`islem = sayi * i;`
	`listBox1.Items.Add(sayi+"*"+i+"="+islem);`
	`}`
	`}`
		 Start
		sayi, islem = 0
		sayi+"Sayının çarpın tablosu"
		sayi+"*"+i+"="+islem
		islem = sayi * i
		i=1 to 10 step 1
		End

2.4 Data Collection and Analysis

The open-ended questions in the RST were analyzed according to the grading key. Qualitative data were transformed into quantitative data and were analyzed through descriptive statistical processes. Researchers created the grading key, revising and amending, if necessary, both the questions and it, in line with the feedbacks from three programming teaching experts. Figure 1 shows the grading key. For the answers that were scored completely correctly, due attention was paid to whether the required answer was correctly established with the required field information, calculations, and connections, and then these answers are scored as "2". Partially correct answers that are scored as '1' are those in which the required field information, calculations, syntax error and connections are partially correct. The questions answered incorrectly, or left blank were scored as "0" in the grading key.

3. Results

3.1 RQ1, How are the teacher candidates' transition skills regarding mathematical expressions' transition into code, flowchart, and verbal expression?

The levels of teacher candidates' transition of mathematical expressions into code, flow chart, and verbal expression, as well as the number of the candidates at respective levels, are given in Chart 1.

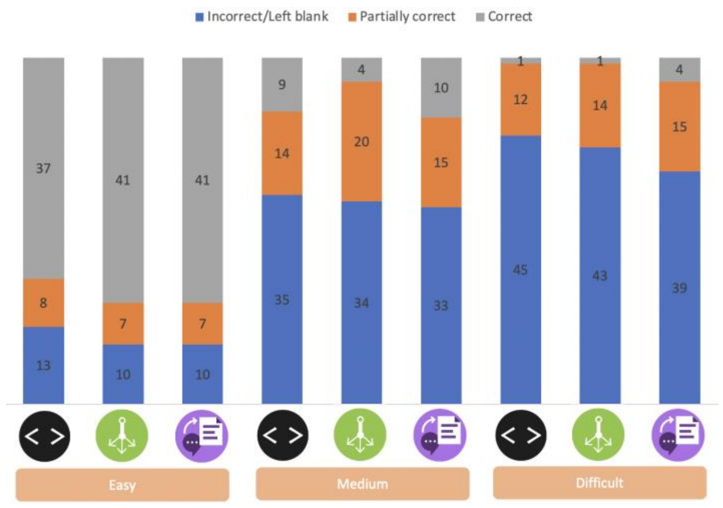


Chart 1. Transition frequencies of mathematical expressions into code, flowchart, and verbal expression

The candidate teachers were asked to present the code, flow chart, and verbal expressions of the relevant question using the mathematical expressions given in the questions. The examination of the answers given to Question 1, Question 2, and Question 3 prepared in easy, medium, and difficult levels, respectively, revealed that 37 candidate teachers in Question 1, 9 in Question 2 and 1 in Question 3 converted the mathematical expression into a code completely correctly. 41 teacher candidates in Question 1, 4 in Question 2, and 1 in Question 3 converted the mathematical expression into a flow chart completely correctly. 41 teacher candidates in Question 1, 10 in Question 2, and 4 in Question 3 were able to convert the mathematical expression into verbal expression completely correctly. Chart 1 shows the scoring distributions for each representation transition. Accordingly, for easy, medium, and difficult level questions, students' inability, or ability to realize the transitions completely and partially at each level shows a parallel distribution. Among the levels, the medium level seems closer to the difficult one. For Question 1, the candidate teachers converted the mathematical expression into flow chart, and verbal expression better than they converted it into the code. For Question 2 and 3, most of the candidate teachers were unable to convert or incorrectly converted the mathematical expression into code, flowchart, and verbal expression.

3.2 RQ2, How are the teacher candidates' transition skills regarding verbal expressions' transition into code, mathematical expressions, and flowchart?

The levels regarding the verbal expressions' transitions into code, mathematical expression, and flow chart, together with the number of the candidate teachers in relevant levels, are presented in Chart 2.

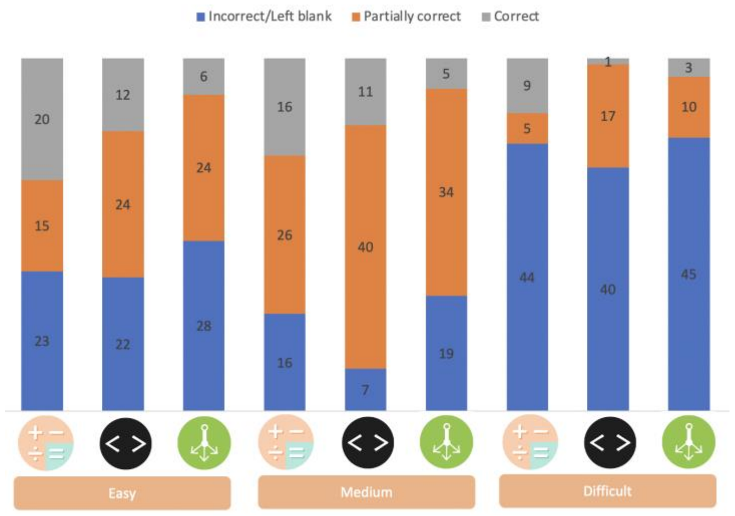


Chart 2. Transition frequencies of verbal expressions into code, mathematical expression, and flowchart

The candidate teachers were asked to present the code, flow chart and mathematical expressions of the relevant question using the verbal expressions given in the questions. The examination of the answers given to Questions 4, 5, and 6 prepared in easy, medium, and difficult levels, respectively, revealed that 12 candidate teachers in Question 4, 11 in Question 5 and 1 in Question 6 converted the verbal expression into a code completely correctly. Twenty teacher candidates in Question 4, 16 in Question 5, and 9 in Question 6 converted the verbal expression into a mathematical expression completely correctly. 6 teacher candidates in Question 4, 5 in Question 5, and 3 in Question 6 were able to convert the verbal expression into flow chart completely correctly. Chart 2 shows the scoring distributions for each representation transition. Accordingly, for easy, medium, and difficult level questions, students' inability, or ability to realize the transitions completely and partially at each level shows a parallel distribution. Among the levels, the easy and medium levels are similar in that undergraduates do the transitions partially. For Question 5, most of the teacher candidates answered the problem's code, mathematical expression, and flow chart partially correctly by using verbal expressions. In Question 4, most of the candidate teachers partially converted the verbal expression into code but could not convert it or converted it incorrectly into mathematical expression and flow chart. Most of the teachers could not convert the verbal expression into code, mathematical expression, and flow chart in Question 6.

3.3 RQ3, How are the teacher candidates' transition skills regarding flowchart's transition into a verbal, mathematical expression, and code?

The levels regarding the flow charts' transitions into a verbal and mathematical expression, and code, together with the number of the candidate teachers in relevant levels, are presented in Chart 3.

The candidate teachers were asked to present the code, mathematical and verbal expressions of the relevant question using the flow charts given in the questions. The examination of the answers given to Question 7, 8, and 9 prepared in easy, medium, and difficult levels, respectively, revealed that 50 candidate teachers in Question 7, 38 in Question 8, and 11 in Question 9 converted the flow charts into a code completely correctly. Fifty-four teacher candidates in Question 7, 41 in Question 8, and 23 in Question 9 converted the flow charts into a mathematical expression completely correctly. Fifty-two teacher candidates in Question 7, 46 in Question 8, and 24 in Question 9 were able to convert the flow charts completely correctly into verbal expressions. Chart 3 shows the scoring distributions for each representation transition. Accordingly, for easy, medium, and difficult level questions, students' inability, or ability to realize the transitions completely and partially at each level shows a parallel distribution, while showing a sequential one among the levels. For Question 8, most of the candidate teachers converted the flow charts into verbal expression better than they converted it into the mathematical expression and code. In Question 7, most of the teachers converted the relevant question into code, mathematical and verbal expression completely correctly by using flow charts. For Question 9, most of the candidate teachers transitioned the flow chart into mathematical and verbal expressions, while 29 of them partially converted it to code.

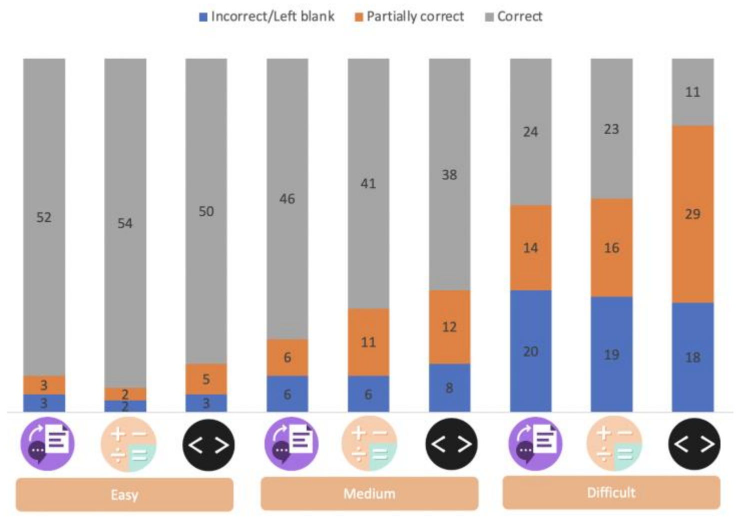
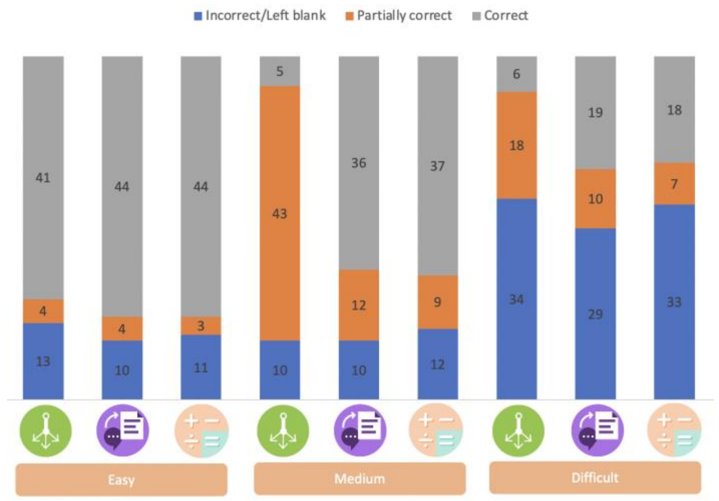


Chart 3. Flowcharts transitions to verbal, mathematical and coding expressions

3.4 RQ4, How are the teacher candidates' transition skills regarding codes' transition into a flowchart, verbal, and mathematical expression?

The levels regarding codes' transitions into a flow chart, mathematical and verbal expression, together with the number of the candidate teachers in relevant levels, are presented in Chart 4.

The candidate teachers were asked to present the flow chart, mathematical and verbal expressions of the relevant question using the questions' code. The examination of the answers given to Questions 10, 11, and 12 prepared in easy, medium, and difficult levels, respectively, revealed that 41 candidate teachers in Question 10, 5 in Question 11, and 6 in Question 12 transferred the code to the flow chart completely correctly. Fourty-four teacher candidates in Question 10, 37 in Question 11, and 18 in Question 12 converted the code into a mathematical expression completely correctly. Fourty-four teacher candidates in Question 10, 36 in Question 11, and 19 in Question 12 were able to convert the code completely correctly into verbal expressions. Chart 1 shows the scoring distributions for each representation transition. Accordingly, for easy, medium, and difficult level questions, students' inability, or ability to realize the transitions completely and partially at each level shows a parallel distribution (excluding flowchart's transition), while the flowchart transition between levels differentiates the structure. Most of the candidate teachers converted the code into a flow chart, mathematical and verbal expression in Question 10. For Question 11, most of the candidate teachers transitioned the code into mathematical and verbal expressions, while



43 of them partially converted it to a flow chart. In Question 12, most of the candidates were unable to convert or incorrectly converted the code into flow chart, mathematical and verbal expression.

Chart 4. Codes transitions to flow chart, verbal, and mathematical expressions

3.5 RQ5, How is the teacher candidates' ability to achieve transitions between representations?

The undergraduates' representations performed under the study were examined, and their ability to fully complete the requested transitions in 12 questions was analyzed. The obtained scores are listed in total in Table 3.

Table 3. Representations and transitions used in the research.

	Undergraduate's Representations				
Questions Representations	+−÷=	< >	🔀	💬	Total$_{\text{Analysis}}$
+−÷=	▨	81	46	55	182
< >	99	▨	52	99	250
🔀	118	99	▨	122	339
💬	45	24	14	▨	83
Total$_{\text{Rendering}}$	262	204	112	276	▨

Table 3 shows that undergraduates have obtained different scores after converting the question representations given to them into the other three ones. The examination of the question representations specified in the table horizontally shows that the representation transition in which the candidates are the most successful in the analysis is one of those made from the flow chart (339), according to the total scores calculated. Therefore, applying the questions to the students through flow charts in coding teaching can create an added value. Lastly, examining the representations, shown vertically in the table, created by the undergraduates of the faculty of education according to the questions showed that the undergraduates were the most successful in verbal transformation (276). In teaching coding, undergraduates should be asked to analyze verbally first because this is the mode of representation, they are most successful in.

4. Discussion

The research aims to reveal the relevance and effectiveness of different representations used in coding teaching and has tried to answer to how the transition skills between each representation employed in the said teaching process are transitioned into other representations. This study showed that candidate teachers' skills of expressing representations used in coding teaching may differ. The examination of mathematical expressions' transition to code, flowchart, and verbal expressions affirms that in terms of all transformations, while the candidate teachers converted easy questions, they either failed to transition or incorrectly transitioned the medium and difficult ones. The expectation that the implementation in the teaching process should progress from easy to difficult (Demirel, 2007) supports the study results. Therefore, we can assume that, as is the case in all teaching processes, simple questions that every candidate teacher can easily apprehend should be included in the curriculum, and then the desired content should be taught using more difficult questions or topics. In addition, Rizvi, Humphries, Major, Jones, and Lauzun (2011) discovered that although the number of undergraduates in computer science departments has increased in recent years, they have a weak knowledge of mathematics. Similarly, undergraduates of CEIT departments in Turkey have relatively low mathematics scores, and therefore they take Mathematics I and II courses before the programming course. In this sense, although the mathematical expressions in RST are included in the curriculum of Mathematics I and II, the fact that these expressions could not be converted or incorrectly converted to other transitions may be associated with the insufficient mathematics background of undergraduate students and the fact that the courses taken are not effective enough to improve their background.

The review of the candidate teachers' transition skills regarding converting verbal expressions into a code, mathematical expression, and flowchart has proved that significant hardship was encountered in all transitions and transitions at all relevant levels. However, the fact that candidate teachers have transitioned the problems presented by a verbal expression into the other ones can be explained by the fact that the transitions with verbal expressions in coding teaching were not solved as expected. Besides, according to Table 3, the undergraduates' presentation of the problems represented with verbal expression, their daily communication representation, more successfully than transitions in other representations reveals that the initial representation they will use in programming teaching should be verbal expressions. In such cases, Lemke (1993, 1998) specifies that students should be assisted to establish appropriate verbal connections with mathematical expressions, diagrams, graphics, and all the other representations. Thus, candidate teachers would be able to think beyond the box and have critical thinking with an effective questioning approach (Crafton et al., 2009; Kazimoglu et al., 2012).

The review of the candidate teachers' transition skills regarding converting flow charts into codes, verbal and mathematical expressions has revealed that there is a significant success in easy, medium, and difficult transition levels in terms of all transformations. Visualizing the coding teaching process that is perceived as complex by learners, thereby making it more apprehensible, can explain this phenomenon. In that vein, Hagevik, Beilfuss, and Dickerson (2006) stated that modal representations, which form a visual state of knowledge, simplify complex meanings, and organize individuals' cognitive processes by supporting them. It is still an expected phenomenon that teacher candidates are more likely to be unable to do or make mistakes with difficult questions when compared to the other two levels during the process of flow charts' transformations. Still, we assert that the transition to codes at this level can be performed rather less compared to the other transitions. In fact, it is remarkable that few candidate teachers have completely correctly answered the transformations in flow charts compared to other transformations, during the teaching-learning process, where the transition from flow charts to codes is often made in the coding teaching. Nonetheless, several studies in the literature suggest that coding should be realized following the creation of the flowchart (Ergin & İpek, 2017; Gajewski, 2018; Türker & Pala, 2020). Accordingly, we assert that a difficult question on flow charts is more distinctive compared to questions regarding the other transitions. Also, the examination of the candidate teachers' transition skills regarding the converting codes into flow charts, verbal and mathematical expressions has shown that learners experienced difficulties in these representation transitions, especially in flow chart transitions compared to the others (Chart 4). Among these transitions, we have found that candidate teachers had problems in transitions from flow chart to code and vice versa.

Remarkably, the transition from flow chart to verbal and mathematical expression and codes is at the forefront according to candidate teachers' ability to perform all transitions. Additionally, in these transitions, what candidates can do best in each representation again matches the same transition. In general, coding teaching requires candidate teachers to set up an algorithm and then convert it into coding. The lecturer may exploit verbal and mathematical expressions or flowcharts during the process of creating an algorithm. The research results showed that by presenting the modal representations in coding teaching, the learners understood the program most successfully and were able to re-transition it into the other representations. Accordingly, coding teaching should be started with flow charts designed in shapes related to the program. In this sense, the verbal representation is the most successful one for candidate teachers in the case of re-representing the curriculum. Therefore, presenting flowcharts to learners and having discussions with them about the program can be regarded as the second step of coding teaching. The next step is to describe the program to be taught through mathematical expressions according to the obtained results. The last step of this process is code representations following the mathematical presentation. As Gajewski (2018) stated, flowcharts are rather effective tools for coding teaching and play an essential bridging role in the algorithm creation and then in the coding execution. Also, Kuljis and Baldwin (2000) affirm that the formal representations used under the scope of a flow chart for learners who are new to coding are highly suitable for the coding teaching. According to Ramadhan (2000), the reason behind this is that supporting (learning) topics related to decision structures and loops in coding teaching with visual tools like shapes has a positive effect on the success. However, as Hu (2004) stated, theories and practical applications should be handled simultaneously in coding teaching. Similarly, this research has also shown that flowcharts are seized more easily by coding learners then the other representations and that transition to the other representations is more successful.

This study has been carried out for the use of text-based coding tools at the undergraduate level. In this context, we suggest examining the relevance and effectiveness of these representations during the use of block-based tools. We also recommend that the coding teaching process be structured according to the order of use of the mathematical, verbal, flowchart, and code representations specified in the research. Finally, future studies may examine relationships between various variables such as the effectiveness of the newly developed representation usage order, and learners' attitudes and their programming self-efficacy.

Acknowledgments

The research has received no financial support. The authors have made equal contribution to the research. The research was carried out following ethical guidelines, and Giresun University approved the research protocol.

References

Abdüsselam, M. S., Turan-Güntepe, E., & Durukan, Ü. G. (2021). Investigation of the Teaching Process of Programming in regards to Problem Solving Process and Perception of Self-Efficacy. *Journal of Bayburt Education Faculty, 16*(31), 149-173.

Ainsworth, S. (2006). DeFT: A conceptual framework for considering learning with multiple representations. *Learning and instruction*, *16*(3), 183-198. https://doi.org/10.1016/j.learninstruc.2006.03.001

Aslanyürek M., Korkmaz A., Büyükgöze S.B. & Gezgin D.M (2018) *Algorithm and Programming All Resolved Question Bank*, Efeakademi Publisher.

Baist, A., & Pamungkas, A. S. (2017). Analysis of Student Difficulties in Computer Programming. *VOLT: Jurnal Ilmiah Pendidikan Teknik Elektro*, *2*(2), 81-92. https://doi.org/10.30870/volt.v2i2.2211

Barr, V., & Guzdial, M. (2015). Advice on teaching CS, and the learnability of programming languages. *Communications of the ACM, 58*(3), 8–9.

Bodur, F. (2010). *Additives to learning visual elements in distance teaching textbooks: Evaluation of Anadolu University distance teaching student views*. *Eskişehir: Anadolu University*.

Çakıroğlu, Ü., Kokoç, M., Kol, E., & Turan, E. (2016). Exploring teaching programming online through web conferencing system: The lens of activity theory. *Journal of Educational Technology & Society*, *19*(4), 126-139.

Çakıroğlu, Ü, Er, B. Uğur, N., & Aydoğdu, E. (2018). Exploring the use of self-regulation strategies in programming with regard to learning styles. *International Journal of Computer Science Education in Schools*, *2*(2), 14-28. https://doi.org/10.21585/ijcses.v2i2.29

Clark, J. M., & Paivio, A. (1991). Dual coding theory and education. *Educational psychology review*, *3*(3), 149-210. https://doi.org/10.1007/BF01320076

Crafton, L., Brennan, M., and Silvers, P. (2009). Creating a critical multiliteracies curriculum: Repositioning art in the early childhood classroom. Making Meaning.

Crafton, L. K., Silvers, P., & Brennan, M. (2009). *Creating a critical multiliteracies curriculum: Repositioning art in the early childhood classroom. In Making meaning*. Springer.

Demirel, Ö. (2007). *Teaching principles and methods: The art of teaching*. Ankara: Pegem A Yayıncılık.

Demirer, V., & Sak, N. (2015). Information and communication technology (ICT) education in Turkey and changing roles of the ICT teachers. *The Journal of International Education, 2* (5), 434-448.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57-73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

Eker M. (2005), *Understanding the algorithm*, Niravan Yayınları, Ankara

Ensmenger, N. (2016). The multiple meanings of a flowchart. *Information & Culture*, *51*(3), 321-351. https://doi.org/10.7560/IC51302

Ergin, H., & İpek,J (2017). Collaborative Creative Problem-Solving Model in Programming Language Teaching: A Case Study. *Ege Journal of educational Technologies*, *1*(2), 135-148.

Erümit, K. A., Karal, H., Şahin, G., Aksoy, D. A., Aksoy, A., & Benzer, A. I. (2019). A model suggested for programming teaching: Programming in seven steps. *Education and Science, 44*(197), 155-183. http://dx.doi.org/10.15390/EB.2018.7678

Gajewski, R. R. (2018). Algorithms, Programming, Flowcharts and Flowgorithm. *E-Learning and Smart Learning Environment for the Preparation of New Generation Specialists*, 393-408.

Ginat, D. (2004). On novice loop boundaries and range conceptions. *Computer Science Education, 14*(3), 165-181.

Gülbahar, Y., & Karal, H. (2018). *Teaching programming from the theory to application [Kuramdan uygulamaya programlama öğretimi]*. Ankara: Pegem Akademi Yayıncılık.

Günel, M., Atila, M. E., & Buyukkasap, E. (2009). The impact of using multi modal representations within writing to learn activities on learning electricity unit at 6th grade. *Elementary Education Online*, *8*(1), 183-199. http://ilkogretim-online.org.tr/index.php/io/article/viewFile/1705/1541

Hagevik R, Beilfuss M, Dickerson D (2006). Multiple representation sin science education. Paper presented at the annual meeting of the National Association for Research in Science Teaching (NARST), April 3–6, San Francisco

Han, S. J., & Kim, S. S. (2016). The effects of app programing education using m-Bizmaker on creative problem solving ability. *The Journal of Korean Association of Computer Education*, *19*(6), 25-32.

Hu, M. (2004). Teaching novices programming with core language and dynamic visualisation. *Proceedings of the 17th NACCQ*, 94-103. Retrieved from https://www.citrenz.ac.nz/conferences/2004/hu.pdf

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science*, *9*, 522-531. https://doi.org/10.1016/j.procs.2012.04.056

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, *37*(2), 83-137. https://doi.org/10.1145/1089733.1089734

Kim, Y., & Lee, M. (2021). Development of an unfolding model of procedures for programming learning of novice programmers. *Computer Applications in Engineering Education*. 1-20. https://doi.org/10.1002/cae.22437

Kuljis, J., & Baldwin, L. P. (2000). Visualisation Techniques for Learning and Teaching Programming. *Journal of computing and information technology*, *8*(4), 285-291. https://doi.org/10.2498/cit.2000.04.03

Lahtinen, E., Ala-Mutka, K. & Jarvinen, H. (2005) A Study of Difficulties of Novice Programmers. *In Acm Sigcse Bulletin, ACM, 37*(3), 14-18.

Lau, W. W., & Yuen, A. H. (2011). Modelling programming performance: Beyond the influence of learner characteristics. *Computers & Education*, *57*(1), 1202-1213. https://doi.org/10.1016/j.compedu.2011.01.002

Law, K. M., Lee, V. C., & Yu, Y. T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, *55*(1), 218-228. https://doi.org/10.1016/j.compedu.2010.01.007

Lemke, J. L. (1993, December 17-18). *Multiplying meaning: Literacy in a multimedia world* [Conference presentation abstract]. Annual Meeting of the National Reading Conference, Charleston, SC, United States.

