International Journal of Computer Science Education in Schools, April 2022, Vol. 5, No. 3
ISSN 2513-8359

Difficult Concepts and Practices of Computational Thinking Using Block-Based Programming

Hyunchang MOON[1]
Jongphil CHEON[2]
Kyungbin KWON[3]

[1] Baylor University, Waco, USA
[2] Texas Tech University, Lubbock, USA
[3] Indiana University, Bloomington, USA

DOI: 10.21585/ijcses.v5i3.129

Abstract

To help novice learners overcome the obstacles of learning computational thinking (CT) through programming, it is vital to identify difficult CT components. This study aimed to determine the computational concepts and practices that learners may have difficulties acquiring and discuss how programming instructions should be designed to facilitate learning CT in online learning environments. Participants included 92 undergraduate students enrolled in an online course. Data were collected from a CT knowledge test and coding journals. Results revealed that four computational concepts (i.e., parallelism, conditionals, data, and operators) and two computational practices (i.e., testing and debugging and abstracting and modularizing) were identified as CT components that were difficult to learn. The findings of this study imply that CT instructions should offer additional instructional supports to enhance the mastery of difficult computational concepts and practices. Further research is necessary to investigate instructional approaches to successful CT learning.

Keywords: computational thinking, block-based programming, Scratch, CT difficulties, CT challenges

1. Introduction

Digital transformation is everywhere. Although innovation in digital technology advances our well-being, the fast rate of world change generates unprecedented social, economic, and environmental challenges. A United Nations report (2019) examining how digital technology would transform our lives and communities emphasized that many people become more vulnerable to uncertain adversity and risks when they do not have the fundamental skills required for finding solutions to real-life problems in the digital age. In this increasingly evolving world, computational thinking (CT) has emerged as a problem-solving skill that new generations of students must acquire to prepare them for tomorrow's challenges and expand their potential. As a response to these issues, educators, researchers, and policymakers are rapidly recognizing that CT is a new core skill needed by all people, not just computer programmers (Wing, 2011). Emphasis is being increasingly placed on developing effective curricula for computer science (CS) and CT education. Also, many efforts have been made in various educational settings to integrate CT components into existing classroom activities.

As part of these ongoing efforts, in 2016 in the United States, the Computer Science for All initiative laid the foundation for providing students in pre-K through 12th grade with opportunities to participate in CS education (National Science Foundation, 2016). Later on, the Common Core State Standards and Next Generation Science Standards were reformed to encompass CT as an interdisciplinary approach. With these recent educational reforms, which incorporated CS/CT into both K-12 and higher education curricula, educators need to adapt their existing pedagogical strategies to properly teach CS/CT to learners. They also need to learn appropriate pedagogies for delivering a new subject, particularly in those aspects of CS/CT competencies. Although recent literature pertaining to CS education in school emphasizes many ways to make CS/CT education more accessible to K-20 students, educators, researchers, and administrators still must manage the ambiguity of CT definitions and methods of instruction and assessment. Particularly, attempts have been made to propose instructional tools to facilitate CT learning, but these studies did not present the most difficult CT components for learners to engage in block-based programming learning. This may be due to the lack of empirical research findings to identify difficult CT concepts and practices in block-based programming environments. Thus, it is crucial to identify difficult-to-learn CT components via learning block-based programming. To situate our study, we first outline CT in general, highlight CT assessments, and then consider what it means in block-based programming and the challenges in CT instruction.