Lemke, J. (1998). *Multiplying meaning: Visual and verbal semiotics in scientific text*. London: Routledge.

Levitin, A. (2012). *Introduction to design and analysis of algorithm*. Pearson, Boston.

Marton, F., & Tsui, A. (2004). *Classroom discourse and the space of learning*. Mahwah, NJ: Erlbaum

McMillan, J. H., and Schumacher, S. (2006). *Research in Education: Evidence-based Inquiry*. Cape Town: Pearson.

Meij, J., & de Jong, T. (2006). Supporting students' learning with multiple representations in a dynamic simulation-based learning environment. *Learning and instruction*, *16*(3), 199-212. https://doi.org/10.1016/j.learninstruc.2006.03.007

Nickerson, J. V. (1995). *Visual programming*. New York, NY: New York University.

Owens, K. D., & Clements, M. K. (1998). Representations in spatial problem solving in the classroom. *The Journal of Mathematical Behavior*, *17*(2), 197-218. https://doi.org/10.1016/S0364-0213(99)80059-7

Pineda, L., & Garza, G. (2000). A model for multimodal reference resolution. *Computational Linguistics*, *26*(2), 139-193. https://doi.org/10.1162/089120100561665

Porter, R., & Calder, P. (2004, January). Patterns in learning to program: an experiment?. In *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30* (pp.241-246). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.2526&rep=rep1&type=pdf

Qian, Y., & Lehman, J. (2020). An Investigation of High School Students' Errors in Introductory Programming: A Data-Driven Approach. *Journal of Educational Computing Research*, *58*(5), 919-945. https://doi.org/10.1177/0735633119887508

Ramadhan, H. A. (2000). Programming by discovery. *Journal of Computer Assisted Learning*, *16*(1), 83-93. https://doi.org/10.1046/j.1365-2729.2000.00118.x

Renumol, V., Jayaprakash, S., & Janakiram, D. (2009). Classification of cognitive difficulties of students to learn computer programming. *Indian Institute of Technology, India*, *12*. http://dos.iitm.ac.in/publications/LabPapers/techRep2009-01.pdf

Rizvi, M., Humphries, T., Major, D., Jones, M., & Lauzun, H. (2011). A CS0 course using scratch. *Journal of Computing Sciences in Colleges*, *26*(3), 19-27.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, *13*(2), 137-172. https://doi.org/10.1076/csed.13.2.137.14200

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in Secondary school: A pedagogical content knowledge perspective. *Informatics in education*, *10*(1), 73-88. https://doi.org/10.15388/infedu.2011.06

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education, 97*,129–141.

Safari, E., & Hejazi, M. (2017). Learning styles and self-regulation: an associational study on high school students in iran. *Mediterranean Journal of Social Sciences, 8*(1), 463. https://doi.org/10.5901/mjss.2017.v8n1p463

Salleh, S. M., Shukur, Z., & Judi, H. M. (2018). Scaffolding model for efficient programming learning based on cognitive load theory. *Int. J. Pure Appl. Math*, *118*(7), 77-83. https://acadpubl.eu/jsi/2018-118-7-9/articles/7/10.pdf

Sedgewick, R., & Flajolet, P. (2013). *An introduction to the analysis of algorithms*. Pearson Education India.

Seeger, F. (1996). *Representations in the mathematics classroom: Reflections and constructions*. University of Helsinki.

Shadiev, R., Hwang, W. Y., Yeh, S. C., Yang, S. J., Wang, J. L., Han, L., & Hsu, G. L. (2014). Effects of unidirectional vs. reciprocal teaching strategies on web-based computer programming learning. *Journal of educational computing research*, *50*(1), 67-95. https://doi.org/10.2190/EC.50.1.d

Sheard, J., Simon, S., Hamilton, M., & Lönnberg, J. (2009, August). Analysis of research into the teaching and learning of programming. In *Proceedings of the fifth international workshop on Computing education research workshop* (pp. 93-104). https://doi.org/10.1145/1584322.1584334

Shneiderman, B., Mayer, R., McKay, D., & Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM*, *20*(6), 373-381. https://doi.org/10.1145/359605.359610

Skiena, S. S. (2020). *The algorithm design manual*. Springer International Publishing.

Stern, E., Aprea, C. & Ebner, G. E. (2003). Improving Cross-Content Transfer in Text Processing by Means of Active Graphical Representation. *Learning and Instruction, 13* (2),191-203. https://doi.org/10.1016/S0959-4752(02)00020-8

Sterritt, R., Hanna, P., & Campbell, J. (2015, March). Reintroducing programming to the school environment. In *INTED 2015-9th International Technology, Education and Development Conference* (pp. 7630-7631). International Academy of Technology, Education and Development. Madrid, Spain. Retrieved from http://library.iated.org/view/STERRITT2015REI

Türker, P. M., & Pala, F. K. (2020). The effect of algorithm education on students' computer programming self-efficacy perceptions and computational thinking skills. *International Journal of Computer Science Education in Schools*, *3*(3), 19-32.

Williams, H. E., Williams, S., & Kendall, K. (2020, June). CS in Schools: Developing a Sustainable Coding Programme in Australian Schools. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 321-327). https://doi.org/10.1145/3341525.3387422

Yağcı, M. (2018). A Study on Computational Thinking and High School Students' Computational Thinking Skill Levels. *International Online Journal of Educational Sciences,* 10(2), 81-96. doi:10.15345/iojes.2018.02.006

Zinovieva, I. S., Artemchuk, V. O., Iatsyshyn, A. V., Popov, O. O., Kovach, V. O., Iatsyshyn, A. V., ... & Radchenko, O. V. (2021, March). The use of online coding platforms as additional distance tools in programming education. In *Journal of Physics: Conference Series* (Vol. 1840, No. 1, p. 012029). IOP Publishing. https://doi.org/10.1088/1742-6596/1840/1/012029

International Journal of Computer Science Education in Schools, August 2021, Vol.5, No.1

ISSN 2513-8359

Examining Equity in an Elementary School Computer Science Intervention Using Component-Based Research

Kaitlyn A. Ferris[1]

Jeanne Century[1]

Huifang Zuo[1]

[1]University of Chicago

DOI: 10.21585/ijcses.v5i1.121

Abstract

This article reports on implementation of a problem-based learning intervention developed with the intention of finding time for computer science (CS) in the elementary school day. This study investigated differences in effects on students in particular socio-demographic groups using a quasi-experimental design. We first provide an overview of the perennial problem of group differences or "gaps" in student outcomes. Then we illustrate how, using component-based research (CBR), we moved beyond the question of whether the intervention worked, to focus on which *parts* of the intervention worked, for *whom,* and under what *conditions*. Using hierarchical linear modeling, this study draws from a sample of 16 elementary schools with 321 teachers and 5791 students in Broward County, Florida, the sixth largest school system in the United States. This study complements a previous paper (Century, Ferris, Zuo, 2020), which examined associations between intervention components and student outcomes by investigating how outcomes differ for students in different socio-demographic groups and whether the presence of particular intervention components amplify or reduce differences. Through CBR, our work illustrates that CS interventions which may appear to benefit students overall, may be less beneficial or even detrimental to particular groups.

Keywords: computer science, elementary, component-based research, equity, implementation research

1. Introduction

1.1 *Differences in Academic Outcomes by Group Exist and Persist*

Differences in academic outcomes between groups of learners are well-documented across racial, ethnic, and gender identities, and socioeconomic, disability, English proficiency, and migrant status (Celeste et al., 2019; Chmielewski, 2019; Gilmour, et al., 2019; Hung et al., 2020; Owens, 2018). Although limited publications report on group differences in computer science (CS) outcomes at the primary grade levels, these differences are clear in other subjects. In the US, the 2019 National Assessment of Educational Progress (NAEP) showed that White students scored 26 and 28 points higher than their Black peers in 4th grade and 8th grade reading assessments, respectively, and 25 and 32 points higher in 4th and 8th grade math assessments, respectively (NAEP, 2019). Group differences in achievement outcomes also exist between Hispanic and White students with 21 and 20 point differences for reading in 4th and 8th grades, respectively, and 18 and 24 point differences for 4th and 8th grade mathematics, respectively (NAEP, 2019)**.** These racial and ethnic group differences can be explained, in part, by differential access to high quality education and inequitable learning experiences. In the US, students who are Black or Hispanic are more likely to attend high-poverty schools (Orfield et al., 2016; Orfield & Lee, 2005) and be denied access to STEM and CS courses and inclusive pedagogy, which often results in segregated CS education and career pathways (Margolis et al., 2017). School poverty rates are also associated with the size of the difference in achievement by racial group and the rate at which these differences increase over time (Reardon et al., 2019).

Given the strong associations between achievement and school-level poverty, it is also important to consider achievement differences across income groups. One study found that for every $1,000 increase in family income, there was an associated 2.1% increase in students' math scores and a 3.6% increase in students' reading scores (Dahl & Lochner, 2012). Additionally, academic outcome differences between students of high- and low-SES groups tend to increase as students get older (Caro, 2009). Unsurprisingly, the 2018 Program for International Student Assessment indicates that students from low SES backgrounds lag far behind more advantaged students in their reading, mathematics, and science performance (OECD, 2020).

Disparities in academic achievement between gender groups also persist. For instance, only 19% of female students scored at a proficient level on the science NAEP test compared to 25% of male students (NAEP, 2019). Similar patterns were observed for scores on the NAEP mathematics test, with only 38% of female students scoring at or above proficiency, compared to 44% of males (NAEP, 2019). Differences also emerged based on English-language proficiency levels. Regardless of grade level, English Language Learners (ELLs) performed lower on reading and mathematics assessments compared to native English speakers, and such differences were particularly pronounced in domains that require more use of the English language, such as reading and writing (Abedi, 2002; Fry, 2007).

1.2 *Group Differences in Attitudes*

In STEM subjects, research on group differences in attitudes have had mixed results, and some group comparisons have yet to be investigated (e.g., ELLs and native English speakers). For instance, some studies indicate that Black students report lower science ability and mathematics beliefs compared to White students (Britner & Pajares, 2001; Pajares & Kranzler, 1995), whereas others have found that, compared with White students, Black students demonstrate higher mathematics attitudes (e.g., mathematics values), and Hispanic students exhibit higher mathematics interest and intrinsic motivation (Else-Quest et al., 2013; Stevens, Olivárez Jr, & Hamman, 2006). Another study found that Black and Hispanic boys specifically, reported lower levels of educational aspirations, persistence, and self-efficacy related to science compared to boys from more economically advantaged backgrounds, White boys, and girls from all backgrounds (Perry et al., 2012). Still other research indicates both Black and Hispanic students have a similar level of interest in STEM majors as their White counterparts (Anderson & Kim, 2006; Hurtado, et al., 2010). Moreover, female students generally report less motivation and confidence in their CS and mathematics-related skills and abilities to learn these subjects compared to male students (Beyer, 2014; Cunningham et al., 2015; Doubé & Lang, 2012; Hur et al., 2017; Maste et al., 2016). Girls and women are also more likely to show math and spatial anxiety (Ferguson et al., 2015; Maloney et al., 2012; Sokolowski et al., 2019). Together, this collection of results underscores the importance of pursuing more research on discrepancies in STEM-related attitudes between different groups of students.

1.3 The Importance of Addressing Group Differences in Computer Science

The need to increase the number of learners interested in CS has never been greater (Scott et al., 2017; Vakil, 2018). Computing knowledge is essential to address the growing demand for CS skills in STEM jobs, yet we must also acknowledge that women and underrepresented minorities earn far few degrees in computing fields at this writing (Sax et al., 2018). Moreover, discrepancies in access to technology in the home and CS educational experiences persist and are particularly pronounced for Black and Hispanic students and youth from lower SES families (Wang et al., 2016). Together, these findings suggests that research attention must be paid to diverse learners' experiences with STEM and CS, teachers', and administrators' roles in broadening participation in CS, instructional practices facilitating inclusive practice and CS learning, and educational structures and policies contributing to equitable CS educational opportunities (Ryoo et al., 2019; Santo et al., 2019).

It is also essential to begin CS education in the early grades. Introducing young learners to even basic CS concepts (i.e., plugged and offline, unplugged lessons) is crucial during the elementary school years for several reasons. First, doing so helps form the foundation for future learning in the subject, and second, it allows students to experiment with a complex and challenging subject before they form opinions of their abilities (Prottsman, 2014). There is little research on CS education in the early grades and even less that examines which specific learning experiences contribute to diverse students' pursuit of CS. The current study addresses these gaps in the literature by exploring the CS experiences of diverse learners in the early grades. This work will further increase our understanding of how to promote educational equity in CS education as measured by academic and attitudinal outcomes.

1.4 *Approaches to Examining Group Differences in Outcomes*

Prior research considering group differences in academic achievement and attitudes has employed a variety of research approaches. However, many studies simply *describe* differences in outcomes and studies that focus on comparing outcomes, often privilege certain socio-demographic groups, namely White, male students, over others (Gutiérrez & Dixon-Román, 2011). Such 'gap gazing' illuminates the presence of differences but does not provide any insights as to the nature of disparities that can inform efforts to change them (Lubienski, 2008).
Research that goes beyond 'gap gazing' operates on the *assumption* that instructional practices, programs, and educational interventions will *not impact all students in the same way* because members of diverse populations have unique needs (Gutiérrez, & Jaramillo, 2006; Rodríguez et al., 2016). Research methods need to account for the nuances that happen when programs are implemented (Vossoughi et al., 2016). To do so, researchers must utilize diverse methodological approaches that move beyond a sole focus on "what works" (Bullock, 2012) to also consider what works *for whom and under what conditions*. Approaching the study of CS achievement and attitudinal outcomes in this manner will elevate the importance of achieving equitable outcomes across diverse groups of learners.

2. Material Studied

This research was carried out through a research-practice partnership (RPP) focused on finding time for CS in the elementary school day, which is often dominated by literacy and mathematics instruction. The strategy was to create interdisciplinary problem-based learning modules that would be taught during the literacy block. The modules were organized around a science or social studies theme and had science, English/Language Arts, and associated CS lessons. The CS lessons were part of the Code.org courses available worldwide.

We referred to the modules as "Time4CS" Modules (see Century, Ferris, & Zuo, 2020 for a more detailed discussion of the modules). During the study, third-fifth grade teachers used two Time4CS Modules during the 2016-2017 school year. Each module was taught over a 5-8 week period and was composed of five lesson "collections." Each collection had four 60-minute lessons written in the 5E Model (i.e., Engage, Explore, Explain, Extend, Evaluate; Bybee, 2009) and had at least one CS lesson. Each module ended with a culminating project that entailed using CS to create a final product highlighting what students learned.

Following a quasi-experimental design, the project's primary research questions explored outcome differences for students in classrooms where teachers implemented and did not implement the Time4CS Modules. These analyses found that teachers who used the Time4CS Modules taught more CS than teachers in the comparison group. However, no differences in attitudinal or academic achievement outcomes were observed between students in treatment and comparison groups (Century, Ferris, & Zuo, 2020). Because this was the case, we utilized *component-based research* (CBR) to conduct further, in-depth analysis to explore associations between *specific components* of the Time4CS Modules and student outcomes. Using CBR revealed positive associations between particular Time4CS Module components and students' attitudinal and achievement outcomes. For example, we found positive associations between teachers' use of interdisciplinary teaching practices and students' achievement outcomes (Century, Ferris, & Zuo, 2020). In the findings reported here, we used CBR to conduct follow-up analyses that move beyond the aforementioned findings to deeply examine differences in student outcomes based on socio-demographic characteristics *and* the effect of teachers' instructional practices on outcomes for students in different groups. To do so, we investigated two additional research questions:

1. Are there differences in students' attitudinal and academic achievement outcomes based on socio-demographic characteristics?
2. Does the implementation of particular Time4CS Module components amplify or reduce group differences in students' attitudinal and academic achievement outcomes?

3. Area and Sample Description

The RPP team included researchers from the University of Chicago and practitioners from Broward County Public Schools (BCPS) in Broward County, Florida. BCPS is the sixth-largest school district in the United States consisting of 234 schools (i.e., 136 elementary, 37 middle, and 33 high schools) with a diverse student population (i.e., 51.2% White, 40.4% Black, 34.7% Hispanic, and 3.8% Asian; 11.9% English Language learners and 13.4% Exceptional students [students with Individualized Education Plans (IEPs)]). Sixteen BCPS elementary schools with 321 teachers and 5791 students from diverse racial, ethnic, and socioeconomic backgrounds participated in the study (Table 1; Table 2). A randomized block design (e.g., Dunlap, Cortina, Vaslow, & Burke, 1996; Kirk, 2007) was used to assign schools to treatment (i.e., eight schools; use of the Time4CS module) or comparison (i.e., 8 schools; regular classroom instructional practice) conditions.

Table 1. Student demographic characteristics for the treatment and comparison conditions

Demographics		Treatment	Control
Gender	Male	1035	932
	Female	962	878
Race	White	958	854
	African American	418	345
	Multi-ethnic	331	321
	Asian	111	74
	Other	220	186
Ethnicity	Hispanic	1081	1012
	Non-Hispanic	1859	1835
Grade Level	3rd	775	549
	4th	672	711
	5th	726	687
SES	Students not receiving free and reduced price lunch	1064	1091
	Students receiving free and reduced price lunch	917	704
English language proficiency	Instructed on acquiring English as a second language	290	269
	Still being monitored/exiting the program to learn English	171	171
	Native English speaker	1393	1245
Experience with Code. org	Previous experience completing Code.org activities prior to completing the questionnaires	1784	1408
	No previous experience completing Code.org activities	93	198

Table 2. Teacher demographic characteristics for the treatment and comparison conditions

Demographics		Treatment	Comparison
Gender	Male	3	20
	Female	67	134
Race/Ethnicity	White	38	80
	Black or African American	8	24
	Hispanic or Latino/Latina	18	30
	Other	NA	4
	Prefer not to answer	3	10
	Multiethnic	3	9
Grade Level	3rd	31	62
	4th	17	49
	5th	22	51
Highest Degree Earned	Bachelor’s Degree	43	81
	Master’s degree	25	72
	Doctoral or Professional Degree (Ph.D., Ed.D., MD, JD)	2	6
Degree in CS	Yes	1	2
	No	69	156

Note. The average years of teaching experience for treatment group teachers was 16.86 years, and for comparison group teachers was 14.38 years. The average years of teaching CS at any grade level for treatment group teachers was 1.41 years, and for comparison group teachers was 1.91 years. The average age for treatment group teachers was 42.66 years, and for comparison group teachers was 41.19 years.

4. Methods

4.1 Theoretical Approach – Component-Based Research

This study used a component-based research (CBR) approach. CBR deconstructs an innovation into its components, organizes those components into categories, and uses the categories as a foundation for looking beyond *whether* the innovation "worked," to examine *what parts* of an innovation work, *for whom,* and *under what conditions.* In this study, CBR was guided by an innovation implementation framework (Century, Cassata, Rudnick, & Freeman, 2012) that organized the components of the Time4CS Modules into two main categories —structural (e.g., number of CS lessons taught, lesson omission) and interactional (e.g., teacher facilitation of cognitively-demanding work, teacher facilitation of student intellectual risk taking). See details in Table 3. We also measured contextual factors hypothesized to influence teachers' Time4CS Module implementation (e.g., teachers' years of teaching CS, perception of their own innovativeness or resourcefulness) utilizing a framework that outlines factors that influence implementation (Century & Cassata, 2014). By using CBR, we were able to identify the impact of specific innovation components on students' attitudinal and achievement outcomes *and* disentangle associations between specific components and outcomes for particular student groups. Employing CBR enabled us to move beyond "gap gazing" to generate greater understanding about potential ways to reduce group differences in attitudinal and achievement outcomes across diverse groups of students.

Table 3. Component-based research: Structural and interactional components

Structural Components	**Measure**
Time4CS Module Completion	Yes/No
Grade-level assigned Code.org CS lessons	Percentage of lessons taught from the Code.org Fundamentals course assigned to the grade
Non-grade-level Code.org CS lessons	Percentage of lessons taught from the other Code.org courses available (Above or below grade level)
Additional CS (Any other non-Code.org CS lessons)	Yes/No
Interactional Components	**Sample Items** **(5-point scale: never-always)**
Teacher facilitation of group work	How often did you encourage group members to solve problems together?
Teacher facilitation of cognitively demanding work	How often did you ask students to explain how they solved a problem?
Teacher facilitation of students taking intellectual risks	How often did you ask students to answer a question even if they were unsure?
Teacher use of interdisciplinary teaching practices	How often did you explicitly cover standards from multiple subject areas in the same lessons?
Teacher facilitation of group work	How often did you encourage group members to solve problems together?

4.2 Measures and Data Sources

Teacher Implementation Questionnaire: This questionnaire, administered in spring 2017 using Qualtrics ©, assessed Time4CS Module structural and interactional component implementation and factors influencing implementation, such as teacher attitudes and demographic characteristics. All items and response scales included in the Time4CS Teacher Questionnaire are listed in Appendix A.

Student Attitude Questionnaire: Students completed questionnaires at the beginning and end of the 2016-2017 academic year using Qualtrics ©. These questionnaires assessed students' attitudes about school (i.e., school affinity, school self-efficacy) and CS (i.e., CS affinity, CS self-efficacy, CS identity, and CS utility) using items scored on a five-point Likert scale. These questionnaires also asked about previous experience with Code.org CS activities (i.e., yes/no) prior to the study. All items and response scales included in the Time4CS Student Questionnaire are listed in Appendix B.

Florida Standards Assessment (FSA) and Achieve3000 scores: Measures of student academic achievement included the Florida Standards Assessment (FSA) and a diagnostic tool BCPS used called Achieve3000. The state administers FSA assessments in ELA and mathematics in 3rd, 4th and 5th grades. The FSA science test is administered in 5th grade, only. The FSA assesses student knowledge and critical thinking skills. The Achieve3000 literacy test is administered three times a year to assess students' reading level as a means to support differentiated instruction. In both cases, BCPS de-identified and shared these data for the 2016-2017 academic year with the research team.

Administrative Data: BCPS also shared students' de-identified socio-demographic data, including gender, ethnicity (i.e., non-Hispanic and Hispanic status), race, ELL proficiency, and socio-economic status (i.e., free and reduced-price lunch [FRL]). Students also self-reported their gender, ethnicity, and race, and we cross-referenced these data with the district data to account for missing values.

4.3 Using CBR to Measure Time4CS Module Implementation

Studies of CS and other school interventions do not typically measure intervention implementation and when they do, implementation measurement is only conducted with treatment groups. However, CBR recognizes that some intervention components may in fact be accessible to teachers *in the comparison group,* and therefore, must be measured as well. This was the case in this study. Specifically, the Code.org CS lessons associated with the Time4CS Modules were publicly available to *all* teachers in the district. Similarly, the Time4CS Modules' interactional components included instructional practices (e.g., facilitating group work, using interdisciplinary teaching practices) that any teacher can use, with or without the Time4CS Modules. Therefore, while teachers in the treatment condition were asked to report on implementation of *all* module components, teachers in the comparison condition were asked to report on the components (and related factors) they could have enacted even though they did not implement the modules.

Structural implementation components: Teachers in the treatment group reported whether they completed (i.e., yes/no) the Time4CS Module with their students. They also reported on how many module lessons they completed and the sections of those lessons they used or omitted. Moreover, because we used a CBR approach, teachers in *both* treatment and comparison groups reported the number of grade-level assigned *and* non-grade-level assigned Code.org CS lessons they completed as well as whether they completed (i.e., yes/no) any additional, non-Code.org CS activities. The number of grade-level assigned, and non-grade-level assigned Code.org CS lessons were converted into completion percentage scores by dividing the number of lessons teachers reported completing by the total number of lessons available to complete. Higher completion percentage scores represented greater CS exposure for students.

Interactional implementation components: All teachers reported on their use of interactional Time4CS Module components. These components entailed the instructional strategies embedded in the module, but as explained above, could also be carried out in "business-as-usual" (comparison) classroom instruction. The questionnaires asked teachers to report on use of interdisciplinary teaching practices (e.g., How often did you explicitly cover standards from multiple subject areas in the same lesson?) as well as their facilitation of group work (e.g., How often do you encourage group members to work to solve problems together?), cognitively-demanding work (How often did you ask students to explain how they solved a problem?), and intellectual risk-taking (e.g., How often did you ask students to answer a question even if they were unsure?). Each scale used a five-point Likert scale (1 = *Never*, 5 = *Always*) with higher scores representing greater use of each component.

Influential Factors: The factor framework mentioned in section 4.1 organizes influential factors (i.e., contexts and conditions) into several levels (i.e., individual, organizational, and environmental) (Century & Cassata, 2014). The questionnaires asked teachers to report on three *individual* (teacher) factors, including innovativeness (e.g., I experiment with new practices all the time), resourcefulness and coping (e.g., I am able to manage the pressure and stress at my school well), and years of teaching CS. Each scale used a six-point Likert scale (1 = *Strongly disagree*, 6 = *Strongly agree*) with higher scores representing greater use of these practices.

4.6 Analytic Strategy

To conduct our analysis, we used Hierarchical Linear Modeling (HLM) with HLM7 software (Raudenbush, Bryk, Cheong, Congdon, & Toit, 2011). First, two-level, intercept-as-outcome models were conducted, which identified significant group differences in achievement and attitudinal outcomes based on students' socio-demographic characteristics (i.e., Level-1 variables)[1]. Next, two-level, random-intercept and coefficient-as-outcome models were conducted to examine the interactive effects between teacher-level implementation of particular interactional Time4CS Module components (i.e., specific instructional practice variables) and students' socio-demographic characteristics (Raudenbush & Bryk, 2002). In this second set of analyses, the Level-1 intercepts were no longer modeled as a function of structural or interactional component implementation *or* factors at Level-2 as in the first set of analyses; instead, they were included as random effects. More specifically, significant Level-1 coefficients for students' socio-demographic characteristics identified in the first set of analyses were modeled as a function of structural or interactional component or factor variables at Level-2 in the second set of analyses.
In all analytic models, students' socio-demographic characteristic variables were entered at Level-1. These variables were dummy-coded as follows: gender (reference group: male); race (reference group: White students); ethnicity (reference group: non-Hispanic students); SES (reference group: students who *do not* receive free and reduced-price lunch); and ELL proficiency (reference group: native English speakers). Students' previous experience completing Code.org activities was included at Level-1 as a control variable as were their socio-demographic characteristics. Students' attitudes about school and CS collected during pre-questionnaire administration (i.e., Fall 2016) were also included at Level 1 in models where attitudinal outcomes were examined. Level-2 teacher variables included structural and interactional implementation components and factors. All categorical variables were uncentered, whereas continuous variables were centered around the group mean for Level-1 variables and the grand mean for Level-2 variables.

5. Results

Significant findings are reported in text. Standardize beta coefficients and standard error terms are reported for all associations between students' socio-demographic characteristics (i.e., Level-1 variables) and their academic achievement and attitudinal outcomes are included in Table 4 and 5, respectively. In Table 6 and 7, we report all standardized beta coefficients and standard error terms for the interactive effects between students' socio-demographic characteristics (i.e., Level-l variables) and structural and interactional components of Time4CS module implementation and factors (i.e., Level-2 variables)[2]. The significant group differences reported in-text and in Tables 4-7 emerged when controlling for all other student socio-demographic characteristic variables in the analytic model.

5.1 Academic Achievement Outcomes

Gender: On the science FSA, female students scored significantly lower than males ($\beta = -5.52$, $p < 0.01$).
Race: On the mathematics FSA, Asian students ($\beta = 5.92$, $p < 0.01$) scored significantly higher, and Black students ($\beta = -8.42$, $p < 0.01$) scored significantly lower than White students. On the ELA FSA, Black students scored significantly lower than White students. This main effect was qualified by a significant interaction, which indicated that this score differential (i.e., gap) on the ELA FSA between Black and White students was **larger** in classrooms where teachers reported completing a **higher percentage of grade-level assigned CS lessons** (a structural component) ($\beta = -34.69$, $p < 0.01$).

[1] In Century, Ferris, & Zuo (2020), we reported on associations between Level-2 variables and students' attitudinal and academic achievement outcomes.
[2] Table 6 and 7 highlight interactive effects on Science FSA and CS Affinity. We have only included tables for these two outcomes due to space limitations. Tables illustrating interactive effects for additional academic achievement (i.e., Mathematics FSA, ELA FSA, and Achieve3000) and attitudinal (i.e., School Affinity, CS Utility, and CS Identity) outcomes are available from the authors upon request.

Table 4. Results from the two-level, intercept-as-outcome models investigating associations between level-1 variables and students' academic achievement outcomes

	Mathematics FSA	ELA FSA	Science FSA	Achieve 3000 scores
Asian students	5.92 (1.69)*	0.85 (1.79)	8.23 (3.60)*	9.71 (10.33)
African American students	-8.42 (1.48)*	-7.54 (1.59)*	-8.46 (2.52)*	-8.96 (7.99)
Hispanic students	-0.06 (1.16)	0.70 (1.07)	2.55 (1.81)	1.07 (5.29)
Female students	-0.77 (0.94)	0.78 (0.89)	-5.52 (1.55)*	-8.79 (4.47)*
Students receiving FRL	-5.70 (1.12)*	-5.80 (1.15)*	-6.58 (1.93)*	-31.03 (5.72)*
English language learners	-8.08 (1.60)*	-13.05 (1.79)*	-8.78 (3.03)*	-8.71 (8.48)

Note. * $p < .05$. Standardized beta coefficients and standard error (SE) terms are presented in the table with SE reported in parentheses.

Table 5. Results from the two-level, intercept-as-outcome models investigating associations between level-1 variables and students' attitudinal outcomes

	School Affinity	CS Affinity	CS Identity	CS Utility
Female students	0.01 (0.07)	-0.13 (0.08)	-0.10 (0.08)	-0.03 (0.07)
Asian students	0.22 (0.13)	0.01 (0.18)	0.28 (0.14)*	0.01 (0.14)
African American students	-0.09 (0.12)	-0.10 (0.14)	-0.08 (0.13)	-0.10 (0.11)
Hispanic students	-0.03 (0.07)	-0.09 (0.07)	0.03 (0.07)	0.07 (0.08)
Students receiving FRL	0.10 (0.08)	0.16 (0.08)*	0.16 (0.08)*	0.11 (0.07)
English-language learners	0.03 (0.11)	0.06 (0.11)	-0.08 (0.14)	-0.04 (0.1)

Note. * $p < .05$. Standardized beta coefficients and standard error (SE) terms are presented in the table with SE reported in parentheses.

On the science FSA, Asian students scored significantly higher compared to White students. This main effect was qualified by a significant interaction indicating that this score differential on the science FSA between Asian and White students was **larger** in classrooms where teachers reported greater use of **interdisciplinary teaching practices** (interactional component) ($\beta = 55.60$, $p < 0.05$). On the other hand, Black students scored significantly lower on the science FSA compared to White students. This main effect was qualified by a significant interaction, which indicated that this score differential on the science FSA between Black and White students was **smaller** in classrooms where teachers reported **higher levels of resourcefulness and coping** (influential factor) ($\beta = 10.45$, $p < 0.05$).

SES: Students receiving FRL scored significantly lower on Achieve3000 Lexile scores ($\beta = -31.03$, $p < 0.05$) and the science FSA ($\beta = -6.58$, $p < 0.01$). Students receiving FRL scored significantly lower on the ELA FSA as well, and this main effect was qualified by a significant interaction, which indicated that this score differential on ELA FSA between students receiving FRL and those who do not was **larger** when teachers **completed more additional (non-Code.org) CS activities** (structural component) ($\beta = -6.41$, $p < 0.01$) and reported **higher levels of resourcefulness and coping** (influential factor) ($\beta = -5.23$, $p < 0.01$). In addition, students receiving FRL scored significantly lower on the mathematics FSA. This main effect was qualified by a significant interaction, which indicated that this score differential on mathematics FSA between students receiving FL and those who do not was **smaller** when teachers reported **higher levels of facilitation of group work** (interactional component) ($\beta = 4.84$, $p < 0.05$).

ELL Proficiency: ELLs ($\beta = -8.78$, $p < 0.01$) scored significantly lower on the science FSA than native English speakers. ELLs scored significantly lower on the ELA FSA than native English speakers. This main effect was qualified by a significant interaction, which indicated that this score differential on ELA FSA between ELLs and native English speakers was **larger** when teachers reported **higher levels of facilitation of group work** (interactional component) ($\beta = -9.75$, $p < 0.01$). However, this score differential on ELA FSA was **smaller** when teachers reported **higher levels of resourcefulness and coping** (influential factor) ($\beta = 5.92$, $p < 0.05$). Furthermore, ELLs scored significantly lower on the mathematics FSA. This main effect was qualified by a significant interaction, which indicated that this score differential on mathematics FSA between ELLs and native English speakers was **smaller** when teachers **reported completing a higher percentage of non-grade-level Code.org CS lessons** (β = 30.13, p < 0.01) (structural component), **facilitating higher levels of student intellectual risk taking** (interactional component) (β = 9.98, p < 0.01), **higher levels of resourcefulness and coping (**influential factor) (β = 6.60, p < 0.05), **and more years of teaching CS** (influential factor) (β = 1.77, p < 0.05). On the other hand, this score differential on mathematics FSA was **larger** when teachers reported higher levels of **facilitation of group work** (interactional component) (β = -12.48, p < 0.01).

5.2 Attitudinal Outcomes

Gender: No significant differences were observed for students' attitudinal outcomes.
Race: Asian students (β = 0.28, p < 0.05) reported significantly higher CS identity attitudes than White students.

SES: Students receiving FRL reported more positive CS affinity attitudes. This main effect was qualified by significant interactions, which indicated that this difference in CS affinity attitudes between students receiving FRL and those that do not was **larger** in classrooms where teachers **taught the Time4CS Modules** (whole intervention) (β = 0.47, p < 0.05), and when teachers reported **greater use of interdisciplinary teaching practices** (interactional components) (β = 0.46, p < 0.01) and **more years of teaching CS** (influential factor) (β = 0.09, p < 0.01). In addition, students receiving FRL reported more positive CS identity attitudes (β = 0.16, p < 0.05).

Table 6. Results from the two-level, random-intercept and coefficient-as-outcome model examining students' scores on the science FSA

	Level 1 Variables				
Level 2 Variables	Asian Students	African American Students	Female Students	Students receiving FRL	ELLs
Non-grade level assigned lessons	71.00 (68.79)	26.26 (31.39)	-3.11 (17.56)	13.86 (23.5)	24.55 (41.29)
Assigned grade-level computer science lessons	-170.22 (147.96)	-22.82 (45.16)	1.66 (30.15)	-56.46 (42.28)	-35.28 (95.93)
Additional computer science	40.93 (32.41)	13.52 (8.87)	-10.36 (6.17)	-10.11 (8.56)	-2.1 (13.49)
Interdisciplinary teaching practices	55.60 (25.19)*	-5.86 (6.91)	-5.06 (4.18)	7.02 (4.94)	-2.62 (7.16)
Group work	-24.68 (29.07)	-6.16 (6.57)	-3.96 (5.59)	-4.23 (6.67)	-2.38 (8.99)
Cognitively demanding work	19.88 (24.42)	1.38 (6.61)	0.29 (4.91)	2.13 (6.10)	-4.14 (8.34)
Student intellectual risk-taking	12.49 (14.72)	4.69 (7.46)	-1.78 (4.53)	-5.63 (5.88)	3.75 (8.21)
Treatment condition	24.07 (18.85)	7.57 (8.5)	2.87 (5.49)	7.62 (7.10)	-0.38 (12.9)
Innovativeness	-16.00 (13.37)	-0.57 (5.13)	0.41 (3.69)	2.00 (4.65)	-8.14 (6.93)
Resourcefulness and coping	21.37 (12.96)	10.45 (4.87)*	0.73 (4.18)	0.75 (5.04)	12.87 (7.83)
Years of teaching CS	-1.03 (3.50)	-2.46 (1.89)	0.53 (0.93)	1.15 (1.28)	3.56 (2.45)

Note. * $p < .05$. Standardized beta coefficients and standard error (SE) terms are presented in the table with SE reported in parentheses. We controlled for students' ethnicity (i.e., Hispanic and non-Hispanic identity) in all models. No significant group differences emerged for ethnicity; therefore, it was not included in any interaction terms at Level-2. ELL = English-language learner.

Table 7. Results from the two-level, random-intercept and coefficient-as-outcome model examining students' CS affinity attitudes

	Level-1 Variables				
Level-2 Variables	Asian Students	African American Students	Female Students	Students receiving FRL	ELLs
Non-grade level assigned lessons	_	_	_	-0.25 (0.67)	_
Assigned grade-level computer science lessons	_	_	_	-1.55 (1.53)	_
Additional computer science	_	_	_	0.05 (0.21)	_
Interdisciplinary teaching practices	_	_	_	0.46 (0.13)*	_
Group work	_	_	_	0.10 (0.15)	_
Cognitively demanding work	_	_	_	-0.13 (0.17)	_
Student intellectual risk-taking	_	_	_	-0.22 (0.15)	_
Treatment condition	_	_	_	0.47 (0.20)*	_
Innovativeness	_	_	_	-0.09 (0.15)	_
Resourcefulness and coping	_	_	_	0.21 (0.12)	_
Years of teaching CS	_	_	_	0.09 (0.03)*	_

Note. * $p < .05$. Standardized beta coefficients and standard error (SE) terms are presented in the table with SE reported in parentheses. Group differences were only observed for students' socio-economic status. We controlled for students' ethnicity (i.e., Hispanic and non-Hispanic identity) in all models. No significant group differences emerged for ethnicity; therefore, it was not included in any interaction terms at Level-2. ELL = English-language learner.

6. Discussion

6.1 Academic Achievement Gaps Persist

Considerable efforts have been made to address inequities in achievement outcomes for diverse populations of students; however, gaps persist. Unsurprisingly, and consistent with previous research, findings from the current study indicate that Black students, students from economically disadvantaged backgrounds (i.e., students receiving FRL), students acquiring English as a second language, and female students scored lower on some achievement outcomes compared to their reference groups (Hadden et al., 2020; Xu, 2019). This study addresses calls from prior research (e.g., Lubienski, 2008) by moving beyond identifying group differences in outcomes to offering potential reasons *why* such gaps persist. Employing CBR in this study enabled us to identify the ways different structural and interactional *components* of the Time4CS Modules influenced the degree of group difference (i.e., amplifying versus reducing it).

6.2 Structural Components of the Time4CS Modules Contributed to Larger or Smaller Group Differences in Outcomes

This study found that CS-related components (e.g., implementation of grade-level assigned Code.org lessons; implementation of non-grade level assigned Code.org lessons; implementation of additional non-Code.org lessons) influenced whether the group difference was amplified or reduced. Our earlier analysis (Century, Ferris, & Zuo, 2020) showed no associations between the percentage of grade-level assigned Code.org lessons and FSA outcomes. However, by disentangling the effect of student group, we observed group differences between Black and White students for mathematics FSA outcomes, and this difference was amplified in classrooms where teachers reported teaching more grade-level assigned CS lessons. In BCPS, similar to other large, urban public school districts, it is possible that disproportionate numbers of Black students reside in low-income areas and attend under-resourced schools with fewer educational supports. When teachers dedicated more time within the literacy block to grade-level assigned CS lessons, this in turn, may have reduced the time Black students had available for instruction in FSA tested areas (e.g., mathematics). This finding suggests Black students face compounded inequities; attending under-resourced schools limits the instructional support present at the outset so when opportunities to engage in CS are presented, outcomes in these other subject areas may suffer.

We also observed through use of CBR that teaching "additional CS activities" was associated with lower ELA FSA scores for students receiving FRL. Because this group by definition come from families with lower incomes and/or parental educational attainment, they also reside in, and attend, more under-resourced neighborhoods and schools. It is possible that the presence of additional CS activities took time away from the particular kinds of ELA instructional supports these learners needed to score highly on the state-administered examination.

Moreover, our results about non-grade level assigned Code.org CS lessons underscore additional equity-related complexities. Our earlier analysis (Century, Ferris, & Zuo, 2020) illustrated that non-grade level assigned Code.org CS lessons were positively associated with students' ELA and mathematics FSA scores. However, when examining specific student groups, the presence of more non-grade level assigned Code.org CS lessons reduced the group difference in mathematics FSA between ELLs and their native English speakers. This group difference may have been reduced for several reasons. For instance, teachers may have selected non-grade-level assigned Code.org CS lessons that complemented and enhanced the curricula they were already teaching, in turn allowing them to provide more appropriate scaffolding for language learners. Or, they may have selected non-grade-level Code.org CS lessons to support their own, and their students', varied experiences, and interests. Emphasizing students' diverse experiences contributes to the creation of an inclusive classroom environment where teaching reflects the diversity of needs of all students and teachers consider how to include *rather than* exclude learners who experience difficulty (i.e., inclusive practice; Abels, 2019), which is central to cultivating positive learning experiences with ELLs (de Jon et al., 2013). Teachers may also have felt that their students were capable of completing these optional lessons, which at times covered more-advanced CS content, because they felt that their students had the ability to understand more-challenging CS concepts. Thus, more non-grade level Code.org CS lessons may have been conducive to their learning, engagement, and growth as students, overall, as well as reducing potential outcome differences based on language proficiency. Conversely, teachers may have carried out more of these lessons covering less-advanced content (i.e., below grade level: fourth grade students completing non-grade level assigned lessons intended for third graders) because they believed they were more-closely aligned with their students' abilities. Additional studies focusing specifically on academic achievement and STEM and CS attitudes are needed with ELLs in the future to better understand how to create equitable learning opportunities for this growing number of students and foster teachers' inclusive practice in their classrooms.

Significant attitudinal effects only emerged in the presence of the structural component related to whether teachers taught the Time4CS Module. Specifically, FRL students demonstrated higher CS affinity attitudes, and this difference was larger in classes where the Time4CS Module was taught. This finding suggests that although we found no differences between treatment and comparison groups as a whole (Century, Ferris, & Zuo, 2020), it appears that for students receiving FRL, engaging with the Time4CS Module (e.g., exposure to CS topics presented in an interesting manner) was beneficial for their CS affinity.

6.3 Interactional Components of the Time4CS Modules Contributed to Larger and Smaller Group Differences in Outcomes

We also observed interaction effects with some of the interactional Time4CS Module components. For example, for mathematics FSA outcomes, teachers' facilitation of student intellectual risk-taking contributed to reducing group differences in outcomes between ELL students and native-English speakers. These findings suggest that ELLs may benefit from opportunities to take risks during instruction, and that those opportunities may positively contribute to better academic achievement. Accordingly, teachers who create intellectually safe and supportive learning environments when teaching CS may positively shape students' overall outlook on learning and motivation to persevere when faced with uncomfortable learning challenges (Rabe, 2018). As such, risk-taking may be especially important for ELLs who need extra support to combat stereotypes about their abilities and what they are truly capable of accomplishing in the classroom (Diamond et al., 2016).

Findings related to teachers' facilitation of group work were mixed. For example, when teachers reported more facilitation of group work, the group difference in mathematics FSA scores between low and high SES students was reduced. This finding supports prior research suggesting that group work promotes academic achievement for disadvantaged students (Bailey & Bradbury-Bailey, 2006). Through group work, these students may receive extra peer-to-peer support when working with their classmates as well as increased opportunities to take ownership of work aligning to their skillsets (Slavin, 2014). However, when teachers reported more facilitation of group work, the difference widened between ELLs and Native-English speakers. This effect also emerged when examining ELA FSA scores. These findings may have resulted because group work requires greater communication and collaboration, which are areas where ELL students may be at a disadvantage due to their more limited English-language proficiency.

In addition, findings on interdisciplinary teaching practices further disentangled effects observed for all students in our previous paper (Century, Ferris, & Zuo, 2020). Our earlier analysis indicated that the presence of interdisciplinary teaching practices was positively associated with ELA FSA outcomes for the whole student population (Century, Ferris, & Zuo, 2020), whereas the current analysis revealed that the presence of these practices contributed to two instances of amplified group differences. Specifically, on the science FSA, the group difference between Asian and White students was larger in classrooms where teachers reported using more interdisciplinary teaching practices. Some research (Farris, 2015) has suggested a link between interdisciplinary teaching practices, multiculturalism, and culturally responsive instructional strategies, which may be beneficial for Asian students' academic achievement, in particular (Change & Le, 2010). Additionally, the group difference in CS attitudes between FRL students and their counterparts was larger when more interdisciplinary teaching practices were present. The reasons behind these findings are unclear, but we recommend that future researchers use CBR to investigate these effects more specifically to gain a better understanding of the role of interdisciplinary instructional practices.

6.4 Influential Factors Also Contribute to Larger or Smaller Group Differences in Outcomes

Two teacher characteristics contributed to changes in group differences in outcomes. First, the group difference in science FSA scores was reduced between Black and White students in classrooms where teachers reported having more resourcefulness and coping skills. Similarly, when teachers reported more resourcefulness and coping skills, the group difference between ELL and native-English speakers on mathematics FSA was reduced. On the other hand, the group difference on the ELA FSA between FRL students and their counterparts was larger in the presence of more teacher resourcefulness and coping. One can speculate that teachers who perceive themselves as being resourceful and having coping abilities might be more adept at inclusive practice (Bentley-Williams, Grima-Farrell, & Laws, 2017), in turn providing Black and ELL students with the opportunities and resources they need to increase their academic achievement. Seeing the findings for FRL students; however, suggests that these benefits are not universal, for reasons this study did not reveal and that should be the focus of future investigations.

More commonly measured teacher characteristics – years of teaching CS – were also associated with reductions in group differences. We observed smaller group differences between ELL students and their native-English speaking counterparts for mathematics FSA and ELA FSA scores. While neither of these characteristics was revealed to have direct effects on student outcomes (Century, Ferris, & Zuo, 2020), one can speculate that for ELL students in particular, having teachers with more experience with CS content is beneficial.

7. Conclusion

The current study explored differences in students' attitudinal and academic achievement outcomes based on teachers' implementation of the Time4CS Modules in their classrooms. We employed CBR to disentangle associations between *specific components* of the Time4CS Modules and students' attitudes and academic achievement. Our results assess beyond whether the Time4CS Modules worked and indicate that academic achievement and CS attitudinal gaps persist. Utilizing CBR allowed us to identify potential reasons why some *components* of the Time4CS Modules (i.e., intervention) were beneficial for certain groups of students, but not others. Use of CBR also underscored areas of continued improvement for researchers and educators striving to provide equitable CS learning experiences for young learners. By pinpointing how students' socio-demographic characteristics interact with aspects of their learning context (e.g., teachers' instructional practices), researchers and practitioners can better understand the contexts and conditions needed to promote desired student outcomes related to CS. Therefore, CBR is a valuable approach for moving beyond “gap gazing” to gain insights about what parts of an innovation worked, for which students, and under what conditions. Despite these promising findings and their implications for researchers and educational professionals dedicated to creating more-equitable CS learning opportunities, this project represents merely a starting point. Future research must address the current study’s limitations to generate insights that can inform the continued development of CS educational experiences for today’s diverse learners.

7.1 Limitations and Future Directions

Our findings must be considered in light of several limitations. First, the team was unable to control for the BCPS academic calendar (e.g., standardized testing schedule), which likely impacted implementation of the Time4CS Modules given that some teachers taught the modules before FSAs were administered, whereas others used the module after testing took place. Greater attention must be placed on when teachers are implementing the intervention in future studies so accurate effects of the intervention on outcomes can be generated. Second, the teacher and student questionnaire data is subject to self-report bias and socially-desirable reporting, which may contribute to inflated (e.g., more positive) response patterns in the data (Van de Mortel, 2008). Future research should adopt multi-method data collection strategies (Bulsara, 2015), including interviews or focus groups with teachers and students. Classroom observations could also be used to capture real-time accounts of teachers’ use of different components to better understand the reasons particular components may be associated with larger or smaller group differences (Archer et al., 2016). Third, the Time4CS Modules used were in the early stage of development during the study, and they may have required revisions to improve usability and to ease teachers’ learning curve when implementing them.

We also want to acknowledge two considerations for future examinations focused on equity. First, we did not use methods that accounted for intersectionality. Although our results underscore the importance of CBR when disentangling the effect of structural and relational components of a curriculum designed to promote more equitable student outcomes, follow-up investigations are also needed to better understand how different student characteristics intersect (e.g., race, gender). An intersectional conceptual and analytical approach (Else-Quest & Hyde, 2016a; Else-Quest & Hyde, 2016b) was not the primary focus of this investigation. We recognize that the use of additive analytic approaches (Cole, 2009) limits the conclusions that can be drawn regarding students’ dual and overlapping identities (e.g., being Asian *and* female; being Black *and* male; being multiethnic; being ELL and a FRL recipient). Nonetheless, our findings emphasize the need to look more deeply at group differences and recognize that instructional practices may be beneficial to some groups, but detrimental to others. As a result, we encourage future researchers to employ intersectional conceptual and analytical approaches to simultaneously examine student characteristics, attitudes, and academic outcomes as such investigations have the potential to better inform the creation of strategies that can promote educational equity.

Second, we want to acknowledge the implicit *and* explicit biases associated with our academic outcome measures, standardized tests (Knoester & Au, 2017). For instance, some studies indicate that standardized tests are inequitable by design (Au, 2010; Morgan, 2016; Well et al., 2019). Thus, focusing on standardized testing outcomes when striving for educational equity across diverse student populations overlooks how the test design itself contributes to group differences. Students’ standardized test scores may be misleading and not accurately depict student learning (Morgan, 2016). Future research needs to acknowledge these shortcomings when referencing empirical literature, use more equitable assessments, and investigate a wide variety of educational outcomes, including students' attitudes toward different subject areas, motivations in the classroom, and interests.