2. Literature Review

2.1 Definitions of Computational Thinking

Alongside the growing recognition of CT as essential for students' future success, several researchers have attempted to define CT and identify its components (e.g., Atmatzidou & Demetriadis, 2016; Barr, Harrison, & Conery, 2011; Berland & Wilensky, 2015; Google, 2016; Israel et al., 2015; Parpert, 1980; Pearson et al., 2015). The term CT was first coined by Seymour Papert (1980), who developed LOGO programming, and was later popularized in the CS community by Jeannette Wing (2006). She described CT as "the thought processes 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. 1). The National Research Council (2010) expanded the nature and scope of CT with diverse applications for the definition. Barr and Stephenson (2011) provided an operational definition of CT for K-12 education, which they described as a problem-solving process and a series of dispositions and attitudes. Aho (2012) refined the term, saying that the solution should be represented as computational steps and algorithms. Román-González (2015) argued that the basic CT concepts—computing and programming—were central to formulating and solving problems. Grover and Pea (2018) redefined CT as a "widely applicable thinking competency" (p. 22) of which problem formulation processes should be considered key in solving problems. Denning and Tedre (2021) advanced CT's definition with a historically grounded view of professional disciplines and highlighted the aspects of "designing computations that get computers to do jobs for us, and for explaining and interpreting the world in terms of information processes" (p. 365). As CT encompasses broad domains across disciplines, there is no standard definition of this term; hence, various components of CT have been differently proposed in line with study contexts, which has influenced the development of a variety of CT assessment tools.

2.2 Assessments of Computational Thinking

Given that an educational assessment contributes significantly to teaching and learning (Black & Wiliam, 1998; Shepard, 2000), a CT assessment is an integral piece that provides valuable information about student learning progress, as well as the effects of instruction. Although it is difficult to unify in a single assessment, it has been agreed that comprehensiveness of assessment is central to enable educators and researchers to evaluate the effectiveness of CT-incorporated instruction in discipline-specific or multi-disciplinary lessons. Without a comprehensive assessment framework, teachers and students cannot understand how they are teaching and learning in a classroom. Grover et al. (2014) suggested considering multiple complementary measures that can reflect deeper learning and contribute to a comprehensive picture of students' learning in CT education. As the clarity of and discussion on the definitions of CT in education have advanced, several comprehensive frameworks for improving CT assessment have been proposed (e.g., Adams et al., 2018; Brennan & Resnick, 2012; Grover & Pea, 2013, 2018; Roman-Gonzalez, 2015; Shute et al., 2017; Zhong et al., 2016). Today, most frameworks of CT rely primarily on works from both the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education Committee (ISTE; Barr & Stephenson, 2011) and the three-dimensional CT model (Brennan & Resnick, 2012). The CSTA and ISTE model includes CT concepts, capabilities, dispositions and predispositions, and classroom culture. Brennan and Resnick's model consists of computational concepts, practices, and perspectives.

2.3 Roles of Block-Based Programming

Several studies have examined the effectiveness of CT intervention to facilitate CT teaching and learning. Some studies explored instructional approaches with diverse target populations in a variety of educational settings (e.g., Atmatzidou & Demetriadis, 2016; Czerkawski & Lyman, 2015; de Paula et al., 2018; Grover et al., 2015; Jenkins, 2015; Román-González et al., 2015; Romero et al., 2017; Yadav et al., 2014). Shute et al. (2017) classified introductory CS/CT practices into four strategies: (a) programming, (b) robotics, (c) game design/play, and (d) unplugged activities. The National Research Council (2010) highlighted the role of programming in constructing a series of steps for solving a computational problem. As an effort to help programming attract and engage students in computational problem-solving, various block-based programming languages where codes are represented as blocks (e.g., Scratch, Alice, Snap!, App Inventor, LEGO Mindstorms, and Blockly) were introduced as an aid to better understanding CT. Brennan and Resnick (2012) suggested suitable settings in the context of Scratch block-based programming for developing CT capacities aligned with three CT dimensions.

Also, although prior research has been conducted mainly on K-12 CS education, CS/CT should be expanded to college students and lifelong students in terms of providing unique and equal opportunities to develop computational problem-solving skills. This type of CS/CT course is designed for students who typically have no prior experience in programming and only have a general knowledge of computing. Hence, it is significant to identify CT components that are difficult for beginners in learning block-based programming.

2.4 Instruction of Computational Thinking

Although block-based programming provides an engaging introduction to programming, researchers have found that novice learners still have difficulties mastering specific programming concepts. The study conducted by Sentence and Csizmadia (2015) found that programming was effective in enhancing CT but recognized as one of the most challenging learning activities. Duncan and Bell (2015) argued that CT cannot be learned automatically simply by using tools that improve CT competencies in previous studies. In a smiliar study, learners found it difficult to learn programming, and the biggest limitation of CT education is that CT components are difficult to teach due to their abstract nature (Czerkawski & Lyman, 2015). This may be because teachers are rarely cognizant of how to approach computational problem-solving using the abstract concepts. Such lack of readiness for teaching computational concepts hinders teachers' abilities to keep students engaged and on track with more in-depth learning. A few studies suggested instructional approaches for promoting the CT process (Czerkawski & Lyman, 2015; Sentence & Csizmadia, 2015); however, these studies did not present which CT components are likely to be most challenging for learners to engage in learning programming. It is fundamental to identify which areas are most challenging to learn CT via programming. Moreover, CT instruction should be designed for students to attain deeper learning outcomes; thus, it gives rise to a need for studies that provide empirical data for CT leaning and explore practical instructional approaches. One way to advance this area of research is to identify which CT components are difficult for novices to learn.

3. Purpose of the Study

The purpose of this study was to examine computational concepts and practices that novice learners may experience challenges with learning in an online course intended to promote CT competencies as they apply to basic computer skills and programming. Two research questions guided this study:

- RQ1: Which computational thinking concepts are difficult for undergraduate students in an online learning environment?
- RQ2: Which computational thinking practices are difficult for undergraduate students in an online learning environment?

The findings would provide empirical evidence associated with the difficulties in learning CT components for novice learners but also expand discussions about how instructions should be formed to support difficult computational concepts and practices.

3.1 Dimensions of Computational Thinking

When programming with Scratch to facilitate the development of CT, multiple dimensions have been considered. In the framework proposed by Brennan and Resnick (2012) along with the Scratch programming language and environment, three key dimensions involve (a) computational concepts commonly found in programming languages, (b) computational practices referred to as the process of building a solution with the concepts, and (c) computational perspectives as the understandings of relationships with oneself, others, and the world. Each dimension includes different subcomponents, such as seven concepts (i.e., sequences, loops, events, parallelism, conditionals, operators, and data); four practices (i.e., being incremental and iterating testing and debugging, reusing and remixing, abstracting and modularizing); and three perspectives (i.e., expressing, connecting, and questioning).

3.2 Computational Concepts and Practices

Among the three dimensions, this study focused on computational concepts and practices and excluded perspectives due to the constraints on capturing changes in participants' perspectives over a short time period. Table 1 provides a summary of the definitions of CT components targeted in the study.

Table 1. Definitions of CT target components

Dimensions	Definitions
Computational Concepts	Sequences: Executing a series of individual steps or instructions for an activity or task Loops: Repeating the same sequence multiple times Events: Triggering specific actions to happen Parallelism: Performing a sequence of actions in parallel Conditionals: Making a decision based on certain conditions Operators: Expressing mathematical, logical, and string operations Data: Storing, retrieving, and updating values in variables and lists
Computational Practices	Being incremental and iterative: Developing solutions step by step Testing and debugging: Finding strategies for solving problems Reuse and remix: Building new solutions on existing works or ideas Abstraction and modularity: Modeling complex systems with simple elements

Note. Adapted from Brennan and Resnick’s framework (2012).

4. Methods

4.1 Participant Characteristics

A total of 92 undergraduate students who were enrolled in an online course, Computing and Information Technology, at a large public university in the southwestern United States participated in this study. Participants were studying with varied majors, were of various ages and included both males and females (male: 59, female: 33; age range: 19-49; average age = 25.21; SD = 11.32). The students learned a set of core knowledge and skills that shape the landscape of computer science, represent information digitally, and create block-based programs to solve problems. This study was approved by the University Institutional Review Board.