7.2 The Call for Research that Will Contribute to More Accurate and Useful Findings about Equitable CS Learning Opportunities

In light of these limitations, the findings of this study communicate a powerful message: CBR enables researchers to examine the impact of specific *aspects* of an intervention for different groups of learners. When studies of CS interventions show positive or negative main effects, it is essential to look deeper through such approaches. Examining the *components* of educational interventions and their effect on *particular* student groups will enable us to learn more about how to develop more equitable CS learning opportunities and teachers' inclusive practice. Although CS interventions vary widely, some of them share common components with others. Using a CBR approach can enable the field to accumulate knowledge about these shared components. This will pave the way for researchers and practitioners to develop robust knowledge about the best structures and interactions to use in CS instruction to meet the learning needs of all students.

Contributors

Drs. Ferris, Century, and Zuo are researchers at Outlier Research & Evaluation, UChicago STEM Education at the University of Chicago. Outlier Research & Evaluation is a group dedicated to change-focused work through research, evaluation, and strategic planning. We dare to make education equitable.

Acknowledgements
The authors would like to acknowledge the contributions of their practitioner colleagues at Broward County Public Schools, specifically Dr. Lisa Milenkovic, Debra Thomas, and Annmargareth Marousky. Their support allowed us to bring Time4CS to BCPS' elementary teachers and students.

References

Abedi, J. (2002). Standardized achievement tests and English language learners: Psychometrics issues. *Educational Assessment, 8*(3), 231-257.Anderson & Kim, 2006

Abels, S. (2019). Science Teacher Professional Development for Inclusive Practice. *International Journal of Physics & Chemistry Education, 11*(1), 19-29.

Archer, J., Cantrell, S., Holtzman, S. L., Joe, J. N., Tocci, C. M., & Wood, J. (2016). *Better feedback for better teaching: A practical guide to improving classroom observations.* John Wiley & Sons.

Au, W. (2010). *Unequal by design: High-stakes testing and the standardization of inequality.* Routledge.

Century, J., Cassata, A., Rudnick, M., & Freeman, C. (2012). Measuring enactment of innovations and the factors that affect implementation and sustainability: Moving toward common language and shared conceptual understanding. *The Journal of Behavioral Health Services & Research, 39*(4), 343-361.

Century, J., & Cassata, A. (2014). Conceptual foundations for measuring the implementation of educational innovations.

Century, J., Ferris, K. A., & Zuo, H. (2020). Finding time for computer science in the elementary school day: A quasi-experimental study of a transdisciplinary problem-based learning approach. *International Journal of STEM Education*, *7*, 1-16.

Bailey, D. F., & Bradbury-Bailey, M. E. (2006). Promoting achievement for African American males through group work. *The Journal for Specialists in Group Work*, *32*(1), 83-96.

Bentley-Williams, R., Grima-Farrell, C., Long, J., & Laws, C. (2017). Collaborative partnership: Developing pre-service teachers as inclusive practitioners to support students with disabilities. *International Journal of Disability, Development and Education*, *64*(3), 270-282.

Beyer, S. (2014). Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education*, *24*(2-3), 153-192. DOI: https://doi.org/10.1080/08993408.2014.963363

Britner, S. L., & Pajares, F. (2001). Self-efficacy beliefs, motivation, race, and gender in middle school science. *Journal of women and Minorities in Science and Engineering*, *7*(4).

Bullock, E. C. (2012). Conducting "Good" Equity Research in Mathematics Education: A Question of Methodology. *Journal of Mathematics Education at Teachers College*, *3*, 43–55.

Bulsara, C. (2015). Using a mixed methods approach to enhance and validate your research. *Brightwater Group Research Centre*, 1-82.

Bybee, R. W. (2009). The BSCS 5E instructional model and 21st century skills. Colorado Springs, CO: BSCS.

Caro, D. H. (2009). Socio-economic status and academic achievement trajectories from childhood to adolescence. *Canadian Journal of Education*, *32*(3), 558-590.

Celeste, L, Baysu, G., Phalet, K., Meeussen, L. & Kende, J. (2019). Can school diversity policies reduce belonging and achievement gaps between minority and majority youth? Multiculturalism, colorblindness and assimilationism assessed. *Personality and Social Psychology Bulletin*, *45*(11), 1603-1618. DOI: https://doi.org/10.1177/01146167219838577.

Chang, J., & Le, T. N. (2010). Multiculturalism as a dimension of school climate: The impact on the academic achievement of Asian American and Hispanic youth. *Cultural Diversity and Ethnic Minority Psychology*, *16*(4), 485-492.

Chimielewski, A.K. (2019). The global increase in the socioeconomic achievement gap, 1964-2015. *American Sociological Review*, *84*(3), 517-544. DOI: https://doi.org/10.1177/0003122419847165.

Cunningham, B. C., Hoyer, K. M., & Sparks, D. (2015). Gender Differences in Science, Technology, Engineering, and Mathematics (STEM) Interest, Credits Earned, and NAEP Performance in the 12th Grade. Stats in Brief. NCES 2015-075. *National Center for Education Statistics*.

Dahl, G. B., & Lochner, L. (2012). The impact of family income on child achievement: Evidence from the earned income tax credit. *American Economic Review*, *102*(5), 1927-56.

de Jong, E. J., Harper, C. A., & Coady, M. R. (2013). Enhanced knowledge and skills for elementary mainstream teachers of English language learners. *Theory into Practice*, *52*(2), 89-97. DOI: https://doi.org/10.1080/00405841.2013.770326

Diamond, E., Furlong, M. J., & Quirk, M. (2016). Academically resilient Latino elementary students bridging the achievement gap. *Contemporary School Psychology*, *20*(2), 160-169. DOI: https://doi.org/10.1007/s40688-016-0088-8

Doubé, W., & Lang, C. (2012). Gender and stereotypes in motivation to study computer programming for careers in multimedia. *Computer Science Education*, *22*(1), 63-78. DOI: https://doi.org/10.1080/08993408.2012.666038

Else-Quest, N. M., & Hyde, J. S. (2016a). Intersectionality in quantitative psychological research: I. Theoretical and epistemological issues. *Psychology of Women Quarterly*, *40*(2), 155-170.

Else-Quest, N. M., & Hyde, J. S. (2016b). Intersectionality in quantitative psychological research: II. Methods and techniques. *Psychology of Women Quarterly*, *40*(3), 319-336.

Else-Quest, N. M., Mineo, C. C., & Higgins, A. (2013). Math and science attitudes and achievement at the intersection of gender and ethnicity. *Psychology of Women Quarterly*, *37*(3), 293-309.

Farris, P. J. (2015). *Elementary and middle school social studies: An interdisciplinary, multicultural approach*. Waveland Press.

Ferguson, A. M., Maloney, E. A., Fugelsang, J., & Risko, E. F. (2015). On the relation between math and spatial ability: The case of math anxiety. *Learning and Individual Differences*, *39*, 1-12. DOI: https://doi.org/10.1016/j.lindif.2015.02.007

Fry, R. (2007). How Far behind in Math and Reading Are English Language Learners? Report. *Pew Hispanic Center*.

Gilmour, A., Fuchs, D. & Wehby, J. (2019). Are student with disabilities accessing the curriculum? A meta-analysis of the reading achievement gap between students with and without disabilities. *Exceptional Children*, *85*(3), 329-346. DOI: https://doi.org/10.1177/00144291875955830.

Gutiérrez, R., & Dixon-Roman, E. (2011). Beyond Gap Gazing: How Can Thinking About Education Comprehensively Help Us (Re)envision Mathematics Education? In *Mapping Equity and Quality in Mathematics Education* (pp. 21–34). DOI: https://doi.org/10.1007/978-90-481-9803-0

Gutiérrez, K. D., & Jaramillo, N. E. (2006). Looking for educational equity: The consequences of relying on Brown. *Yearbook of the National Society for the Study of Education,*

Hadden, I. R., Easterbrook, M. J., Nieuwenhuis, M., Fox, K. J., & Dolan, P. (2020). Self-affirmation reduces the socioeconomic attainment gap in schools in England. *British Journal of Educational Psychology*, *90*(2), 517-536. DOI: https://doi.org/10.1111/bjep.12291

Hung, M., Smith, W. A., Voss, M. W., Franklin, J. D., Gu, Y., & Bounsanga, J. (2020). Exploring student achievement gaps in school districts across the United States. *Education and Urban Society*, *52*(2), 175-193. DOI: https://doi.org/10.1177/0013124519833442

Hur, J. W., Andrzejewski, C. E., & Marghitu, D. (2017). Girls and computer science: experiences, perceptions, and career aspirations. *Computer Science Education*, *27*(2), 100-120. DOI: https://doi.org/10.1080/08993408.2017.1376385

Hurtado, S., Eagan, K., & Chang, M. (2010). Degrees of success: Bachelor's degree completion rates among initial STEM majors. *Higher Education Research Institute at UCLA, January*.

Knoester, M. & Au, W. (2017) Standardized testing and school segregation: like tinder for fire?. *Race Ethnicity and Education*, 20(1), 1-14, DOI: https://doi.org/10.1080/13613324.2015.1121474

Lubienski, S. T. (2008). On “Gap Gazing” in Mathematics Education: The Need for Gaps Analyses. *Journal for Research in Mathematics Education*, *39*(4), 350–356. Retrieved from https://www.jstor.org/stable/40539301

Maloney, E. A., Waechter, S., Risko, E. F., & Fugelsang, J. A. (2012). Reducing the sex difference in math anxiety: The role of spatial processing ability. *Learning and Individual Differences*, *22*(3), 380-384. DOI: https://doi.org/10.1016/j.lindif.2012.01.001

Master, A., Cheryan, S., & Meltzoff, A. N. (2016). Computing whether she belongs: Stereotypes undermine girls’ interest and sense of belonging in computer science. *Journal of educational psychology*, *108*(3), 424. DOI: http://doi.org/10.1037/edu0000061

Margolis, J., Estrella, R., Goode, J., Holme, J. J., & Nao, K. (2017). *Stuck in the shallow end: Education, race, and computing*. MIT press.

Morgan, H. (2016). Relying on high-stakes standardized tests to evaluate schools and teachers: A bad idea. *The Clearing House: A Journal of Educational Strategies, Issues and Ideas*, *89*(2), 67-72. DOI: https://doi.org/10.1080/00098655.2016.1156628

National Assessment of Educational Progress (Project). (2019a). *An Overview of NAEP. National Center for Education Statistics*, Institute of Education Sciences, US Department of Education.

OECD (2020). *PISA 2018 Results. Are students ready to thrive in an interconnected world?*. Volume VI. Paris: OECD Publishing. Retrieved from https://www.oecd-ilibrary.org/docserver/d5f68679-en.pdf?expires=1612562559&id=id&accname=ocid177408&checksum=DFA001F165222D5E66493C4028B68F98.

Orfield, G., Ee, J., Frankenberg, E., & Siegel-Hawley, G. (2016). "Brown" at 62: School Segregation by Race, Poverty and State. *Civil Rights Project-Proyecto Derechos Civiles*.

Orfield, G., & Lee, C. (2005). Why segregation matters: Poverty and educational inequality. *The Civil Rights Project at Harvard University.*

Owens, A. (2018). Income segregation between school districts and inequality in students’ achievement. *Sociology of Education*. *91*(1), 1-27. DOI: https://doi.org/10.1177/0038040717741180.

Pajares, F., & Kranzler, J. (1995). Self-efficacy beliefs and general mental ability in mathematical problem-solving. *Contemporary Educational Psychology*, *20*(4), 426-443.

Perry, B. L., Link, T., Boelter, C., & Leukefeld, C. (2012). Blinded to science: Gender differences in the effects of race, ethnicity, and socioeconomic status on academic and science attitudes among sixth graders. *Gender and Education*, *24*(7), 725-743. DOI: https://doi.org/10.1080/09540253.2012.685702

Prottsman, K. (2014). Computer science for the elementary classroom. *ACM Inroads*, *5*(4), 60-63. DOI: https://doi.org/10.1145/2684721.2684735

Rabe, S. B. (2018). Learning from failure: An action research case study on developing growth mindset through academic risk-taking in an athletic training program.

Raudenbush, S. W. & Bryk, A. S. (2002). *Hierarchical linear models: Applications and data analysis methods.* New Delhi, London, United Kingdom: Sage.

Raudenbush, S. W., Bryk, A. S., Cheong, Y. F., Congdon, R. T., & Toit, M. (2011). HLM 7: Hierarchical linear and nonlinear modeling. *Chicago, IL: Scientific Software International.*

Reardon, S. F., Weathers, E. S., Fahle, E. M., Jang, H., & Kalogrides, D. (2019). *Is separate still unequal? New evidence on school segregation and racial academic achievement gaps* (No. 19-06). CEPA Working Paper.

Rodríguez, C., Amador, A., & Tarango, B. A. (2016). Mapping Educational Equity and Reform Policy in the Borderlands: LatCrit Spatial Analysis of Grade Retention. *Equity and Excellence in Education*, *49*(2), 228–240. DOI: https://doi.org/10.1080/10665684.2016.1144834

Ryoo, J., Chapman, G., Flapan, J., Goode, J., Margolis, J., Ong, C., ... & Diaz, L. (2019, February). Going Beyond the Platitudes of Equity: Developing a Shared Vision for Equity in Computer Science Education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 657-658).

Santo, R., DeLyser, L. A., Ahn, J., Pellicone, A., Aguiar, J., & Wortel-London, S. (2019, February). Equity in the who, how and what of computer science education: K12 school district conceptualizations of equity in 'cs for all' initiatives. In *2019 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)* (pp. 1-8). IEEE.

Sax, L. J., Blaney, J. M., Lehman, K. J., Rodriguez, S. L., George, K. L., & Zavala, C. (2018). Sense of belonging in computing: The role of introductory courses for women and underrepresented minority students. *Social Sciences*, *7*(8), 122, 1-23.

Scott, A., Martin, A., McAlear, F., & Koshy, S. (2017, June). Broadening participation in computing: examining experiences of girls of color. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 252-256).

Slavin, R. E. (2014). Cooperative learning and academic achievement: Why does groupwork work? *Annals of Psychology*, *30*(3), 785-791. DOI: https://doi.org/10.6018/analesps.30.3.201201

Sokolowski, H. M., Hawes, Z., & Lyons, I. M. (2019). What explains sex differences in math anxiety? A closer look at the role of spatial processing. *Cognition*, *182*, 193-212.

Stevens, T., Olivárez Jr, A., & Hamman, D. (2006). The role of cognition, motivation, and emotion in explaining the mathematics achievement gap between Hispanic and White students. *Hispanic Journal of Behavioral Sciences*, *28*(2), 161-186. DOI: https://doi.org/10.1177/0739986305286103

Vakil, S. (2018). Ethics, identity, and political vision: Toward a justice-centered approach to equity in computer science education. *Harvard Educational Review*, *88*(1), 26-52.

Van de Mortel, T. F. (2008). Faking it: social desirability response bias in self-report research. *Australian Journal of Advanced Nursi*ng, 40-48.

Vossoughi, S., Hooper, P. A. K., & Escudé, M. (2016). Making Through the Lens of Culture Visions for Educational Equity. *Harvard Educational Review*, *86*(2), 206–232. DOI: https://doi.org/10.17763/0017-8055.86.2.206

Wang, J., Hong, H., Ravitz, J., & Hejazi Moghadam, S. (2016, February). Landscape of K-12 computer science education in the US: Perceptions, access, and barriers. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 645-650).

Wells, A.S., Keener, A., Cabral, L., & Cordova-Cobo, D. (2019) The more things change, the more they stay the same: The resegregation of public schools via charter school reform, *Peabody Journal of Education, 94*(5), 471-492, DOI: https://doi.org/10.1080/0161956X.2019.1668209

Xu, J. (2019). Challenges with Academic Achievement for Esl Students At The Elementary Level From A Teacher's Perspective. Retrieved from https://dune.une.edu/theses/222

Appendix A

Time4CS Teacher Implementation Questionnaire Items (Module 2, Grade 4)

School Information

1. What is the name of your school?
2. This year, what grade STEM+C Integrated Literacy Block materials did you use?

Module Completion

As part of the STEM+C Integrated Literacy Block project, you were provided with activities and lessons to use during a daily 180-minute integrated literacy block. There were two components of this180-minute block:

1) The 90-minute **Reading Block** using your normal ELA curriculum and/or suggested texts and center activities provided as part of the STEM+C materials; and

2) The 90-minute interdisciplinary **STEM+C Module**, which includes the 5E lessons that integrate science, social studies, and computer science.

Throughout the questionnaire, we will use "**Integrated Literacy Block**" to refer to the entire 180-minutes. We will use "**Reading Block**" and "**STEM+C Module**" to refer to the two **90-minute** components of the 180-minute integrated Literacy Block.

1. First, we'd like to ask you to think back to **last semester** (Fall 2016 - Winter 2016). Did you teach the STEM+C Integrated Literacy Block **Module 1**?
   1) Yes
   2) No

2. Now we'd like you to think about the most **recent semester** (Winter 2016 - Spring 2017). Did you teach the STEM+C Integrated Literacy Block **Module 2**?

   1) Yes
   2). No

Time

3. Now we're interested in learning more about how you used the Integrated Literacy Block materials.

Of the **total 180 minutes** designated each day for the Integrated Literacy Block, how many minutes a day, on average, did your students spending doing the following? **The times you select across all questions must add up to 180 minutes.**

_______ Doing STEM+C Module 5E activities
_______ Reading ELA **texts** recommended in the STEM+C Module materials
_______ Doing ELA **center activities** recommended in the STEM+C Module
_______ Doing other activities

4. You said students engaged in **other** activities during the 180-minute Integrated Literacy Block. What types of other activities did they do? Please select all that apply.

   1) **Literacy** activities not outlined in the STEM+C Module materials (including Achieve3000 articles)
   2) **Science** activities not outlined in the STEM+C Module materials
   3) **Social studies** activities not outlined in the STEM+C Module materials
   4) **Computer science** activities not outlined in the STEM+C Module materials
   5) Other (please explain):

Now we would like you to **just think about the STEM+C Module** component of the 180-minute Integrated Literacy Block. These are the 5E lesson activities on science, social studies, and computer science.

5. The STEM+C Module lessons include "**core**" learning resources that you were expected to use as well as "non-core" or **supplemental** elements that were optional. Please consider the entire STEM+C Module (**both the core AND supplemental activities of the 5E lessons**) when answering the following questions, unless otherwise stated.

   Approximately how many weeks did you spend doing the **entire STEM+C Module**?

   ▼ 1 week (1) ... 12 weeks

Each STEM+C Module lesson is intended to take 90 minutes, but we know there will be variations in how much time each teacher actually spends.

6. Do you think 90 minutes was sufficient to complete the STEM+C Module 5E lesson activities as outlined in the Module?
   1) Yes
   2) No

7. About how much more time (than the 90 minutes) each day do you feel you needed to complete all of the STEM+C Module 5E lesson activities?
   1) Less than 15 minutes
   2) 15-30 minutes
   3) 31-45 minutes
   4) 46-60 minutes
   5) More than 60 minutes

We also know that some teachers worked with media specialists and other "specials" teachers (e.g., art, music) to implement the STEM+C Module 5E lesson activities. For example, some teachers had their media specialist do the online computer science activities that were part of the STEM+C Module.

8. Did you have a media specialist or other specials teacher do any part of the STEM+C Module 5E lessons with your students?

1) Yes
2) No

You said you had a media specialist or other specials teacher do parts of the STEM+C Module lessons with your students. Please take those activities into account and answer as best as you can for yourself and on behalf of the specials teacher(s) when answering the following questions.

Lesson Omission

9. Did you omit any of the STEM+C Module lessons entirely? *By "entirely," we mean you skipped the entire lesson and did not do any parts of it.*

1) Yes
2) No

10. About how many STEM+C Module lessons did you omit entirely for any reason?

A few About half Most

0 5 10 14 19 24

Number of lessons ()	

11. In all the lessons you taught, did you ever omit one or more of the 5E components (e.g., "Engage," "Explain") for any reason? Do not count 5E components that a specials teacher taught on your behalf as an omission.

1) Yes
2) No

12. About how often did you omit each of the following 5E components for any reason? (Scale: Never, Rarely, Sometimes, Often, Always)

1) Engage
2) Explore
3) Explain
4) Elaborate/Extend
5) Evaluate

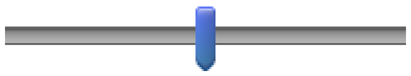
13. You said you omitted some portion of the STEM+C Module. From the list below, please select the top three reasons you did so.
    1) The activities were not identified as core parts of the lesson.
    2) I didn’t have access to the necessary technology (e.g., computers, Internet) to complete the activities.
    3) I had access to the necessary technology (e.g., computers, Internet) but it wasn’t working.
    4) The lesson parts were too long.
    5) The lesson parts required too much preparation.
    6) The lesson parts weren’t written clearly enough to teach them.
    7) I didn’t understand the content of the lesson parts well enough to teach them.
    8) The lesson parts weren’t relevant to my students.
    9) The lesson parts weren’t as important to teach compared to others I needed to teach.
    10) Other: (please explain) ____

Lesson Modification

Sometimes teachers modify lessons by departing from what is written in the curriculum materials. This includes using different materials to complete the lesson and/or teaching the lessons in a different way than written (e.g., doing the lesson as a whole group activity instead of partner work). Modification does not include omitting or skipping portions of the STEM+C Module.

14. Of the STEM+C Module lessons that you taught; did you modify them in any way?
    1) Yes
    2) No

15. About how many of the STEM+C Module lessons did you modify?

	A few		About half		Most	
	0	5	10	14	19	24
Number of lessons ()						

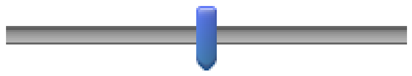
Lesson Supplementation

Sometimes teachers **add to** curricular materials like the STEM+C Module with their own activities or lessons.

16. Did you add any **new activities** to the STEM+C Module? *This includes anything you added that wasn't part of the STEM+C Module core and non-core “supplemental” activities.*
    1) Yes

2) No

17. In about how many STEM+C Module lessons did you add activities?

	A few		About half		Most	
	0	5	10	14	19	24
Number of lessons ()						

Culminating Project

18. Did your students complete a culminating project?
   1) Yes
   2) No

19. Which version of the culminating project did your students complete?
   1) Unplugged
   2) Plugged
   3) Both

20. Did your students complete any additional culminating projects that were not identified as part of the STEM+C Module?
   1) Yes
   2) No

21. Please describe the additional culminating project that your students completed.

____

Student Journaling

22. As part of the STEM+C Module, did your students use journals?
   1) Yes
   2) No

23. From the options below, please select the one that best represents how your students used journals.
   1) My students used a separate journal for each subject (e.g., one journal for math, one for science, one for literacy, etc.).
   2) My students used one journal for all subjects.

ACHIEVE3000

As part of the STEM+C Integrated Literacy Block project, you were asked to administer Achieve3000 diagnostic assessments to students. We'd like to know about your students' use of other (non-assessment) Achieve3000 materials and activities (e.g., nonfiction texts from Achieve3000).

24. On average, since the beginning of the school year, how many times **per week** did you have your students use Achieve 3000 (not-assessment) materials and activities?

	0	1	2	3	4	5
Number of times ()						

25. When students used Achieve3000 for non-assessment materials and activities, approximately how much time did they spend each time they used it? Please include time spent at school and when assigned for homework.

	0	15	30	45	60	75	90
Number of minutes ()							

Computer Science Materials

Now think back to both last semester (Module 1) and the most recent semester (Module 2).

26. Which of the following **Code.org, Course 3** and/or **Scratch** activities did you use with your students? Please select all that apply.

    Please include any lessons taught by a media specialist or other specials teacher on your behalf.

	Module 1	Module 2
Course 3, Lesson 1: Computational Thinking (unplugged)		
Course 3, Lesson 2: Maze (plugged)		
Course 3, Lesson 3: Artist (plugged)		
Course 3, Lesson 4: Functional Suncatchers (unplugged)		
Course 3, Lesson 5: Artist Functions (plugged)		
Course 3, Lesson 6: Bee Functions (plugged)		
Course 3, Lesson 7: Bee Conditionals (plugged)		
Course 3, Lesson 8: Maze Conditionals (plugged)		
Course 3, Lesson 9: Songwriting (unplugged)		
Course 3, Lesson 10: Dice Race (unplugged)		
Course 3, Lesson 11: Artist Nested Loops (plugged)		
Course 3, Lesson 12: Farmer While Loops (plugged)		
Course 3, Lesson 13: Bee Nested Loops (plugged)		
Course 3, Lesson 14: Bee Debugging (plugged)		
Course 3, Lesson 15: Bounce (plugged)		
Course 3, Lesson 16: Play Lab Create a Story (plugged)		
Course 3, Lesson 17: Play Lab Create a Game (plugged)		
Course 3, Lesson 18: Internet (unplugged)		
Course 3, Lesson 19: Crowdsourcing (unplugged)		
Course 3, Lesson 20: Digital Citizenship (unplugged)		
Course 3, Lesson 21: Artist Patterns (plugged)		
Scratch Lesson 2: Moving Sprites		
Scratch Lesson 3: Broadcasting a Message		
I did not use any of these Code.org or Scratch Lessons (25)		

27. When using the Code.org, Course 3 or Scratch lessons, approximately what proportion of time did your students spend in each of the following configurations? All percentages must add up to 100%.

If your students worked with a media specialist or other specials teacher, please answer on his/her behalf to the best of your ability.

Whole Class: _______ Small Group: _______ Partner: _______
Independent: _______ Total: ________

28. On average, what proportion of your students did all of the Code.org, Course 3 or Scratch activities you used?

If your students worked with a media specialist or specials teacher, please answer on his/her behalf to the best of your ability.

1) None
2) Some
3) About half
4) Most
5) All

Now we would like to know about any computer science activities you used with your students **during the entire 180-minute Integrated Literacy Block** that were **not identified** in the STEM+C Module.

29. Which additional computer science materials, if any, did you use? Please select all that apply.

    If your students worked with a media specialist or specials teacher, please answer on his/her behalf to the best of your ability.

1) Code.org, Course 1
2) Code.org, Course 2
3) Code.org, Course 4
4) Accelerated Course
5) Code Studio Hour of Code activities
6) Barefoot Computing
7) Khan Academy computer science courses
8) Videos on computational thinking and computer science
9) Google's computational thinking resources
10) Programs that use Blockly programming language
11) Computer science vocabulary
12) Scratch
13) Other (please explain) ____
14) I did not use any additional computer science materials

30. About how many Code.org **Course 1** lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 18

31. About how many Code.org Course 2 lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 19

32. About how many Code.org Course 4 lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 22

Implementation - Instructional Practices (Treatment)

For the following questions, please only think of your teaching during the STEM+C Module lessons.

33. When you taught **STEM+C Module lessons**, about how often did you **explicitly**…

1) Refer to the overarching Module problem
2) Explicitly make lesson content relevant to students (e.g., ask about past experiences, apply content to students' daily lives)
3) Cover standards from multiple subject/content areas in the same lesson
4) Make connections across content areas during discussions with students

34. When you taught STEM+C Module lessons, approximately what proportion of time did your students spend in each of the following configurations? Proportions must add up to 100%

Whole Class: _______ Small Group: _______ Partner : _______

Independent: _______ Total : ________

35. How often did you intentionally and explicitly do the following when students were working in groups or pairs during STEM+C Module lessons? (Scale: Never, Rarely, Sometimes, Often, Always)
    1) Encourage group members to work to solve problems together
    2) Encourage group members to share ideas and strategies respectfully
    3) Encourage group members to listen to what others have to say

36. During STEM+C Module lessons, about how often did you explicitly ask students to do the following?

(Scale: Never, Rarely, Sometimes, Often, Always)

1) Organize, process, manipulate, and/or evaluate data
2) Explain how they solved a problem
3) Consider other students' ideas in comparison to their own
4) Demonstrate reasoning (e.g., noting relationships between concepts, making comparisons)
5) Consider the merits of using different approaches or problem-solving strategies
6) Solve problems in more than one way
7) Communicate their thought process to others

37. During **STEM+C Module lessons**, about how often did you **explicitly ask students** to do the following?

(Scale: Never, Rarely, Sometimes, Often, Always)

1) Answer a question even if they were unsure
2) Try new things even if they might make mistakes
3) hare their ideas even if they were different from others

Support – Implementation

38. Apart from the professional development workshops at the beginning of the school year, did you receive additional support in implementing the STEM+C integrated module? Please select all that apply.
    1) Broward STEM+C Module contacts (i.e., Annmargareth Marousky, Kelly Thomas, and/or Lisa Milenkovic)
    2) Your school's STEM+C Module designee
    3) Media specialist(s)
    4) Other teachers **at your school** implementing STEM+C Module materials
    5) Other teachers **from different schools** implementing STEM+C Module materials
    6) Other teachers **not implementing** STEM+C Module materials

7) PLC facilitator(s)
8) In-service facilitator(s)
9) School leaders (e.g., principal, assistant principal)
10) Canvas website
11) Online discussion boards or forums
12) Other online resources
13) Other (please explain) ____
14) I did not receive any additional support.

Attitudes Toward Interdisciplinary Teaching Practices (Treatment)

Now we're interested in learning about your attitudes toward interdisciplinary teaching practices. The STEM+C Module is an example of interdisciplinary teaching that integrates literacy, science, computer science, and social studies.

39. Prior to teaching the STEM+C Module, how much experience did you have teaching interdisciplinary modules?
   1) None
   2) A little bit
   3) A fair amount
   4) Quite a bit
   5) A lot

For the following questions, please think about interdisciplinary teaching practices **in general**, not just about the STEM+C Module.

40. How much do you agree or disagree with the following statements about your interdisciplinary teaching practices in general? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)
   1) I have sufficient content knowledge across multiple subject areas to be very effective at teaching interdisciplinary modules.
   2) I have nearly every skill I need to teach interdisciplinary modules well at the elementary school level.
   3) I have nearly every skill I need to teach interdisciplinary modules well at the elementary school level.
   4) I am really good at teaching interdisciplinary modules at the elementary school level.
   5) I am effective at making interdisciplinary modules relevant to my students.
   6) I have sufficient access to the hardware and/or software I need to teach interdisciplinary modules effectively

41. How much do you agree or disagree with the following statements about interdisciplinary teaching practices **in general**? Interdisciplinary teaching… (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)
    1) Is more valuable for elementary students' content understanding than teaching each subject separately.
    2) Is more engaging for elementary students than teaching each subject separately.
    3) Helps students understand the relevance and applicability of the subjects they are learning.

42. How much do you agree or disagree with the following statement? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)

    Teaching interdisciplinary modules is a feasible instructional model for elementary classrooms.

43. You said you disagree that teaching with interdisciplinary modules is a feasible instructional model for elementary classrooms. What are the **top three reasons** why you disagreed?
    1) I don't have enough time.
    2) I lack content knowledge across multiple subject areas.
    3) I don't have sufficient access to adequate hardware and/or software.
    4) I am not confident teaching with technology.
    5) I have difficulty managing my students.
    6) I don't have adequate materials.
    7) Other (please explain):____

Leadership Attitudes Toward CS (Treatment)

We're interested in your perception of your school leaders' attitudes towards computer science. Your answers are **confidential** and will not be shared with anyone outside of the research team.

44. How much do you agree or disagree with the following statements?
    My school leaders… (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)
    1) Think it's important for all students to learn computer science in elementary school.
    2) Believe that learning computer science in elementary school will help students get a good job someday.
    3) Communicate the value of computer science to students, parents, teachers, and staff.

Teacher Attitudes Survey (All)

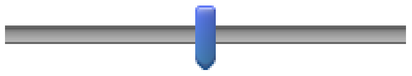
45. How much do you agree or disagree with the following statements about **your** teaching **in general?** (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)
    1) I have nearly every skill I need to teach elementary school well.
    2) I am really good at teaching elementary school.
    3) I am effective at making schoolwork relevant to my students.

46. How much do you agree or disagree with the following statements about your teaching in general? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)
    1) I experiment with new practices all the time.
    2) I am always looking for new ways of doing things in my teaching.
    3) I am constantly the first to try new things in my school.

47. How much do you agree or disagree with the following statements about your teaching in general? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)
    1) I am able to manage the pressure and stress at my school well.
    2) I see difficult tasks through to the end.
    3) I find ways to accomplish my goals.
    4) When planning for my work, I prepare for potential challenges.
    5) I am able to manage my work even when there are unexpected changes and constraints.

After this last set of questions about computer science, you will be done!

48. On a scale of 0-5, how familiar are you with the term "computer science?"

	familiar		ewhat familiar		/ familiar
	0	1	3	4	5
Level of Familiarity ()					

49. If someone asked you what computer science is, what would you say?

50. If you had to pick, which one of the following would you choose to explain what computer science is?
    1) Keyboarding and typing.
    2) Designing websites, apps, and video games.
    3) Studying how computers work.
    4) Fixing computers when they break.
    5) Building computers and other hardware.
    6) Making spreadsheets and graphs.
    7) Using computers to solve complex problems.
    8) Doing science on a computer.
    9) Analyzing data.
    10) Robotics.
    11) Other (please explain):

Thanks for telling us what you think computer science is. For this survey, when we say computer science, we mean:

> Studying and using computers to help solve problems and to create technology, like video games and apps.

51. How much do you agree or disagree with the following statements about your teaching of computer science? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)
    1) I understand computer science concepts well enough to be a very effective teacher of computer science at the elementary school level.
    2) I have nearly every skill I need to teach computer science well at the elementary school level.
    3) I am really good at teaching computer science at the elementary school level.
    4) I am effective at making computer science relevant to my students.

52. How much do you agree or disagree with the following statements about computer science? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)
    1) I enjoy computer science
    2) I am interested in computer science
    3) I like doing computer science.

53. What part of computer science do you like the most? Please select only one answer.

1) Designing websites, apps, and/or video games.
2) Computer programming and coding
3) Data analysis
4) Robotics
5) Fixing computer hardware problems
6) Fixing computer software problems
7) Studying how computers work
8) Building computers and other hardware
9) Learning new computer programming languages (e.g. Python, C++, Java)
10) Other (please explain):

54. How much do you agree or disagree with the following statements regarding learning computer science in elementary school? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)

1) Is just as important for students to learn as non-core subjects, like art, music, and foreign language.
2) Is just as important for students to learn as core-subjects, like math, science, and English/Language Arts.
3) Is not developmentally-appropriate for elementary school children.
4) Will ensure students are prepared for future academic success.
5) Will ensure students get good jobs in the future.

55. How much do you agree or disagree with the following statements regarding learning computer science in elementary school? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)

Learning computer science in elementary school can help students learn how to...

1) Consider alternative approaches to their work.
2) Analyze (organize, process, manipulate, evaluate) data.
3) Problem solve when something doesn't work the way they want it to work.

56. How much do you agree or disagree with the following statements regarding learning computer science in elementary school? (Scale: Strongly Disagree, Disagree, Somewhat Disagree, Somewhat Agree, Agree, Strongly Agree)

Learning computer science in elementary school can help students learn how to...

1) Explain the logic and reasoning supporting their solutions.
2) Explain why they agree or disagree with the work of other students.
3) Communicate their thought processes to others.

School Grade (Control)

57. What grade(s) do you teach? Please select all that apply.

1) 3rd grade

2) 4th grade
3) 5th grade

Essential Program Elements (Control)

First, we'd like to ask you some questions about Achieve3000.
As part of the STEM+C Integrated Literacy Block project, you were asked to administer Achieve3000 diagnostic assessments to students. We'd like to know about your students' use of other (**non-assessment**) Achieve3000 materials and activities (e.g., nonfiction texts from Achieve3000).

58. About how many times did you have your students use Achieve3000 (non-assessment) materials and activities?

0 5 10 15 20 25 30 35 40 45 50

Number of times ()	

59. When students used Achieve3000 (non-assessment) materials and activities, approximately how much time did they spend each time they used it? Please include time spent at school and when assigned for homework.

0 15 30 45 60 75 90

Number of minutes ()	

Now we are interested in learning about your use of Code.org's computer science activities. This includes the Computer Science Fundamentals Courses 1-4, the Accelerated Course, and the eight Code Studio Hour of Code activities (e.g., Play Lab, Artist, Frozen).

60. Since the start of the school year, which Code.org materials, if any, have you used with your students? Please select all that apply.
1) Course 1
2) Course 2
3) Course 3
4) Course 4
5) Accelerated Course
6) Hour of Code activities
7) I did not use any Code.org materials with my students.

A Code.org "lesson" may be either an unplugged lesson, or a "stage" (that is composed of a series of puzzles).

61. About how many Code.org **Course 1** lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 18 (18)

62. About how many Code.org Course 2 lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 19 (19)

63. About how many Code.org Course 3 lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 21 (21)

64. Q112 About how many Code.org Course 4 lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 22 (22)

65. About how many Code.org Accelerated Course lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 20 (20)

66. About how many Code.org Hour of Code lessons (unplugged or stages) did most/all of your students complete?

▼ 1 (1) ... 8 (8)

67. Now we would like a little more specific information on Course 2 lessons only. Which Course 2 lessons did all/most of your students complete?

1) Course 2, Lesson 1: Graph Paper Programming (unplugged)
2) Course 2, Lesson 2: Real-Life Algorithms (unplugged)
3) Course 2, Lesson 3: Maze Sequence (plugged)
4) Course 2, Lesson 4: Artist Sequence (plugged)
5) Course 2, Lesson 5: Getting Loopy (unplugged)
6) Course 2, Lesson 6: Maze Loops (plugged)
7) Course 2, Lesson 7: Artist Loops (plugged)
8) Course 2, Lesson 8: Bee Loops (plugged)
9) Course 2, Lesson 9: Relay Programming (unplugged)
10) Course 2, Lesson 12: Conditionals with Cards (unplugged)
11) Course 2, Lesson 13: Bee Conditionals (plugged)
12) Course 2, Lesson 14: Binary Bracelets (unplugged)
13) Course 2, Lesson 15: The Big Event (unplugged)
14) Course 2, Lesson 16: Flappy (plugged)
15) Course 2, Lesson 17: Play Lab: Create a Story (plugged)
16) Course 2, Lesson 18: Your Digital Footprint (unplugged)
17) Course 2, Lesson 19: Artist: Nested Loops (plugged)

68. Please select any other (not from Code.org) computer science activities that you used in the last semester.

1) Barefoot Computing

2) Khan Academy computer science courses
3) Videos on computational thinking and computer science
4) Google’s computational thinking resources
5) Programs that use Blockly programming language
6) Scratch
7) Other (please explain): ____
8) I did not use any other computer science-related activities.

Implementation - Instructional Practices (Control)

69. For the following questions, please **only think** of your teaching from the **past 12 weeks**.

    Over the **past 12 weeks**, about how often did you… (Scale: Never, Rarely, Sometimes, Often, Always)

    1) Explicitly make lesson content relevant to students (e.g., ask about past experiences, apply content to students’ daily lives)
    2) Cover standards from multiple subject/content areas in the same lesson
    3) Make connections across content areas during discussions with students

70. Over the **past 12 weeks**, approximately what proportion of time did your students spend in each of the following configurations? All proportions must add up to 100%.
    Whole Class: _______ Small Group: _______ Partner: _______
    Independent: _______ Total: ________

71. Over the past 12 weeks, about how often did you intentionally and explicitly do the following when students were working in groups or pairs? (Scale: Never, Rarely, Sometimes, Often, Always)
    1) Encourage group members to work to solve problems together
    2) Encourage group members to share ideas and strategies respectfully
    3) Encourage group members to listen to what others have to say

72. Over the **past 12 weeks**, about how often did you **explicitly ask students** to do the following? (Scale: Never, Rarely, Sometimes, Often, Always)
    1) Organize, process, manipulate, and/or evaluate data
    2) Explain how they solved a problem
    3) Consider other students' ideas in comparison to their own
    4) Demonstrate reasoning (e.g., noting relationships between concepts, making comparisons)
    5) Consider the merits of using different approaches or problem-solving strategies
    6) Solve problems in more than one way
    7) Communicate their thought process to others

73. Over the **past 12 weeks**, about how often did you **explicitly ask students** to do the following? (Scale: Never, Rarely, Sometimes, Often, Always)
    1) Answer a question even if they were unsure
    2) Try new things even if they might make mistakes
    3) Share their ideas even if they were different from others'

Attitudes Toward Interdisciplinary Teaching Practices (Control)

Now we're interested in learning about your attitudes toward interdisciplinary teaching practices. Here is an example of what we mean by "interdisciplinary teaching practices":

You teach a 7-week module on North American Regions by:
-integrating reading, researching, writing, and presenting about a North American region (ELA);
-studying the geography and natural/man-made landmarks of a region (social studies);
-researching and studying the climate, vegetation, and natural resources of the region (science);
-and using models/simulations to create a map of the region (computer science).

74. How much experience do you have teaching interdisciplinary modules like the one described in the example above?
    1) None
    2) A little bit
    3) A fair amount
    4) Quite a bit
    5) A lot

75. How much do you agree or disagree with the following statements about interdisciplinary teaching practices?
    1) I have enough time to teach interdisciplinary modules.
    2) I have sufficient content knowledge across multiple subject areas to be very effective at teaching interdisciplinary modules.
    3) I have nearly every skill I need to teach interdisciplinary modules well at the elementary school level.
    4) I am really good at teaching interdisciplinary modules at the elementary school level.
    5) I am effective at making interdisciplinary modules relevant to my students.

76. How much do you agree or disagree with the following statements about interdisciplinary teaching practices?

    1) I have sufficient access to the hardware and/or software I need to teach interdisciplinary modules effectively.

77. How much do you agree or disagree with the following statements about interdisciplinary teaching practices? Interdisciplinary teaching…

1) Is more valuable for elementary students' content understanding than teaching each subject separately.
2) Is more engaging for elementary students than teaching each subject separately.
3) Helps students understand the relevance and applicability of the subjects they are learning.

78. How much do you agree or disagree with the following statement?
1) Teaching interdisciplinary modules is a feasible instructional model for elementary classrooms.

79. You said you disagree that teaching with interdisciplinary modules is a feasible instructional model for elementary classrooms. What is the main reason you disagreed? Please select only one.
1) I don't have enough time.
2) I lack content knowledge across multiple subject areas.
3) I don't have sufficient access to adequate hardware and/or software.
4) I am not confident teaching with technology.
5) I have difficulty managing my students.
6) I don't have adequate materials.
7) Other (please explain):

Leadership Attitudes Toward CS (Control)

80. We're interested in your perception of your school leaders' attitudes toward computer science. Remember, **your answers are confidential** and will not be shared with anyone outside of the research team. How much do you agree or disagree with the following statements? My school leaders…
    1) Think it's important for all students to learn computer science in elementary school.
    2) Believe that learning computer science in elementary school will help students get a good job someday.
    3) Communicate the value of computer science to students, parents, teachers, and staff.

Appendix B

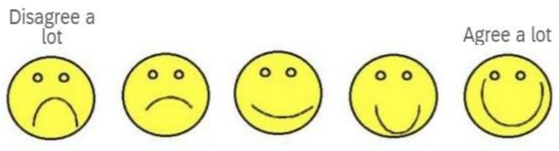
Time4CS Student Questionnaire Items

School Background Questions

1. What school do you go to? [drop-down menu listing schools]
2. What is your teacher's last name? [drop-down menu listing teachers' names and a free-response option if students did not see their teacher's name]

General School Affinity

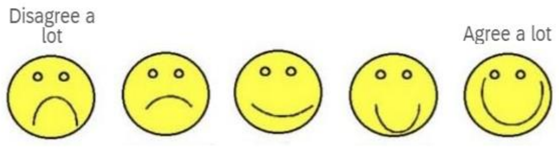
How much do you agree or disagree with the following statements about school?



1. I like schoolwork.
2. My schoolwork is interesting.
3. I enjoy doing my schoolwork.

General School Self-Efficacy

How much do you agree or disagree with the following statements about school?



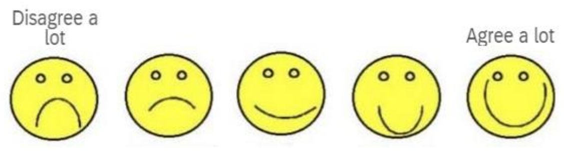
1. I have the ability to do my schoolwork.
2. I am better at schoolwork than most of the other kids at my school.
3. I am very good at doing my schoolwork.
4. I can figure out how to solve the most difficult problems in school if I try.

Knowledge of Computer Science

1. Have you ever heard of "computer science" before? [Yes/No]
2. If someone asked you what computer science is, what would you say? [Free-response]
3. Which one of the following would you choose to explain what computer science is? [Single response choice]
   3.1. Computer science is…
      3.1.1. Being good at computers.
      3.1.2. Doing science on a computer.
      3.1.3. Engineering.
      3.1.4. What is inside a computer?
      3.1.5. Learning how computers work.
      3.1.6. Building computers.
      3.1.7. Making websites, apps, and video games.
      3.1.8. Studying computers.
      3.1.9. Using computers to solve problems.
      3.1.10. Computer programing and coding.
      3.1.11. Typing.
      3.1.12. Robotics.
      3.1.13. Other (please explain) [Free-response]
   3.2. Prompt: Thank you for telling us what you think computer science is. For this survey, when we say computer science, we mean: Studying and using computers to help solve problems and to create technology, like video games and apps.

Computer Science Affinity

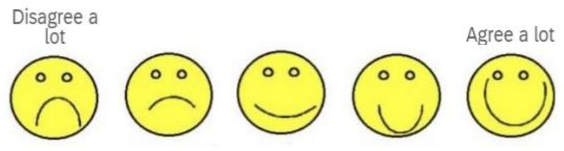
How much do you agree or disagree with the following statements about school?



1. I like computer science.
2. I think computer science is interesting.
3. I want to learn more computer science.
4. I like doing computer science activities.

Computer Science Self-Efficacy

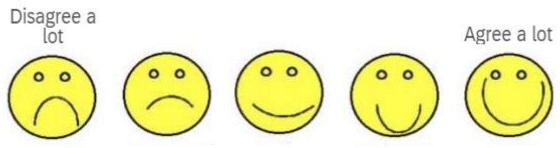
How much do you agree or disagree with the following statements about school?



1. I have the ability to learn computer science.
2. I am better at computer science than most of the other kids at my school.
3. I am very good at computer science.
4. I can figure out how to solve hard problems in computer science if I try.

Computer Science Identity

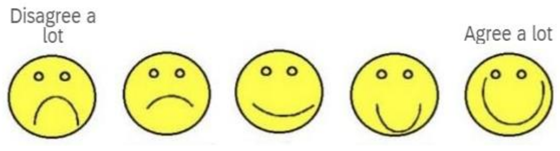
How much do you agree or disagree with the following statements about school?



1. Kids like me do computer science.
2. I like computer science more than other kids at my school.
3. I think I could become a computer scientist one day.
4. I do computer science in my free time.

Computer Science Utility

How much do you agree or disagree with the following statements about school?



1. It’s important for me to learn computer science.
2. Computer science is useful for me to learn.
3. Learning computer science can help me get a job someday.

Prior Computer Science Experience

1. Have you ever done robotics before? [Yes/No]
2. Have you done activities on Code.org before [Yes/No]

Demographics

Now we’d like to ask you some questions about you?

1. Are you a boy or a girl? [Boy/Girl]
2. Are you Hispanic/Latino? [Yes/No]
3. What is your ethnicity? [Select all that apply]
   - 3.1. African American
   - 3.2. Asiana
   - 3.3. White
   - 3.4. Other
4. How old are you? [Drop-down menu listing age choices]

Note.

1. Researchers should contact the authors prior to use of items from this questionnaire or the questionnaire in its entirety. All requests can be sent to Jeanne Century at outlieruchicago@gmail.com

2. Researchers planning to utilize items from this questionnaire, or this questionnaire in its entirety, should use the following citation in their reference list.

Century, J., Ferris, K. A., & Zuo, H. (2020). Finding time for computer science in the elementary school day: a quasi-experimental study of a transdisciplinary problem-based learning approach. *International Journal of STEM Education, 7,* 1-16.

International Journal of Computer Science Education in Schools, August 2021, Vol. 5, No. 1
ISSN 2513-8359

Assessment of Eighth Grade Students' Domain-General Computational Thinking Skills

Halit Karalar[1]
Muhammet Mustafa Alpaslan[1]

[1]Muğla Sıtkı Koçman University

DOI: 10.21585/ijcses.v5i1.126

Abstract
The aim of this study was to assess the domain-general CT skills of 8th grade students in Turkey. In the study, first, the domain-general CT scale was adapted to Turkish, and then relations of the CT skills with gender, computer possession, internet access, and programming experience were examined. This survey research was conducted with the 284 eighth grade students. The data were analyzed through confirmatory factor analysis, independent sample t-test and Pearson correlation test. The results of the confirmatory factor analysis and Cronbach alpha showed that the Turkish version of the domain-general CT scale was valid and reliable. T-test results showed no significant difference in the CT skills of the students according to gender, computer possession and internet access. A statistically significant difference in algorithm, evaluation, generalization, and general CT skills was found between students who learned programming and those who did not. Correlational tests revealed that there was a positive and significant relationship between the programming experience of students who learn programming and their CT skills. As students' programming experience increased, their CT skills also increased. The results of the research were discussed, and recommendations for policymakers and implementers were included.