4.2 Research Setting

The course was delivered completely online via a web-based learning management system. The course aimed to deliver a set of core competencies that shape the background of computer science and essential career readiness skills such as critical thinking, problem-solving, and communication. The learning modules were designed to provide students with programming experiences using the Scratch block-based programming language. Scratch programming is intended to be adopted in an introductory CS/CT course for people of all ages and across disciplines (Resnick et al., 2009), and it offers editors both online and offline to make it easy for learners to create and share programming projects. Out of 15 online learning modules, a total of eight modules were related to Scratch programming projects aligned with learning objectives. In each module, programming activities related to computational concepts were provided along with clear instructions and requirements to to clarify the learning process and expectations. Student performance was assessed regularly to ensure students achieved the intended learning outcomes. The programming quizzes and assignments were graded with evaluation criteria, and constructive feedback was provided to foster active participation in the learning process. The research data was collected in the last programming project where learners demonstrated their problem-solving skills through block-based programming. The programming tasks were to complete predesigned and semifinished Scratch programming projects with a set of requirements, but the final project was to program a game with Scratch by applying the CT concepts and skills learned in the previous module.

4.3 Instruments

The computational concepts and practices were assessed by (a) a computational thinking test (CTt; Roman-Gonzalez, 2015) and (b) coding journals. All 92 participants completed the CTt and coding journals. The CTt scale ($\alpha = 0.79$) had significant correlations with other standardized tests on problem-solving skills, and its validity was confirmed for block-based programming learners. The CTt scale includes 28 multiple-choice questions to measure the understanding level of computational concepts (i.e., basic direction and sequences, loops-repeat time, loops-repeat until, if-simple conditional, if/else-complex conditional, while conditional, and simple function). The CTt was initially designed and has been used for research targeting secondary school students (e.g., Bati, 2018; Chan et al., 2021; Guggemos, 2021; Román -González et al., 2017, 2018, 2019; Wiebe et al., 2019) and a few studies have been conducted for undergraduate students (e.g., Cachero et al., 2020; Guggemos et al., 2019; Kousis, 2019). Also, the CTt aims to measure the developmental level of computational problem-solving (Román-González et al., 2017). As the target population was novices on the subject of computer science, we adapted this scale for the study to measure the core computational concepts according to the developmental level of beginner rather than the age level, which may allow further insights. Six of the 11 CT components were covered by the CTt (see Table 2). Since five of the 11 CT components were covered by the CTt, the remaining components were measured through the coding journal.

The coding journal questionnaire for Scratch programming project assignments was developed by the researchers. Open-ended questions are used in CT-related studies to provide insight into the participants' understanding of computational practices (Cetin, 2016; Ozoran et al., 2012). Participants were asked to share their programming experiences with reflective writing in response to four open-ended questions as they performed programming tasks using Scratch: (a) overall programming process or steps to create your program, (b) what worked well during programming, (c) what issues you faced during programming, and (d) what needs to be improved in the next programming project. The coding journal questionnaires were designed to lead the students to validate and embellish on the findings from the CTt responses, which were also helpful in finding what interventions could help improve their learning experiences on computational concepts. Table 2 presents a summary of the measurements deployed to measure computational thinking components.

Table 2. A summary of CT components and corresponding instruments

Dimensions	Components	CTt	Coding Journal
Computational Concept	Sequences	O	O
	Loops	O	O
	Events	O	O
	Parallelism	X	O
	Conditional	O	O
	Operators	O	O
	Data	X	O
Computational Practice	Being incremental and iterative	X	O
	Testing and debugging	X	O
	Reuse and remixing	X	O
	Abstraction and modularity	X	O

Note. Symbol "O" indicates measured; "X" indicates unmeasured.

4.4 Data Collection and Analysis

After completing all computational concept-related activities, an online form of CTt was linked in a module. Participants received an extra point for voluntary participation in the test. Their answers to the CTt items were stored in the database and statistically analyzed. Afterward, we conducted descriptive and repeated measures analysis of variance (ANOVA) analyses for the CTt scores to determine the changes in scores.

In each module, participants used Scratch to perform programming tasks. Their experiences were gathered from the coding journals for the assignment where all computational concepts and practices needed to be applied. A total of 92 coding journals were analyzed by thematic analysis. The authors organized the data and then coded the Scratch coding journals following the three-step guidelines from Miles and Huberman (1994) for deductive thematic analysis: (a) data reduction, (b) data display, and (c) data drawing and conclusion. The qualitative data was coded for the frequencies of different types of CT components and then recoded using iteratively refined codes by two of the researchers with high levels of interrater secured. Their responses were reexamined and categorized into seven computational concepts and four computational practices based on Brennan and Resnick's framework. Finally, tables were created based on the four categories aligned with the journal questions: (a) process, (b) success, (c) challenge, and (d) improvement (see Tables 4–6).