Keywords: Computational thinking, domain-general computational thinking, scale adaptation, middle school students

1. Introduction

It is important for students to have Computational Thinking (CT) skills in order to be successful in professions in today's digital world (Vallance & Towndrow, 2016). CT plays the role of a bridge between the fields of science, technology, engineering, and mathematics (STEM), which are the main source of innovation of 21st century digital economies, and computer science, and is an important skill for the development of computer literacy and thinking skills including problem solving (Barr & Stephenson, 2011). Since CT requires deep learning compared to superficial or rote learning (Wing, 2006), it can be transferred to other areas and problem-solving contexts. In this respect, it is argued that CT is a necessary skill not only for STEM professions but also for all other professions (Denning, 2017). What is more, CT, which is one of the skills sets that all students should have such as reading, writing and mathematics in the current century, has become a necessary skill not only for computer science but also for all fields of learning (Wing, 2006).

It is necessary to integrate CT into education systems in order to provide students with the skills they will need in the future (Grover & Pea, 2013). For this reason, CT has been included in the Next Generation Science Standards (NGSS) in the USA and integrated into STEM curricula at K12 levels (Tang et al., 2020). In parallel with these developments, CT has been integrated into education programs in many countries including Finland, Norway, South Korea, Israel, Poland, New Zealand, Estonia (Tikva & Tambouris, 2021). Studies examining CT studies published between 2006 and 2018 reported that CT was integrated into primary school curricula in 52 countries (Tang et al., 2020).

Although there is no common ground on the definition of CT, which has attracted the attention of educators for the last 15 years, it is widely accepted that CT is a thinking skill that includes a cognitive problem-solving process that can be developed in many ways, not only through computer programming (Aho, 2012; Guzdial, 2008; Selby & Woollard, 2013; Shute et al., 2017; Tsai et al., 2021; Wing, 2011; Yadav et al., 2016). Similarly, Denning (2017) argues that CT is not only the way computer scientists think, but also a necessary thinking skill for other fields. Shute et al. (2017) argued that CT can be defined as “the conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that were reusable in different contexts”. Previous studies have examined students' domain-specific CT skills (computer science or programming) in different contexts. (Erümit et al., 2020; Günbatar, 2020; Korucu et al., 2017; Oluk et al., 2018; Oluk & Korkmaz, 2016; Wu & Su, 2021) However, little is known about how the programming skills acquired from different programming environments change their domain-general CT skills; whether there is a difference between the domain-general CT skills of students who gain programming skills and those who do not especially in middle school. For this reason, two purposes were adopted in the study. First, due to lack of domain-general CT scale in Turkish version, it was aimed to adapt the “Computational Thinking Scale for Computer Literacy Education” (CTSCLE) developed by Tsai et al. (2021) to Turkish. Secondly, it was aimed to assess students' domain-general CT skills. The following questions were sought in the study:

- What is the reliability and validity of the domain-general CT scale?
- Is there a significant difference between students' domain-general CT skills and gender?
- Is there a significant difference between students' domain-general CT skills and programming experiences?
- Is there a significant difference between students' domain-general CT skills and computer possession?
- Is there a significant difference between students' domain-general CT skills and internet access?
- Is there a significant relationship between students' domain-general CT skills and programming experiences?

1.1 CT Definition

There is no consensus definition on CT. Although the first emergence of CT was based on the book "Mindstorms: Children, computers, and powerful ideas" written by Papert in 1980 (Zhang & Nouri, 2019), it was Wing (2006) who was first to introduce the concept of "Computational Thinking". He stated that CT includes problem solving, system design and understanding human behavior by using concepts related to computer science. Wing later changed the definition to highlight that CT is a thinking process, and CT is defined as a process that involves presenting problems and their solutions in a form that can be effectively processed by an information-processing agent (Wing, 2011). Later, Aho (2012) stated that the concept of formulating the problems emphasized in this definition includes the algorithm. According to Aho's definition, CT is emphasized as a thinking process in which the solution of its problems can be presented through algorithms containing sequential steps. In the functional definition developed by the Computer Science Teachers Association (CSTA) and the International Society of Technology in Education (ISTE), it was emphasized that CT is a thinking skill consisting of but not limited to 6 skill sets (ISTE & CSTA, 2011). These skill sets: (1) Formulate problems so that they can be solved by information processing units. (2) Organize and analyze data rationally. (3) Presenting data through abstractions such as simulations and models. (4) Automating problem-related solutions using algorithmic thinking. (5) Identifying, organising, and implementing appropriate solutions in the most efficient and effective way. (6) Transferring the solution of the problem to other problems, making generalisations. Although the definitions differ, it can be stated that CT is a thinking process that includes problem solving and consists of a set of skill sets obtained by modeling the mental processes that a computer scientist uses while solving the problem.
CT is an umbrella concept that includes many sub-skill sets. In the literature, different classifications are made regarding the sub-skill sets included in CT (Barr & Stephenson, 2011; Brennan & Resnick, 2012; Denner et al., 2012; Grover & Pea, 2013; ISTE & CSTA, 2011; Kalelioglu et al. al., 2016; Selby & Woollard, 2013). As stated by Tsai et al. (2021), approaches other than Selby and Woollard (2013) include sub-skills specific to computer science or programming, making it difficult to measure domain-specific CT skills. The Selby and Woollard (2013) approach, which includes domain-specific CT sub-skills, consists of five sub-skills: abstraction, decomposition, algorithmic thinking, evaluation, and generalization. Abstraction refers to a mental process that involves focusing on the main points rather than the details to solve a problem. Decomposition states a mental process that involves breaking down the problems into small and manageable parts in order to solve them. Algorithmic thinking refers to a mental process in which it is determined how to solve a problem step by step. Evaluation is a mental process that involves determining the best solution from different solutions to a problem,

taking into account the resources. Generalization states a mental process that involves determining how to solve certain problems and applying them in solving other similar problems.

1.2 CT Assessment Tools

Due to the lack of a commonly accepted definition of CT, different approaches have emerged regarding how to measure CT skills. Measurement tools developed for these approaches are based on Román-González et al. (2019) under seven titles. CT diagnostic tools are the tools aiming to measure the CT capability level. CT result evaluation tools are the tools that aim to measure whether the learners have gained sufficient content knowledge thanks to a training on CT skills. CT process assessment tools are intended to provide feedback to the student, which are usually provided by automated means to develop and increase learners' CT skills. CT data mining tools are to aim to provide feedback to students by applying data mining on students' interaction records in online environments. CT skill transfer tools are intended to measure whether students are able to transfer their CT skills to different types of problems, contexts, and situations. CT perception and attitude tools are tools that aim to measure learners' perceptions and attitudes towards CT and related subjects (computer, computer science, programming, digital literacy, etc.). CT word evaluation tools are amied to measure CT dimensions and components, which are emphasized as CT language and can be expressed verbally. Tsai et al. (2021) emphasized that all of the above evaluation approaches are specific to computer science or programming, and that there is a need for domain-general CT measurement tools. Similarly, Alsop (2019) suggested that children's learning of CT skills cannot be measured by focusing only on programming constructs (CT concepts), it should also include learning behavior and metacognitive practices.

In the literature, there are several Turkish instruments to assess students' CT skills. However, these scales are domain dependent scales that are specific to programming, or they include 21st century skills. For example, "Computational Thinking Levels Scale" developed by Korkmaz, Çakır, and Özden (2015, 2017) for university and middle school students includes five skills such as "algorithmic thinking", "problem solving", "critical thinking", "creative thinking" and "cooperativity". However, this scale does not include the core CT skills such as abstraction and decomposition stated by Selby and Woodland (2013). The other available scale for assessing CT skills of Turkish middle school students is "Self-Efficacy Perception Scale for Computing Thinking Skills" developed by Gülbahar, Kert, and Kalelioğlu (2019). This scale includes factors related to programming, which make it a domain-specific scale. Since CT skills are problem-solving-based thinking skills that are necessary to not only computer science but also other fields including science, mathematics etc., there is a need for domain-general (non-computer science or programming-specific) Turkish scales. In this study, it was aimed to adapt the CTSCLE scale developed by Tsai et al. (2021) into Turkish in order to provide a domain-general CT skill scale.

1.3 Factors Related to CT Skills

Studies addressing students' CT skills have associated the different factors on students' CT skills. Some studies focused on effect of gender on CT skills. The effects of gender on CT skills are controversial. Some studies reported that there was no difference between CT skills mean score of male and female students. For instance, Alsancak (2020) examines the gender difference on CT skills with 722 Turkish secondary school students. T-test results showed that there was no significant difference between female and male students' CT skills mean score, on the one hand, (t (719) = -.98, *p* = .33). On the other hand, Gulbahar et al. (2020) reported that there was significant difference between female and male CT skills in favor of female students (t (43748) = 7.42, *p* <.01). Therefore, there is a need to test the relations between gender and CT skills measured in the CTSCLE.
Other studies addressing CT skills mostly focused on effects of having a computer, Internet access, mobile device, or other technological tools, or programming experience on CT skills. For instance, Oluk and Korkmaz (2016) investigated that the relations of CT skills with Scratch programming skills and computer use. They found that Scratch programming skills were highly correlated with the CT skills (Spearman rho= .93, *p* <.01) whereas no relation between computer use and CT skills (U = 102, *p* = .74). In another study, Alsancak Sarıkaya (2019) reported that students did not differ based on their computer experience (F (5,253) = 2.25 *p* = .05) but they did in internet use (F (5,253) = 2.27 *p* = .048). In another study, İbili et al. (2020) reported that CT skills did not differ based on students' programming experience (F (2, 588) = 1.423, *p* >.05). These findings of aforementioned studies indicate that relations between programming experience and CT skills are controversial. Thus, it would be important to examine the links of these variables with the CT skills by utilizing a domain-general CT skills scale, the CTSCLE, to better understand the role of computer related experiences on CT skills.

2. Method

2.1 Research Method

In this study, one of the quantitative research models, the survey model was used. Survey model is based on collecting data from the sample of the population and interpreting these data in order to determine the characteristics of a population or to reveal the existing situation (Fraenkel et al., 2012). This method was chosen because the aim of this study is to examine the domain-general CT skills of middle school students. In addition, comparative analysis of CT skills according to various variables were made using the relational research method. Relational research model is a method used to examine the relationships between variables (Fraenkel et al., 2012).

2.2 Participants

The population of this study is 8th grade students in Turkey. Since collecting data from the entire population is not possible in terms of economy and time, the researcher can define an accessible population. Therefore, one of Turkey's western provinces where it could be easily accessed by researchers was determined to be the accessible population. Six public schools (3 rural and 3 urban areas) in this province were randomly determined. 284 middle school students studying in these schools voluntarily participated in the study. Characteristics of the participants were given in Table 1.

Table 1. Demographic characteristics of the participants

Demographic feature		Frequency	Percent
Gender			
	Boy	137	48.2
	Girl	147	51.8
Computer possession			
	No	103	36.3
	Yes	181	63.7
Internet access			
	No	23	8.1
	Yes	261	91.9
Programming environments experienced (multiple choice be selectable)			
	Scratch	152	0.54
	Code.org	103	0.36
	Arduino	22	0.08
	Other (Google CS First, Hour of Code, Blockly.games, CodeMonkey, Codecademy, Codesters etc.)	78	0.27
	No programming experience	60	0.21

2.3 Data Collection Tools

The data collection tool in this study consisted of two parts. In the first part, the personal information form aimed at collecting the students' demographic information (gender, etc.) was included and in the other part, the Turkish version of the CTSCLE was placed.

The CTSCLE was developed by Tsai et al. (2021) with 19 items in five factors (abstraction, decomposition, algorithmic thinking, evaluation, and generalization). The abstraction, algorithmic thinking, evaluation, and generalization sub-dimensions of the scale consist of four items and the decomposition sub-dimension consists of three items. The scale was originally administrated with 388 students (255 boys, 132 girls, mean age 14.58) studying at a secondary school in Taiwan. A 5-point Likert-type scale was used in the form of Strongly Disagree (1) and Strongly Agree (5). Tsai et al. (2021) found that 19 items were grouped under five factors and the total variance explained by 19 items was 64.03%. It was reported that the Alpha reliability coefficient value of the whole scale was 0.91 and varied between 0.74 and 0.83 in the sub-dimensions (See Table 2). For the discrimination validity of the scale, the AVE (average variable extracted) values of each sub-dimension were examined and whether the square roots of the AVE values were greater than the correlation values between the sub-dimensions. Since the AVE values of the scale sub-dimensions were greater than 0.50 (0.74 - 0.83) and the square roots of the AVE values of each sub-dimension were larger than the correlation values in the related sub-dimensions, it was concluded that the scale provides the decomposition validity. The reliability and validity analysis showed that the scale was reliable and valid in Taiwanese sample.

Table 2. Sample example of sub-dimensions in the CTSCLE and their Cronbach values

Sub-dimension	Example item	Item number	Cronbach alpha*
Abstraction	I usually try to analyze the common patterns of different problems.	4	.81
Decomposition	I usually think about how to split a big problem into several small ones.	3	.74
Algorithmic thinking	I usually try to lay out the steps of a solution.	4	.77
Evaluation	I usually try to find the most effective solution for a problem.	4	.83
Generalization	I usually think about how to apply a solution to other problems.	4	.75

* Cronbach alpha values reported by Tsai et al. (2021).

2.4 Process and Data Analysis

Turkish version of the CTSCLE was prepared by following steps suggested by Çapık et al. (2018). First, the permission was granted from the authors who developed the scale. Later, two doctoral experts who graduated from the United States translated the scale into Turkish. In the translation form English to Turkish, semantic and conceptual deduction were made. Next, the opinions of three researchers from the departments of Turkish Language Education, Computer Education and Instructional Technologies, and English Language Education were taken about the semantic translation. According to their opinions, revisions were made. Then, back-translation to English was made by two independent experts and compared with the original version of the scale to ensure that each version was semantically compatible with each other. Later, two information technologies in-service teachers checked the Turkish scale to determine its comprehensibility by the target audience. After taking the opinions of the information technologies teachers, the Turkish version of the scale was finalized.
More than one statistical technique was used in data analysis. First, outliers and missing data were checked. Then, the normality values of the data were examined. Since the skewness and kurtosis values were in the range of -1 and +1, the data were accepted as having a normal distribution (Hair et al., 2019). Next, Cronbach alpha coefficient was calculated to determine the level of reliability. In the literature, data with a Cronbach alpha value above .70 can be considered reliable (Hair et al., 2019). In order to examine the factor structure, measurement model or confirmatory factor analysis (also known as the measurement model or confirmatory factor analysis [CFA]), which is one of the structural equation model methods, was performed in the MPlus 7.0 program. CFA, a statistical technique that tests the construct validity of the scale, was used to test how fit the data obtained to the previously determined model. CFA is recommended to use when the factor structures are pre-determined. Because the factor names and structures had been already known in the study, CFA was used to confirm that. The maximum likelihood method was used as the estimation method in the CFA because the normal distribution assumptions were not severely violated. Parallel to the literature, good fit values were accepted as Chi-square / df ratio less than 3, RMSEA value lower than .06 and CFI value greater than .95 (Hu & Bentler, 1999). In addition, t-test and one-way ANOVA tests were used to test the difference according to demographic variable groups, since the data showed normal distribution. Lastly, Pearson correlation test was used to test the relationship between the programming environments experienced and their domain-general CT skills.

3. Results

3.1 Reliability and Validity of the Adapted CTSCLE

In this study, the validity and reliability analysis of the scale were done before interpreting the data. First, Cronbach's alpha value was calculated for each sub-dimension for reliability. In Table 3, Cronbach's alpha values for each sub-dimension were given along with descriptive statistics. Since the Cronbach alpha values were higher than the acceptable level of .70, the scale was at a reliable level.

Table 3. Descriptive statistics

	Mean	SD	Skewness	Kurtosis	Alpha
Abstraction	3.49	0.95	-0.35	-0.33	.86
Decomposition	3.37	1.01	-0.36	-0.28	.86
Algorithmic Thinking	3.75	0.99	-0.76	0.20	.90
Evaluation	3.83	0.99	-0.80	0.29	.89
Generalization	3.62	0.91	-0.51	0.25	.82
Total	3.61	0.85	-0.70	0.53	.96

Results of CFA showed that the chi-square value was found to be 325.25, χ2(142) = 325.37. The chi-square / df ratio was calculated as 2.29. Although this ratio is lower than the 3.00 value accepted in the literature, the chi-square value is a statistic that is affected by the number of people in the sample. For this reason, other fit values are recommended to be checked.

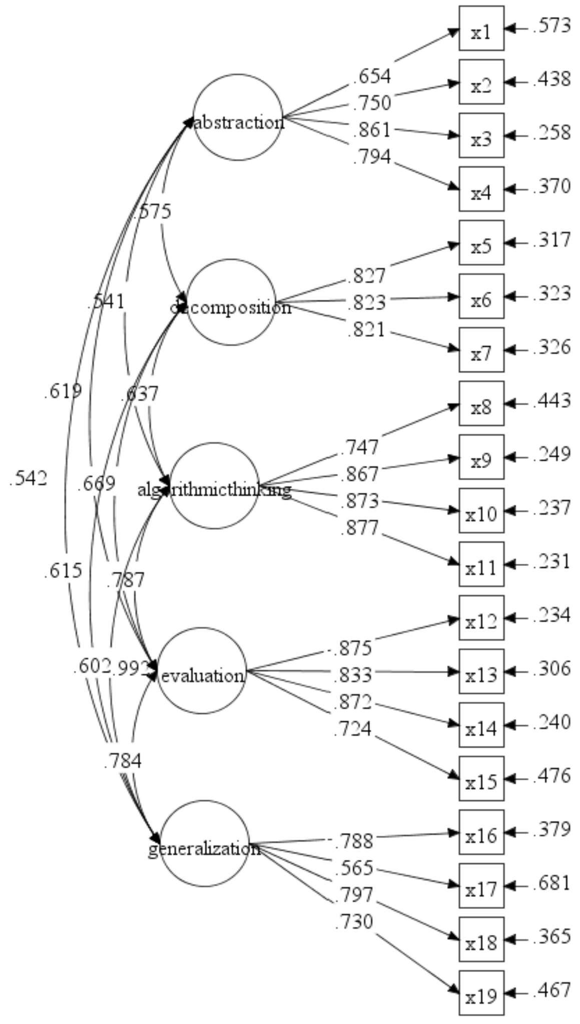


Figure 1. Factor loadings of the CTSCLE

The most accepted and more reliable (robust) fit indices in the literature are generally RMSEA, TLI and CFI values. The RMSEA value was found as .057 CI (.047, .067). This value is smaller than the limit value .06 and indicates a good fit. In addition, CFI and TLI values were calculated as .95 and the limit value was around .95. SRMR value was found as .035. These values showed that the factor structure of the scale was in good fit with the data and accepted as evidence for the construct validity of the scale. In addition, factor loadings higher than .30 indicated that each item was related to the factor (Figure 1). Finally, there was a high level of correlation between the factors of the scale. This shows that the structure is similar. After testing the validity and reliability of the scale, the average value for each factor was calculated and given in Table 3. According to these average values, it showed that students had intermediate level skills in abstraction, decomposition and generalization sub-dimensions, and high skill level in algorithm and evaluation sub-dimensions (low level: 1.00-2.33, medium level: 2.34-3.66 and high level: 3.67- 5.00). While the lowest mean score was in the decomposition dimension, the highest one was in the evaluation sub-dimension.

3.2 Students' CT Skills and Gender

It was found for females in the highest mean score in evaluation and the lowest one was in the decomposition sub-dimension. Similar findings were observed for male students. Whether the students' CT score changed according to their gender was tested with the independent sample t-test. Although female students had a higher average than boys in all sub-dimension and total, results of the t-test showed that these differences were not statistically significant (See Table 4).

Table 4. Gender comparison on the domain-general CT skills

	Boy (*n* = 137)		Girl (*n* = 147)			
	Mean	*SD*	Mean	*SD*	*t*	Cohen's *d*
Abstraction	3.45	0.98	3.53	0.92	0.74	0.08
Decomposition	3.31	1.05	3.42	0.96	0.90	0.11
Algorithmic Thinking	3.66	1.01	3.83	0.96	1.48	0.17
Evaluation	3.78	1.04	3.88	0.94	0.90	0.10
Generalization	3.57	1.00	3.66	0.81	0.84	0.10
Total	3.55	0.91	3.67	0.80	1.10	0.13

Note. *df* = 282.

3.3 Students' CT Skills and Programming Experiences

The CT skills of students who had never learned programming and those who have taken any programming course were compared in order to test the effect of students' programming experiences on their CT skills. According to the results of the independent sample t-test, there was a significant difference in some sub-dimensions while not in others (See Table 5). A statistically significant difference was found in favor of students taking programming language in algorithm, evaluation, generalisation and total averages. The highest significant difference was in the evaluation sub-dimension (Cohen's *d* = .37), while the lowest significant difference was calculated in the total mean score (Cohen's *d* = .30). These findings showed that Algorithmic Thinking, Evaluation and Generalization skills can be gained with programming skills while Abstraction and Decomposition skills can be improved without programming experiences.

Table 5. Students' programming experiences comparison on the CT skills

	Yes (*n* = 224)		No (*n* = 60)			
	Mean	*SD*	Mean	*SD*	*t*	Cohen's *d*
Abstraction	3.53	0.92	3.32	1.04	1.54	.21
Decomposition	3.39	0.97	3.29	1.13	0.68	.09
Algorithmic Thinking	3.83	0.92	3.46	1.16	2.59 *	.35
Evaluation	3.91	0.94	3.53	1.10	2.68 *	.37
Generalization	3.68	0.87	3.38	1.01	2.32 *	.32
Total	3.67	0.81	3.40	1.00	2.21 *	.30

Note. * *p* < .05. *df* = 282.

3.4 Students' CT Skills and Computer Possession

CT skills of students who had a computer and those who did not was compared by utilizing independent sample t-test. Descriptive statistics and results of t-tests were given in Table 6. Students who had a personal computer rated the highest mean value on the evaluation sub-dimension whereas the lowest one was the decomposition. Similarly, students who did not have a personal computer did the highest mean value on the evaluation and the lowest was on the decomposition. Results of t-tests yielded no significant difference between students who had a personal computer or not on their CT skills.

Table 6. Students' computer possession comparison on the CT skills

	Yes (*n* = 181)		No (*n* = 103)			
	Mean	*SD*	Mean	*SD*	*t*	Cohen's *d*
Abstraction	3.46	0.97	3.54	0.92	.71	.08
Decomposition	3.33	1.05	3.43	0.92	.80	.09
Algorithmic Thinking	3.73	1.00	3.79	0.98	.52	.05
Evaluation	3.84	1.00	3.82	0.97	.14	.02
Generalization	3.63	0.94	3.60	0.85	.26	.03
Total	3.60	0.87	3.64	0.82	.38	.03

Note. df = 282.

3.5 Students' CT Skills and Internet Access

CT skills of students who had internet access and those did not was compared by utilizing Welch's t-tests because of extremely unequal sample size across the groups. Descriptive statistics and results of t-tests were given in Table 7. Both student groups rated the highest mean value on the evaluation sub-dimension whereas the lowest one was the decomposition. Results of Welch's t-tests yielded no significance difference between students who had internet access or not on their CTs.

Table 7. Students' internet access comparison on the CT skills

	Yes (*n* = 261)		No (*n* = 23)			
	Mean	*SD*	Mean	*SD*	*t*	Cohen's *d*
Abstraction	3.49	0.94	3.46	1.06	.17	.03
Decomposition	3.38	1.00	3.26	1.09	.53	.11
Algorithmic Thinking	3.78	0.97	3.40	1.09	1.76	.37
Evaluation	3.85	0.98	3.62	1.07	1.08	.22
Generalization	3.62	0.90	3.58	0.99	.23	.03
Total	3.62	0.84	3.46	0.98	.87	.18

Note. df = 282.

3.6 The Relationship between Students' CT Skills and Programming Experiences

Students can take computer and programming courses as elective or compulsory courses at different grade levels and generally learn programming in Code.org, Scratch, Ardunio and other programming environments. Scoring was made according to the order of importance of programming languages in order to determine the level of programming courses. Accordingly, Code.org and others (Google CS First, Hour of Code, Blockly.games, CodeMonkey, Codecademy, Codesters etc.) created a scale-type measurement tool with one score, Scracth two points, and Ardunio three points. In this scale, the scores were between the highest score "7 (experiencing all programming environments)" and the lowest "0 (no programming experiences)". Programming experiences average value was calculated as 1.94 (SD = 1.61) which showed that students had low-level programming experiences. The correlation coefficient between the level of programming environments experienced and CT skills were given in Table 8.

All correlation coefficients between CT sub-dimensions were statistically significant and the highest correlation coefficient was found between abstraction and decomposition (r = .78). On the other hand, a statistically significant relationship of the programming experiences was found to be the algorithm, evaluation, and the total score. Accordingly, this result showed that as the students' experience in different learning environments increases, algorithmic thinking, evaluation and total computational thinking skills would increase.