5. Results

5.1 CTt Analysis Results (RQ1: Which computational thinking concepts are difficult for undergraduate students?)

For the first research question, CTt scores showed that the participants' understanding of each CT concept differed considerably. Table 3 shows a summary of the CTt mean scores, of which each subscale ranges from 1 to 4. As shown in Table 3, while "basic direction and sequences" among the seven computational concepts had the highest mean score of 3.29 out of 4 (M = 3.29, SD =1.0); "while conditional" had the lowest mean score of 1.49 (M = 1.49, SD =1.02); followed by "if-simple conditional" (M = 1.75, SD = 1.10); "if/else complex conditional" (M = 2.03, SD = 1.31); "simple function" (M = 2.18, SD = 1.29); "loops-repeat until" (M = 2.68, SD = 1.05); and "loops-repeat time" (M = 3.17, SD = .98). As demonstrated in Table 3, the values of the two computational concepts, "while conditional" and "if conditional," was relatively lower than those of the other concepts. Also, a one-way repeated measures ANOVA was computed to evaluate if there was any change in participants' CT sub-concept scores when measured in the seven computational concepts. The results of the ANOVA indicated a significant effect for the CT concept (Wilks' Lambda = .23, F (6,86) = 47.07, $p < .01$, $\eta^2 = .77$). Also, there was significant evidence that the mean score of each concept was different. Pairwise comparisons indicated that each pairwise difference in scores was significant, $p < .05$, suggesting that participation in the subscale decreased participants' mean scores of CTt subscales. That is, the average score tended to decrease gradually as the difficulty of the CT concept increased. However, there was no statistically significant difference in mean test scores between "simple function" and "if/else complex conditional" ($p = 0.87$).

Table 3. A summary of descriptive analysis results (RQ1)

CTt Concepts	Mean	*SD*
Basic direction & sequences	3.29	1.15
Loops-repeat time	3.17	.98
Loops-repeat until	2.68	1.05
Simple function	2.18	1.29
If/else complex conditional	2.03	1.31
If-simple conditional	1.75	1.10
While conditional	1.49	1.20

5.2 Coding Journal Analysis Results (RQ1 & RQ2: Which computational thinking concepts and practices are difficult for undergraduate students?)

The results of the content analyses from the student coding journals showed the computational concepts and practices areas where participants had difficulties as they programmed with Scratch. The responses to the open-ended questions of the coding journals (i.e., overall process, success, challenge, and improvement) produced a more diverse set of answers. After thoroughly validating the data analysis, a list of difficult computational concept and practice areas for beginners to learn block-based programming online was identified. As shown in Table 4, the most common responses to the open-ended question regarding issues faced during programming were the use of "conditionals" (e.g., if/else and nested conditionals) and "data" (e.g., variables and lists). When asked what needed to be improved in the next programming project, students described the uses of "if/else conditional," "data," and "operators" (e.g., numeric, logical, and string manipulation) when it comes to computational concepts. In contrast, the concepts considered successfully learned were "sequences," "loops," and "events." In terms of "parallelism," in the early simple programming, the codes were parallelized as intended, but as the number of sprites and the complexity of the programs increased, the parallelism tended to become more challenging. Table 4 summarizes the content analysis results for computational concepts. The responses to the first question in the coding journal, overall programming process, were categorized as codes for computational practices.

Table 4. A summary of the content analysis results related to computational concepts (RQ1)

	Concepts	Frequencies	Quotes
Success (N=150)	Sequences	46%	"Programming the correct sequences was easy."
	Events	32%	"What worked well was getting the character to move, look, sound, and event."
	Loops	22%	"Repeat background sound and pauses worked very well."
Challenge (N=182)	Data	37%	"Creating a new variable and list caused me to re-write the code several times."
	Conditionals	33%	"I am facing a lot of simple mistakes when I initially use control blocks such as if/else and repeat until."
	Parallelism	30%	"I struggled to know how to run simultaneously with the multiple movements."