Table 8. Relationship between students' programming experiences and CT skills

		1	2	3	4	5	6	7
1	Abstraction	1						
2	Decomposition	.78**	1					
3	Algorithmic Thinking	.75**	.74**	1				
4	Evaluation	.72**	.65**	.63**	1			
5	Generalization	.68**	.65**	.69**	.77**	1		
6	Total	.78**	.76**	.71**	.70**	.65**	1	
7	Programming experiences	.11	.06	.15*	.18**	.11	.14*	1

Note. * p <.05. ** p <. 01.

4. Discussion

Little has been known about how different programming environments would promote students' domain-general CT skills and whether there is a difference between the CT skills of the middle school students who have programming experience and those who do not. In this study, due to the lack of domain general CT skill scale in Turkey, first, it was aimed to adapt the CTSCLE developed by Tsai et al. (2021) to Turkish and then, to examine the domain-general CT skills of the students through this scale.

Results showed that there was no significant difference in CT skills of the students according to gender. This finding is parallel to many studies (Alsancak Sarıkaya, 2019; Atmatzidou & Demetriadis, 2016; İbili et al., 2020; Korucu et al., 2017; Mindetbay et al., 2019; Oluk & Korkmaz, 2016; Tsai et al., 2021; Wu & Su, 2021; Yağcı, 2018). In the light of these findings, it can be said that gender does not play an active role in CT. However, there are also studies that contrast with this finding. A study conducted with 5th and 6th grade level 2015-2016-2017-2018-year Bebras (International Challenge on Informatics and Computational Thinking) activity to participate in the 97.494 students in Turkey reported that girls were more successful than boys in CT skills (Gulbahar et al., 2020). The large sample size and the environments used in teaching programming may have caused this difference. There may also be differences in CT skills of students in different programming learning environments (Ardito et al., 2020; Pala & Mıhçı Türker, 2019; Wu & Su, 2021; Yildiz Durak, 2020).
One important finding of the study is that there was a significant difference in the CT skills of the students according to whether they have gain programming skills or not. In this study, it was also found that the algorithmic thinking, evaluation, and generalization skills of students who gained programming skills in different environments were statistically higher than students who did not learn programming skills whereas no significant difference in abstraction and decomposition skills was found. This finding showed that students could improve their abstraction and decomposition skills even without learning any programming. In the study conducted by Günbatar (2020), it was found that middle school students who took the Information Technologies & Software (IT&S) course and learned programming had significantly higher CT skills than those who did not take the course. The development of students' CT skills after they are exposed to programming courses is actually expected. Moreover, learning any programming language is seen as one of the most effective ways to improve students' CT skills (Lye & Koh, 2014). In the previous studies, it was found that there was a high-level significant relationship between Scratch programming skills of middle school students and their CT skills (Oluk & Korkmaz, 2016). Also, students 'algorithmic thinking skills were improved through learning any programming language (Grover et al., 2016). Furthermore, programming teaching with Scratch improved middle school students' CT and algorithmic thinking skills (Erümit et al., 2020; Oluk et al., 2018) and problem solving, algorithmic thinking and creative thinking skills (Alsancak Sarıkaya, 2019; Yünkül et al., 2017).
Another result of the research was that there was no significant difference in the CT skills of the students who had a computer or not and internet access or not. In previous studies, CT skills did not differ in terms of having a computer (İbili et al., 2020), duration of computer use (Oluk & Korkmaz, 2016), weekly internet usage duration, mobile technology usage qualification and mobile technology (Korucu et al., 2017). In a study, Alsancak Sirakaya (2020) found that students' CT skills differed according to their internet experience, mobile device experience, mobile internet experience and daily mobile internet usage time but did not differ according to their computer experience, the number of times they checked their mobile devices in a day and the purpose of using mobile technology.

A significant positive relationship was found between the programming experience and CT skills. This finding implies that as students' programming experience in different environments such as Code.org, Scartch, Arduino increase, their CT skills also increase significantly. In the study conducted by Oluk and Korkmaz (2016), it was found that there is a highly significant relationship between the Scratch programming skills of middle school students and their CT skills. Tsai et al. (2021) found that students with 1 year or more programming experience had higher CT skills than those who did not. These findings show that students' programming experience is an important factor in terms of CT skills. However, there are studies that contradict these findings. In the study conducted by Alsancak Sarıkaya (2019), it was concluded that there was no significant difference in CT skills between students who saw themselves as novice and intermediate level programming skills. In another study, Ibili et al. (2020) reported that CT skills did not differ based on students' programming experience. That may be due to the difference in educational approaches, techniques, and activities in the teaching programming environment (Erümit et al., 2020).

As a result of the study, it was found that some students were about to graduate from middle schools without programming experiences; however, their abstraction and decomposition skills developed in some way, but algorithmic thinking, evaluation and generalization skills were not developed sufficiently. Results of this study suggest to complete the lack of computer laboratories in these schools and to employ qualified information technology teachers in order to provide CT skills, especially for students at schools located in rural areas where they suffer the lack of equipped computer laboratories. By doing so, students in rural areas will be provided with equal opportunities in order for them to be effectively prepared for 21st century professions by improving their CT skills.

This study has some limitations. The first limitation of the study is that the number of middle school students is small; therefore, the generalizability should be approached with caution. Nevertheless, the number of the sample in the study was above the recommended participants for statistical tests and research findings have potential to portray middle school students' CT skills in Turkey. The second limitation of the study is that different variables other than gender, programming experience, computer possession, and internet access could be used to explain the CT skills. Lastly, the future studies could examine students' CT skills in more depth with qualitative studies.

5. Conclusion

In this study, the Computational Thinking Scale for Computer Literacy Education, developed by Tsai et al. (2020), was adapted to Turkish, and its validity and reliability were examined. The CFA and Cronbach alpha results showed that the adapted scale was valid and reliable. Although the scale adapted in the study was a domain-general, as suggested by Tsai et al. (2021), the scale can be used for measuring students' domain-specific CT skills in other disciplines including physics and STEM by adding appropriate words in expression. Addition to this, it was emphasized that the scale can be used to measure CT skills of all participants at the secondary school level and above (Tsai et al., 2021). Moreover, the scale should be used by researchers working in different fields to measure CT skills, regardless of domain-general or domain-specific.

It was also concluded that the domain-general CT scale was suitable for Turkish culture, and there was no significant difference in students' CT skills according to gender, computer possession and Internet access. A statistically significant difference in algorithmic thinking, assessment, generalization, and general CT skills was found between students who learned programming and those who did not, in favor of students learning programming. In the study, it was concluded that there was a positive and significant relationship between the programming experience of students who learn programming and their CT skills. As students' programming experience increased, their CT skills also increased.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 833–835. https://doi.org/10.1093/comjnl/bxs074

Alsop, Y. (2019). Assessing computational thinking process using a multiple evaluation approach. *International Journal of Child-Computer Interaction, 19*, 30–55. https://doi.org/10.1016/j.ijcci.2018.10.004

Alsancak Sarıkaya, D. (2019). Programlama öğretiminin bilgi işlemsel düşünme becerisine etkisi [The effect of programming teaching on computational thinking]. *The Journal of Turkish Social Research, 23*(2), 575–590.

Alsancak Sirakaya, D. (2020). Investigation of computational thinking in the context of ICT and mobile technologies. *International Journal of Computer Science Education in Schools, 3*(4), 50–59. https://doi.org/10.21585/ijcses.v3i4.73

Ardito, G., Czerkawski, B., & Scollins, L. (2020). Learning computational thinking together: Effects of gender differences in collaborative middle school robotics program. *TechTrends, 64*(3), 373–387. https://doi.org/10.1007/s11528-019-00461-8

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems, 75*, 661–670. https://doi.org/10.1016/j.robot.2015.10.008

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting, Vancouver, BC, Canada,* 1–25. https://doi.org/10.1.1.296.6602

Çapık, C., Gözüm, S., & Aksayan, S. (2018). Kültürlerarası ölçek uyarlama aşamaları, dil ve kültür uyarlaması: Güncellenmiş rehber [Intercultural scale adaptation stages, language and culture adaptation: Updated guideline]. *Florence Nightingale Journal of Nursing 26*(3), 199–210. https://doi.org/10.26650/fnjn397481

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*(1), 240–249. https://doi.org/10.1016/j.compedu.2011.08.006

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM, 60*(6), 33–39. https://doi.org/10.1145/2998438

Erümit, A. K., Şahin, G., & Karal, H. (2020). YAP programlama öğretim modelinin öğrencilerin bilgi-işlemsel düşünme becerilerine etkisi [The effects of YAP programming teaching model on students' computational thinking skills]. *Kastamonu Education Journal, 28*(3), 1529–1540. https://doi.org/10.24106/kefdergi.3915

Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (2012). *How to design and evaluate research in education* (8th ed.). Boston, MA: McGraw Hill.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2016). Factors influencing computer science learning in middle school. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16, March*, 552–557. https://doi.org/10.1145/2839509.2844564

Guzdial, M. (2008). Education: Paving the way for computational thinking. *Communications of ACM, 51*, 25–27. https://doi.org/10.1145/1378704.1378713

Gülbahar, Y., Kalelioğlu, F., Doğan, D., & Karataş, E. (2020). Bilge Kunduz: Enformatik ve bilgi-işlemsel düşünmeyi kavram temelli öğrenme için toplumsal bir yaklaşım [Bebras: An approach for concept based learning of informatics and computational thinking]. *Ankara University Journal of Faculty of Educational Sciences, 53*(1), 241–272. https://doi.org/10.30964/auebfd.560771

Gülbahar, Y., Kert, S. B., & Kalelioglu, F. (2019). Bilgi işlemsel düşünme becerisine yönelik öz yeterlik algısı ölçeği: Geçerlik ve güvenirlik çalışması [The self-efficacy perception scale for computational thinking skill: validity and reliability study]. *Turkish Journal of Computer and Mathematics Education, 10*(1), 1–29. https://doi.org/10.16949/turkbilmat.385097

Günbatar, M. S. (2020). Computational thinking skills, programming self-efficacies and programming attitudes of the students. *International Journal of Computer Science Education in Schools, 4*(2), 24–35. https://doi.org/10.21585/ijcses.v4i2.96

Hair, J. F., Blacks, W. C., Babin, B. J., & Anderson, R. E. (2019). *Multivariate data analysis* (8th ed.). Andover, Hampshire, UK: Cengage Learning.

Hu, L. T., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. Structural Equation Modeling, 6(1), 1–55. https://doi.org/10.1080/10705519909540118

İbili, E., Günbatar, M. S., & Sarıkaya, M. (2020). Bilgi-işlemsel düşünme becerilerinin incelenmesi: Meslek liseleri örneklemi [An examination of the computational thinking skills: Sample of vocational high schools]. *Kastamonu Education Journal, 28*(2), 1067–1078. https://doi.org/10.24106/kefdergi.683577

ISTE, & CSTA. (2011). *Operational definition of computational thinking for K-12 education*. http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf

Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing, 4*(3), 583–596.

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior, 72*, 558–569. https://doi.org/10.1016/j.chb.2017.01.005

Korkmaz, Ö., Çakır, R., & Özden, M. Y. (2015). Bilgisayarca düşünme beceri düzeyleri ölçeğinin (BDBD) ortaokul düzeyine uyarlanması [Computational thinking levels scale (CTLS) adaptation for secondary school level]. *Gazi Journal of Educational Science, 1*(2), 143–162.

Korucu, A. T., Gencturk, A. T., & Gundogdu, M. M. (2017). Examination of the computational thinking skills of students. *Journal of Learning and Teaching in Digital Age, 2*(1), 11–19.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Mindetbay, Y., Bokhove, C., & Woollard, J. (2019). What is the relationship between students' computational thinking performance and school achievement? *International Journal of Computer Science Education in Schools*. https://doi.org/10.21585/ijcses.v0i0.45

Oluk, A., & Korkmaz, Ö. (2016). Comparing students' Scratch skills with their computational thinking skills in terms of different variables. *International Journal of Modern Education and Computer Science, 8*(11), 1–7. https://doi.org/10.5815/ijmecs.2016.11.01

Oluk, A., Korkmaz, Ö., & Oluk, H. A. (2018). Effect of Scratch on 5th graders' algorithm development and computational thinking skills. *Turkish Journal of Computer and Mathematics Education (TURCOMAT), 9*(1), 54–71. https://doi.org/10.16949/turkbilmat.399588

Pala, F. K., & Mıhçı Türker, P. (2019). The effects of different programming trainings on the computational thinking skills. *Interactive Learning Environments*, *0*(0), 1–11. https://doi.org/10.1080/10494820.2019.1635495

Román-González, M., Moreno-León, J., & Robles, G. (2019). Combining assessment tools for a comprehensive evaluation of computational thinking interventions. In S.-C. Kong & H. Abelson (Eds.), *Computational Thinking Education* (pp. 79–98). Singapore: Springer. https://doi.org/10.1007/978-981-13-6528-7_6

Selby, C., & Woollard, J. (2013). *Computational thinking: The developing definition*. https://eprints.soton.ac.uk/356481/

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Tang, K. Y., Chou, T. L., & Tsai, C. C. (2020). A Content analysis of computational thinking research: An international publication trends and research typology. *Asia-Pacific Education Researcher, 29*(1), 9–19. https://doi.org/10.1007/s40299-019-00442-8

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers and Education, 148*, 1-22. https://doi.org/10.1016/j.compedu.2019.103798

Tikva, C., & Tambouris, E. (2021). Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature Review. *Computers and Education*, *162*, 1-23. https://doi.org/10.1016/j.compedu.2020.104083

Tsai, M.-J., Liang, J.-C., & Hsu, C.-Y. (2021). The computational thinking scale for computer literacy education. *Journal of Educational Computing Research, 59*(4), 579–602. https://doi.org/10.1177/0735633120972356

Vallance, M., & Towndrow, P. A. (2016). Pedagogic transformation, student-directed design and computational thinking. *Pedagogies: An International Journal*, *11*(3), 218–234. https://doi.org/10.1080/1554480X.2016.1182437

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

Wing, J. M. (2011). *Research notebook: Computational thinking - what and why? The Link*. http://www.cs.cmu.edu/sites/default/files/11-399_The_Link_Newsletter-3.pdf

Wu, S. Y., & Su, Y. S. (2021). Visual programming environments and computational thinking performance of fifth- and sixth-grade students. *Journal of Educational Computing Research, 2*. https://doi.org/10.1177/0735633120988807

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. *TechTrends, 60*(6), 565–568. https://doi.org/10.1007/s11528-016-0087-7

Yağcı, M. (2018). A study on computational thinking and high school students' computational thinking skill levels. *International Online Journal of Educational Sciences, 10*(2), 81–96. https://doi.org/10.15345/iojes.2018.02.006

Yildiz Durak, H. (2020). The effects of using different tools in programming teaching of secondary school students on engagement, computational thinking and reflective thinking skills for problem solving. *Technology, Knowledge and Learning, 25*(1), 179–195. https://doi.org/10.1007/s10758-018-9391-y

Yünkül, E., Durak, G., Çankaya, S., & Abidin, Z. (2017). Scratch yazılımının öğrencilerin bilgisayarca düşünme becerilerine etkisi [The effects of Scratch software on students' computational thinking skills]. *Necatibey Faculty of Education Electronic Journal of Science and Mathematics Education, 11*(2), 502–517.

Zhang, LC, & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers and Education*, *141*. https://doi.org/10.1016/j.compedu.2019.103607

Use Hopscotch to Develop Positive Attitudes Toward Programming for Elementary School Students

Jiahui Wang

Kent State University

DOI: 10.21585/ijcses.v5i1.122

Abstract

With the advancement in technology and the emphasis on computer science education, there has been a strong push for more widespread programming instruction at K-12 and higher education levels. Existing research has mostly focused on students at the secondary and post-secondary levels. Not much research has involved students at the elementary school age, which has been considered a critical age to cultivate an interest in programming. The current study aimed to investigate the effects of a block-based programming interface (e.g., Hopscotch) on elementary school students' attitudes toward programming. In this study, eighteen elementary school students in 4th -5th grades participated in a programming curriculum for about seven weeks in the US. A survey on attitudes toward programming was distributed before and after the curriculum, to explore the change in attitudes toward programming. Students' views about the block-based programming interface (e.g., Hopscotch) were also examined after the curricular activities. Students' activities in lessons and artifacts from the culminating project were observed. The findings indicated that elementary school students had positive views about programming in the block-based programming interface. Also, the block-based programming activities contributed to more positive attitudes toward programming. Implications and limitations of the study were discussed.

Keywords: Computer science, block-based programming, elementary school students, attitudes toward programming, Hopscotch

1. Introduction

1.1 Computer Science Education

United States has an increasing demand for STEM workers than at any time in history, and teaching computer science to students is essential for recruiting STEM workers (Guzdial & Morrison, 2016). However, existing findings indicated that K-12 educators in the United States did not attach enough importance to computer science education (Google Gallup, 2015). Many colleges and universities have also reported declines in enrollment in computer science related courses and majors (Bowman, 2018). As a result, United States has been confronted with a pronounced lack of talents in computer science related professions (Seehorn et al., 2011). The shortage is especially pronounced among females (Grover & Pea, 2013) and non-Asian minorities (Banning & Folkestad, 2012).

1.2 Benefits of Programming Learning

Programming learning has been found to be beneficial to students. According to Wing (2006), the most important skill students can acquire from learning computer science is computational thinking, which refers to the use of abstract thinking to seek a solution to a problem. More specifically, computational thinking is defined as "the thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (Wing, 2011, p. 20). Computational thinking has been theorized as a multi-dimensional construct, which encompasses computational concepts, computational practices, and computational perspectives (Brennan & Resnick, 2012). Wing also emphasized the important role computational thinking plays in learning of all subject domains, as she said, "to reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" (Wing, 2006, p. 33).

In addition to facilitating the development of computational thinking, previous work has also identified some positive effects brought by computational way of thinking, which includes promoting divergent thinking and metacognitive skills (Clements & Gullo, 1984), critical thinking (Clements, & Gullo, 1984; Liao & Bright, 1991), as well as enhancing spatial relations and problem-solving abilities (Fessaki, Gouli, & Mavroudi, 2013; Miller, Kelly, & Kelly, 1988).

1.3 Block-Based Programming Environments

Despite the increasing importance of programming and the benefits of programming learning, computer science education has faced certain challenges, indicated by previous studies (Denner, Werner, & Ortiz, 2012; Robins, Rountree, & Rountree, 2003). Most importantly, programming is a complex mental process and a challenging task (Law, Lee, & Yu, 2010). The conventional text-based environments could burden the learners with syntax and cause frustrations. The learners could lose interest in programming very quickly in these text-based programming environments. To overcome these barriers, various efforts have been made to develop tools and activities to popularize computer science education and engage students in programming activities within K-12 and higher education contexts (Lye & Koh, 2014). With the advancements in learning technologies and their widespread applications in students' learning, several block-based programming interfaces have been designed and introduced to teachers and students to support programming learning at various education levels (Amanullah & Bell, 2020; Denner, Werner, & Ortiz, 2012; Leidl et al., 2017). These block-based interfaces allow the design of programming logics using children-friendly drag-and-drop of blocks. Compared to text-based programming, block-based programming interfaces are useful in reducing the abstractness of syntax and the complexity of programming. They could alleviate frustrations associated with writing abstract syntax and debugging syntax. These programming environments could help novice programmers focus better on the core aspects of programming, that is, the logical thinking process. These environments would allow learners to see how the commands work immediately and detect errors much more easily (Lye & Koh, 2014). Often, the block-based programming interfaces adopt cartoon characters that appeal to the young learners and attract them to construct animations, games, or digital stories relevant to their interests and passions.

Regarding the effects of block-based programming compared to text-based programming, a recent meta-analysis on this topic (Xu et al., 2019) compared block-based programming environments with text-based environments with respect to their impacts on students' cognitive and affective outcomes. With most of the studies examined focusing on secondary education and higher education, the meta-analysis revealed a small effect size favoring block-based programming interfaces on cognitive outcomes; while for affective outcomes, the analysis only identified a trivial effect size. Moreover, the meta-analytic study revealed a significant effect size for elementary school, but only one study focused on the elementary school-age population.

The block-based programming interfaces include but are not limited to Hopscotch, Alice, Scratch, and Minecraft. It has been suggested that students became less burdened by the syntax of programming and became more motivated about programming as exposed to a block-based programming environment named Scratch (Kaucic & Asic, 2011). Block-based programming tools have become an essential component of computer science curricula for high school and university classrooms. Empirical evidence has suggested these block-based programming environments have been proven successful at the secondary level (Burke & Kafai, 2012; Campe et al., 2020; Grover, Pea, & Cooper, 2015; Gunbatar & Karalar, 2018; Meerbaum-Salant, Armoni, & Ben-Ari, 2013; Price & Barnes, 2015) and in higher education (Cetin, 2016; Korkmaz, 2016; Yukselturk & Altiok, 2017). For example, in the study conducted by Burke et al. (2012), researchers explored the effect of Scratch on enhancing middle school students' programming skills. The findings suggested the middle school students were able to master basic programming concepts (e.g., loops, events) to create digital stories. Similarly, another study (Meerbaum-Salant, Armoni, & Ben-Ari, 2013) introduced middle school students to Scratch and the students were found to be able to acquire some important programming concepts, despite having troubles learning some concepts such as repeated execution and variables. Furthermore, Grover and colleagues (2015) introduced middle school students to block-based programming in an introductory programming course. The results revealed that students achieved significant learning gains in algorithmic thinking skills, and they were also able to transfer the knowledge from the block-based programming activities to a text-based programming context. Another study that involved pre-service teachers in a university (Yukselturk & Altiok, 2017) reported that a block-based programming environment (e.g., Scratch) was effective in alleviating negative attitudes such as low self-efficacy among pre-service teachers who came from non-CS backgrounds. Taken together, these studies suggested that exposing learners to block-based programming would be effective in helping young learners acquire basic programing concepts that could be transferrable to text-based programing contexts. It is also reasonable to speculate that the block-based programming tools would be effective in prompting more positive attitudes toward programming among the young learners, due to the fact that the drag-and-drop interface would engage the learners in the logical thinking process while preventing the frustration associated with writing syntax.

1.4 Programming Learning at an Early Age

Developing positive attitudes and stimulating interests in programming at an early age is essential for broadening participation in computer science career (Hainey et al., 2019). Thus, more work focusing on introducing programming at earlier education levels are needed to identify effective approaches and curriculum to facilitate programming education at a young age.

Literature review revealed that the majority of existing research efforts have primarily focused on the contexts of secondary education and higher education, and not so much research has been undertaken on programming education at early age. Among these efforts for young learners, Bers and her colleagues' work has focused on integrating programming in the early childhood classroom (e.g., Bers, 2019, 2020). For example, programming robots have been adopted to support early learning of programming (Kazakoff et al., 2013; Strawhacker & Bers, 2015; Sullivan & Bers, 2013). In one study, Kazakoff and colleagues (2013) found that early learners' sequencing skills could be improved after taking part in a workshop involving the use of programming robots. Moreover, research has been conducted to examine the use of a block-based programming environment - ScratchJr among learners in early childhood (Flannery et al., 2013; Strawhacker & Bers, 2019; Sullivan & Bers, 2019). Specifically, it was suggested that ScratchJr was an effective tool in that the young students (Grade K-2) acquired the foundational programming concepts (Strawhacker & Bers, 2019).

In recent years, growing attention has been given to integrating programming instruction in elementary school (Allsop, 2019; Bell, Duncan, et al., 2016; Bell, Witten, et al., 2016; Duncan et al., 2017). For example, Hainey and colleagues (2019) utilized a novel approach called games-based construction learning (GBCL) to teach programming concepts in upper elementary school. Their findings indicated that the games-based construction learning (GBCL) approach was an effective approach to teach programming concepts in upper elementary school. The elementary school students were able to learn programming concepts effectively. Additionally, other work has focused on methods to evaluate the Computational Thinking (CT) process in an elementary school classroom (Allsop, 2019) and measure students' understanding of computer science concepts (Denner, Werner, & Ortiz, 2012).

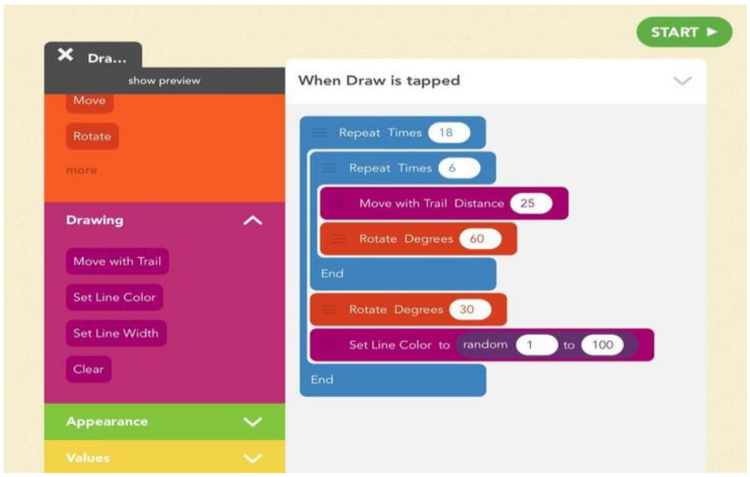
With existing evidence showing the effectiveness of block-based programming among students in secondary schools and universities, as well as limited evidence in early childhood education, it is reasonable to expect these block-based interfaces could also benefit elementary school students by involving less complicated programming tasks. It is safe to postulate that block-based programming environments would be useful in stimulating elementary school students' interest, familiarizing them with introductory programming concepts, and eventually building a foundation for text-based programming. In fact, a recent study (Chen et al., 2019) provided some evidence for this assumption. The researchers examined the relationship between undergraduate students' final grades in introductory computer science courses and their very first programming languages before adolescence. The findings revealed that those who received higher final grades in CS courses had their initial exposure to programming in graphical language rather than textual, in or before early adolescent years. The findings suggested graphical language should be adopted for young learners' initial exposure to programming, if programming is to be taught before early adolescence.

A recent meta-analysis, however, indicated that the effects of block-based programming among elementary school students were not adequately studied (Xu et al., 2019). Only a few empirical studies have focused on this population with specific attention given to the cognitive outcomes of block-based programming environments, but findings on the cognitive effects were mixed. For example, adopting a pretest and posttest design, Lai & Yang (2011) examined if block-based programming (e.g., Scratch) would have a positive effect on elementary school students' problem-solving skills and the findings indicated block-based programming activities improved students' problem-solving abilities significantly. On the contrary, Kalelioglu & Gülbahar (2014) failed to replicate the positive effects of block-based programming on problem-solving abilities. Besides examining the influence on problem-solving skills, Sáez-López, Román-González, & Vázquez-Cano (2016) studied the impact of block-based programming on computational thinking and computational practices, and their findings suggested significant improvements in these two areas.

Additionally, the study conducted by Baser (2013) suggested a positive relationship between students' attitudes toward programming and their achievements in programming. It also needs to be pointed out that one of the most significant obstacles in computer science education is the negative attitudes towards programming among students (Bishop-Clark, Courte, & Howard, 2006; Yukselturk & Altiok, 2017). If students could develop early interests in and positive attitudes toward programming, it is more likely they would pursue a major in computer science related disciplines, leading to a career in computer science later on. Thus, it is important to cultivate positive attitudes toward programming, especially at an early age. However, the effects of block-based programming on elementary school students' affective outcomes are not fully understood, either (Xu et al., 2019). Empirically, Duncan & Bell (2015) implemented a programming course for elementary school students aged 11-12. The researchers surveyed the participants to see if the programming course changed their attitudes to computing as career. It was pointed out the study did not measure learners' attitudes prior to the programming course. As all the tests were administered after the course, it was not possible to identify the improvement in attitudes toward programming. More empirical work following a pretest and posttest design is needed to examine the effects of block-based programming on learners' attitudes, especially at the elementary level. Thus, the current study aimed to close the gap in the existing literature by providing data to illustrate the effects of block-based programming on elementary school students' attitudes toward programming.

1.5 Current Study

For teaching programming in elementary classrooms, many tools and resources are available (Duncan, Bell, & Tanimoto, 2014). For example, ScratchJr was developed as an introductory programming environment for young learners aged 5-7, where learners could create a program/story by sequencing different types of programming blocks (e.g., triggering blocks, motion blocks, looks blocks, sound blocks, control blocks, and end blocks, Leidl et al., 2017). In contrast, Hopscotch was designed to target ages 10-15 and was considered as an age-appropriate tool for teaching programming concepts such as loops, randomization, and conditionals to these elementary school students who participated in the current study. Hopscotch was selected also due to its ease of use and compatibility with iPad. Hopscotch, as a typical block-oriented interface, adopts a drag-and-drop graphical programming environment (see a screenshot in Figure 1). Learners don't need to write complicated syntax, and instead, they could construct a programming work (e.g., animation, game, digital stories, etc.) by building a number of blocks in the interface. Hopscotch could introduce novice programmers to fundamental programming concepts such as loops, randomization, and conditional, just to name a few. The current study explored the effects of block-based programming (e.g., Hopscotch) for developing elementary school students' positive attitudes toward programming by adopting a pretest and posttest design.



```
When Draw is tapped
Repeat Times 18
  Repeat Times 6
    Move with Trail Distance 25
    Rotate Degrees 60
  End
  Rotate Degrees 30
  Set Line Color to random 1 to 100
End
```

Figure 1. A screenshot of the Hopscotch interface.

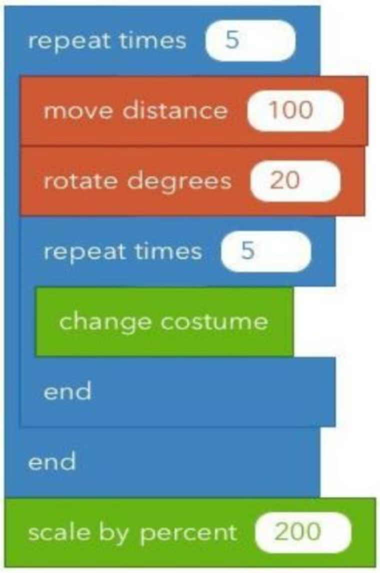
The study was designed to address the following two research questions:

> **RQ1:** How does block-based programming experience in Hopscotch influence elementary school students' attitudes toward programming?
>
> **RQ2:** How do the elementary school students view block-based programming in Hopscotch?

2. Method

2.1 Context & Participants

The study was situated and carried out in a weekend school in the United States. Eighteen elementary school students in grades 4-5 (13 males and 5 females) who had no prior experience in programming participated in the curricular activities. Informed consents were obtained from parents and students. The curriculum comprised seven 1-hour lessons and presented an introduction to elementary programming for this group of students. The curriculum began with an introduction to the field of computer science, which included topics such as careers in computer science, solving problems with computer science, as well as applications of computer science in daily life. The Hopscotch interface was also briefly introduced in the first lesson. Lesson 2 through 6 introduced the students to basic programming concepts such as variables, loops, randomization, and conditionals. Students were first instructed how these concepts could be used in daily life situations to solve real-world problems. The programming instruction was then provided to demonstrate examples and model the process of implementing the commands in the Hopscotch interface. The iPad used by the instructor was projected to the whole classroom to demonstrate the process in the Hopscotch interface. Figure 2 presents an example of how a loop could be accomplished in the Hopscotch interface. Then the students were instructed to exercise the commands and implemented a similar (but not exactly the same) program that incorporated the concepts in the Hopscotch interface on their individual iPads. During the process, the instructor circled around the room and provided scaffolding and feedback for the students as they worked on their own individual programs in Hopscotch. The instructor provided hints and showed how she might approach executing certain commands, instead of giving a direct answer to the problem. The final lesson challenged the elementary school students to self-design a program that incorporated the programming concepts (e.g., variables, loops, randomization, and conditionals) they had learned throughout the first six lessons of the curriculum. For the final project, the students were expected to test and revise the commands to ensure the program could run properly as intended. The final work required the students to apply the previously learned programming concepts in a new context. Each student shared their finished work at the end.



```
repeat times 5
  move distance 100
  rotate degrees 20
  repeat times 5
    change costume
  end
end
scale by percent 200
```

Figure 2. An example of a loop in block-based programming (e.g., Hopscotch).

2.2 Data Collection and Measures

In order to understand how block-based programming (e.g., Hopscotch) influences elementary school students' attitudes toward programming, multiple data sources were collected. First, the study adopted a within-subjects pretest and posttest design. A survey was distributed to the students before the class, which included four Likert-scale questions (see Table 1) that were adapted from a previously validated scale on attitudes toward computer science (Hoegh & Moskal, 2009). The survey was readministered to the participants as a posttest after the learning experience in Hopscotch. While the learners responded to the statements, the researcher circled around the room and provided clarifications as needed. In fact, no student raised any questions about the statements. The reliability for the pretest is Cronbach's Alpha $\alpha = .645$, and the reliability for the posttest is $\alpha = .762$. On the posttest, students also responded to an additional question, "Programming in Hopscotch is a positive experience for me," on a 5-point Likert scale from *strongly disagree* to *strongly agree*. They were also requested to provide more explanations for their selections. Students' activities throughout the lessons and their artifacts created in the Hopscotch interface were observed.

3. Results

3.1 RQ1: How does block-based programming in Hopscotch influence elementary school students' attitudes toward programming?

To decide the appropriate statistical test to examine if there was a significant difference between the pretest and posttest responses to the four statements that gauged students' attitudes toward programming, data were checked for normality using the Shapiro-Wilk test. The assumption of normal distribution was rejected, which indicated the need to use Wilcoxon signed-rank tests to conduct the analyses. Table 1 presents the descriptive statistics for the level of agreement with each individual statement. The results indicated a significant increase in agreement with the statement after participating in the programming curricular activities. Specifically, for "I will take more programming courses and learn more about programming in future", participants agreed more with the statement after the programming experience, Z = 3.755, *p* = .000. For "I hope that my future career will involve programming", the results showed a statistically significant difference between pretest and posttest, Z = 3.906, *p* = .000. For "Having background knowledge and understanding of computer science is valuable in daily life", there was a significant increase in agreement with the statement after the curricular activities, Z = 2.887, *p* = .004. For "The challenge of solving problems using computer science appeals to me", participants had a significantly higher level of agreement with the statement on the posttest than on the pretest, Z = 3.317, *p* = .001. For all the statements, participants' levels of agreement with the statements significantly increased from the pretest to the posttest. This result may show that Hopscotch-based programming activities led to more positive attitudes toward programming among elementary school students. Overall, these findings suggested that block-based programming (e.g., Hopscotch) was helpful in getting elementary school students interested in programming and motivated to learn more about computer science in future.

Table 1. Descriptive Statistics for the level of agreement with the statements

	Before (*n* = 18)		After (*n* = 18)	
Please circle the number below that indicates how much you agree or disagree with each statement: (1) strongly disagree, (2) disagree, (3) neither agree nor disagree, (4) agree, (5) strongly agree.	*M*	*SD*	*M*	*SD*
S1: I will take more programming courses and learn more about programming in future.	2.94	0.87	4.00	0.49
S2: I hope that my future career will involve programming.	3.11	0.47	4.39	0.61

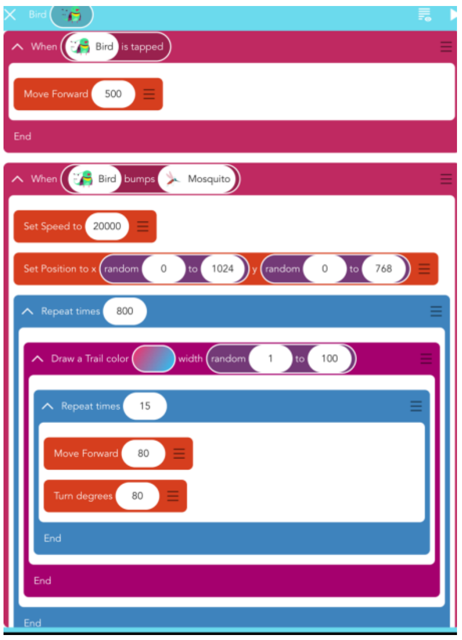
S3: Having background knowledge and understanding of computer science is valuable in daily life.	3.44	0.51	4.00	0.59
S4: The challenge of solving problems using computer science appeals to me.	3.00	0.69	3.61	0.98

3.2 RQ2: How do the elementary school students view block-based programming in Hopscotch?

The results demonstrated that elementary school students' views about block-based programming in Hopscotch were positive after participating in the Hopscotch-based programming curriculum. As students were asked their level of agreement with the statement "Programming in Hopscotch is a positive experience for me", 12 strongly agree with the statement, and 6 agreed with the statement. One participant wrote, "I really enjoyed the app; I would like to learn more about programming." It was also mentioned by two participants, "Hopscotch is easy to use." Another participant expressed her appreciation of the app by noting, "I like the way how the app works; it allows me to see how my codes work right away." Other comments from the participants included the following:

- Programming is fun.
- I like the app.
- Programming is interesting.
- I had lots of fun with programming.
- I like the idea of building animations with my favorite characters.

Overall, the elementary school students felt positive about and engaged in using Hopscotch to practice programming, and the researcher's in-class observations also corroborated this finding. Although the curriculum only involved some basic programming concepts, students learned how to apply abstract thinking to solve a problem (e.g., implement a simple program in Hopscotch). It can be argued that Hopscotch-based programming instruction helped elementary school students comprehend the fundamental programming concepts, which can be gleaned from their final programming products. The final projects students created demonstrated they had mastered the previously learned knowledge and skills throughout the curricular activities (see an example of a student's final project codes in Figure 3). The findings suggested block-based programming activities in Hopscotch were beneficial in assisting the development of programming skills for elementary school students.



```
Bird
When Bird is tapped
    Move Forward 500
End
When Bird bumps Mosquito
    Set Speed to 20000
    Set Position to x random 0 to 1024 y random 0 to 768
    Repeat times 800
        Draw a Trail color width random 1 to 100
            Repeat times 15
                Move Forward 80
                Turn degrees 80
            End
        End
    End
```

Figure 3. An example of students' final project in Hopscotch.

4. Discussion

The study contributed to the effort to expand programming education at the elementary level. The study sought to enhance our understanding of how block-based programming influences elementary school students' attitudes towards programming. The findings revealed block-based programming was effective in cultivating positive attitudes toward programming among elementary school students. It was observed that the students were able to make sense of the programing concepts and were able to apply the concepts to the culminating project. The present results contributed to a growing body of literature seeking to understand the effects of block-based programming at various age levels.

Results further showed participants' views about the block-based programming environment were positive, as evidenced by their responses to the statement "Programming in Hopscotch is a positive experience for me" and the explanations they provided for their selections. Generally, the block-based programming environment was well received by the students, for example, one comment was "I like the app". Participants also thought programming in Hopscotch was a fun experience. For example, three participants respectively commented, "programing is fun", "Programming is interesting", and "I had lots of fun with programming". Another participant expressed similar feeling in the response, "I really enjoyed the app". The fun in the programming experience could possibly be attributed to the fact that the programming interface provided characters that appeal to the young participants, and they can build programs that incorporate these characters. For example, one participant provided support for this speculation and mentioned that "I like the idea of building animations with my favorite characters". Another factor that possibly contributed to the positive views about the Hopscotch programing environment is that the Hopscotch app is easy to use and allows the learners to test the program immediately and fix errors (i.e., debugging and problem solving) as needed. For example, one participant commented, "I like the way how the app works; it allows me to see how my codes work right away". Based on the observation, the students enjoyed the process of building up an animation/program through the app, and a high level of interest and engagement was observed throughout the curricular activities.

This finding is in line with two studies that focused on a different block-based programming environment called Scratch. For example, Sáez-López and colleagues (2016) involved elementary school students in 5th-6th grades in block-based programming activities. Based on students' responses to the questionnaire, students demonstrated positive attitudes toward the block-based programming interface. More recently, Mladenović and colleagues (2017) compared block-based programming (e.g., Scratch) and text-based programming (e.g., Python) for game-based programming among 5th-grade elementary school students. By surveying learners' attitudes toward the programming interfaces after the activities, learners displayed more positive attitudes toward programming to Scratch compared to Python. This observation provided more support for learners' positive attitudes toward the block-based programming activities.

The study yielded several practical implications. The current study demonstrated the possibility of developing positive attitudes toward programming by exposing elementary school students to a programming curriculum involving the use of the Hopscotch block-based programming environment. Teachers and parents of elementary school students should take that into consideration.

Hopscotch-based programming instruction represented the very first exposure to computer programming among the learners who participated in the current study. The results of the study indicated Hopscotch was helpful in developing an early interest in programming among the young learners. The findings provided important implications for computer science educators in the elementary school setting. Next, the study revealed that elementary school students demonstrated interest in using Hopscotch for programming, but teachers' guidance is also important in identifying meaningful curricular activities that are educationally valuable in the Hopscotch interface.

The current study also benefited a broader population of elementary school students and teachers, including students from traditionally underrepresented groups in computer science. The curriculum also provided implications for the design of an effective curriculum to arouse children's early interest in programming and develop programming skills, which will help broaden participation in computer science careers in the long-term.

The findings of the present study should be interpreted in light of several limitations, and future work should seek to address these limitations.

One limitation is that the study adopted a small sample size, and it needs to be acknowledged that the present findings may be limited in generalizability. Future studies could benefit from adopting larger sample sizes to explore the effects of block-based programming activities on attitudes toward programming among young learners.

Second, the study investigated the effects of Hopscotch-based programming instruction on attitudes toward programming. A future study should continue to examine the effects of block-based programming instruction on learners' attitudes toward programming by adopting a more comprehensive scale to measure learners' attitudes.

Another limitation of the study was that the attitude survey was conducted right after the curricular activities. Future research is recommended to adopt a longitudinal design and also examine the long-term effects of programming curriculum on learners' attitudes toward programming in the long run.

Previous studies have shown that boys tend to have more positive attitudes toward programming as compared to girls (Baser, 2013; Rubio et al., 2015). These gender differences in students' attitudes towards programming could possibly influence their interests in pursuing computer science majors and eventually undertake careers involving computer science. Future studies could focus on exploring approaches to bridge the gender-based differences in learners' attitudes toward programming.

It is also worth conducting more work to understand the processes of programming learning. For example, a future study could use screen recording and think-aloud protocols (Ericsson & Simon, 1998) to study the processes of programming among elementary school students. The study could examine how novice programmers build and comprehend the block-based codes, which can be used to then improve the learning processes as well as the instructional approaches that target novice learners.

Finally, future work is also recommended to examine ways to integrate programming instruction into elementary school STEM learning by infusing computational thinking into the project-based learning of STEM contents.

Funding

The author(s) received no financial support for the research or publication of this article.

References

Allsop, Y. (2019). Assessing computational thinking process using a multiple evaluation approach. *International Journal of Child-Computer Interaction*, *19*, 30–55.

Amanullah, K., & Bell, T. (2020). Revisiting code smells in block based languages. *ACM International Conference Proceeding Series*, *274*.

Banning, J., & Folkestad, J. E. (2012). STEM education related dissertation abstracts: A bounded qualitative meta-study. *Journal of Science Education and Technology*, *21*(6), 730–741.

Baser, M. (2013). Attitude, gender and achievement in computer programming. *MiddleEast Journal of Scientific Research*, *14*(2), 248–255.

Bell, T., Duncan, C., & Atlas, J. (2016). Teacher feedback on delivering computational thinking in primary school. *ACM International Conference Proceeding Series*, *13-15-Octo*, 100–101.

Bell, T., Witten, I. H., & Fellows, M. (2016). *CS Unplugged: An enrichment and extension programme for primary-aged students*. University of Canterbury. CS Education Research Group, New Zealand.

Bers, M. U. (2019). Coding as another language: a pedagogical approach for teaching computer science in early childhood. *Journal of Computers in Education*, *6*(4), 499–528.

Bers, M. U. (2020). *Coding as a playground: Programming and computational thinking in the early childhood classroom*. Routledge.

Bishop-Clark, C., Courte, J., & Howard, E. V. (2006). Programming in pairs with Alice to improve confidence, enjoyment, and achievement. *Journal of Educational Computing Research*, *34*(2), 213–228.

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

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, Vol. 1, 25.

Burke, Q., & Kafai, Y. B. (2012). The writers' workshop for youth programmers: digital storytelling with scratch in middle school classrooms. *43rd ACM Technical Symposium on Computer Science Education*, 433–438.

Campe, S., Denner, J., Green, E., & Torres, D. (2020). Pair programming in middle school: variations in interactions and behaviors. *Computer Science Education*, *30*(1), 22–46.

Cetin, I. (2016). Preservice teachers' introduction to computing: exploring utilization of scratch. *Journal of Educational Computing Research*, *54*(7), 997–1021.

Chen, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2019). The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages. *Computer Science Education*, *29*(1), 23–48.

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology*, *76*(6), 1051.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, *58*(1), 240–249.

Duncan, C., Bell, T., & Tanimoto, S. (2014). Should your 8-year-old learn coding? *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 60–69.

Duncan, C., & Bell, T. (2015). A pilot computer science and programming course for primary school students. *ACM International Conference Proceeding Series*, *09-11-Nove*, 39–48.

Duncan, C., Bell, T., & Atlas, J. (2017). What do the teachers think? Introducing computational thinking in the primary school curriculum. *ACM International Conference Proceeding Series*, 65–74.

Ericsson, K. A., & Simon, H. A. (1998). How to study thinking in everyday life: Contrasting think-aloud protocols with descriptions and explanations of thinking. *Mind, Culture, and Activity*, *5*(3), 178–186.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87–97.

Flannery, L. P., Kazakoff, E. R., Bontá, P., Silverman, B., Bers, M. U., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. *ACM International Conference Proceeding Series*, 1–10.

Google Gallup. (2015). *Images of computer science: Perceptions among students, parents and educators in the US.*

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43.

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.

Gunbatar, M. S., & Karalar, H. (2018). Gender differences in middle school students' attitudes and self-efficacy perceptions towards MBlock programming. *European Journal of Educational Research*, *7*(4), 925–933.

Guzdial, M., & Morrison, B. (2016). Growing computer science education into a STEM education discipline. *Communications of the ACM*, *59*(11), 31–33.

Hainey, T., Baxter, G., & Ford, A. (2019). An evaluation of the introduction of games-based construction learning in upper primary education using a developed game codification scheme for scratch. *Journal of Applied Research in Higher Education*, *12*(3), 377–402.

Hoegh, A., & Moskal, B. M. (2009). Examining science and engineering students' attitudes toward computer science. *39th IEEE Frontiers in Education Conference*, 1–6.

Kalelioglu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, *13*(1), 33–50.

Kaucic, B., & Asic, T. (2011). Improving introductory programming with Scratch? *2011 Proceedings of the 34th International Convention MIPRO*, 1095–1100.

Kazakoff, E. R., Sullivan, A., & Bers, M. U. (2013). The Effect of a Classroom-Based Intensive Robotics and Programming Workshop on Sequencing Ability in Early Childhood. *Early Childhood Education Journal*, *41*(4), 245–255.

Korkmaz, Ö. (2016). The effect of scratch-based game activities on students' attitudes, self-efficacy and academic achievement. *International Journal of Modern Education and Computer Science*, *8*(1), 16–23.

Lai, A. F., & Yang, S. M. (2011). The learning effect of visualized programming learning on 6th graders' problem solving and logical reasoning abilities. *2011 International Conference on Electrical and Control Engineering, ICECE 2011 - Proceedings*, 6940–6944.

Law, K. M., Lee, V. C., & Yu, Y. T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, *55*(1), 218–228.

Leidl, K. D., Umaschi-Bers, M., & Mihm, C. (2017). Programming with ScratchJr: A review of the first year of user analytics. *Proceedings of International Conference on Computational Thinking Education*, 116–121.

Liao, Y. K. C., & Bright, G. W. (1991). Effects of computer programming on cognitive outcomes: A meta-analysis. *Journal of Educational Computing Research*, *7*(3), 251–268.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239–264.

Miller, R. B., Kelly, G. N., & Kelly, J. T. (1988). Effects of Logo computer programming experience on problem solving and spatial relations ability. *Contemporary Educational Psychology*, *13*(4), 348–357.

Mladenović, M., Krpan, D., & Mladenovi, S. (2017). Learning programming from scratch. *Turkish Online Journal of Educational Technology, November Special Issue*, 419–427.

Price, T. W., & Barnes, T. (2015). Comparing textual and block interfaces in a novice programming environment. *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, 91–99.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172.

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & Angel, P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education*, *82*, 409–420.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129–141.

Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., ... & Verno, A. (2011). *CSTA K-12 Computer Science Standards: Revised 2011*.

Strawhacker, A., & Bers, M. U. (2015). "I want my robot to look for food": Comparing Kindergartner's programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education*, *25*(3), 293–319.

Strawhacker, A., & Bers, M. U. (2019). What they learn when they learn coding: investigating cognitive domains and computer programming knowledge in young children. In *Educational Technology Research and Development* (Vol. 67, Issue 3). Springer US.

Sullivan, A., & Bers, M. (2019). Computer science education in early childhood: the case of ScratchJr. *Journal of Information Technology Education: Innovations in Practice*, *18*(1), 113–138.

Sullivan, A., & Bers, M. U. (2013). Gender differences in kindergarteners' robotics and programming achievement. *International Journal of Technology and Design Education*, *23*(3), 691–702.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Wing, J. (2011). Research notebook: Computational thinking-What and why. *The Link Magazine, 6.*

Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study. *Computer Science Education*, *29*(2–3), 177–204.

Yukselturk, E., & Altiok, S. (2017). An investigation of the effects of programming with Scratch on the preservice IT teachers' self-efficacy perceptions and attitudes towards computer programming. *British Journal of Educational Technology*, *48*(3), 789–801.