Improvement (N=110)	Conditionals	41%	"I would like for my next programming project to flow better with no issues."
	Data	36%	"The only difficulty that I faced during the process was that it was hard for me to place the correct variable in order to keep the correct commands consistent."
	Operators	23%	"I want to be more comfortable with the operators and I think continuing to explore more operators and use more in depth."

Note. Values in percent indicate relative frequencies.

In addition, concerning the computational practice in programming, a summary of the content analysis results is presented in Table 5. First, as a result of analyzing the responses to the overall process for the programming project, participants described the process as incremental and iterative by approaching and developing a solution in small steps. Second, although participants perceived that they were doing best in "reusing and remixing" (i.e., building on their own or others' work), "testing and debugging" (i.e., trial and error, fixing an error) was reflected as the most difficult computational practice element even after they had attempted a number of trials and errors. For instance, participants most often expressed, "I cannot see where I'm making a mistake to fix it," or "I know the problem, but I don't know how to solve it," or "I spent a lot of time and effort trying to solve the problem, but I can't solve it." Last, "abstracting and modularity" was the most frequent response as computational practice when participants were asked what they wanted to improve for the next Scratch project. Participants wanted to find more ways to efficiently abstract solutions by analyzing problem patterns to solve problems. They also wanted to improve in converting their solutions efficiently. Table 5 presents example quotes from the coding journal regarding computational practices.

Table 5. A summary of the content analysis results related to computational practices (RQ2)

	Practices	Frequencies	Quotes
Process (N=131)	Being incremental & iterating	61%	"The process I used to create my program was to first read through the blackboard instructions and understand the steps to create. After this, I began to create the project by developing a project in small steps."
	Remixing & reusing	25%	
	Testing & debugging	9%	
	Abstracting & modularizing	5%	
Success (N=103)	Remixing & reusing	65%	"What worked well during programming was remixing. I looked at our starter and example projects several times as well as looked at other students that have created Scratch projects similar."
	Being incremental & iterating	26%	
	Testing & debugging	9%	
Challenge (N=74)	Testing & debugging	72%	"I attempted multiple different methods to complete this task but for some reason I was not able to successfully execute."
	Abstracting & modularizing	28%	
Improvement (N=114)	Abstracting & modularizing	65%	"The most used block was the if blocks. A new block that became very helpful for me were the created blocks. It saved a lot of room and time when building collections of codes."
	Testing & debugging	35%	

Note. Values in percent indicate relative frequencies.

6. Discussion and Implications

This study aimed to identify the computational concept and practice components that learners may have difficulties learning with online programming, to lay the groundwork for an effective teaching approach. Along with Brenan and Resnick's dimensional framework (2012), the CTt scale provided meaningful results for the understanding of computational concepts. Through the coding journal analysis, information on achievements in computational concepts and practices were obtained. In particular, the differences in learning were revealed in some concepts and practices of computational thinking. The results from the two data analyses showed that the relatively easy CT concepts were "sequences," "loops," and "events," and relatively easy CT practices were "being incremental and iterating" and "reusing and remixing." Conversely, four concepts (i.e., parallelism, conditionals, data, and operators) and two practices (i.e., testing and debugging and abstracting and modularizing) were identified as difficult CT components to achieve in block-based programming. In particular, the problems of using "conditionals" were consistent with the results of the coding journal analysis in that all of the CTt scores on the "conditionals" (i.e., if-simple conditional, if/else complex conditional, and while conditional) were low.

Findings suggest that educators should pay more attention to the levels of learning difficulty of the computational concepts—"parallelism" (e.g., complex sets of activities in parallel); "conditionals" (e.g., if-simple conditional, and if/else complex conditional, and while conditional); "data" (e.g., variables and lists); and "operators" (e.g., numeric and string manipulation). Also, to facilitate the process of CT development in practice, instructions should incorporate the elements of computational practices (e.g., testing and debugging and abstracting and modularizing). Instructional approaches can be suitable for the difficulty level of the computational concepts and practices. The following instructional approaches can be considered.

Table 6. A summary of the key findings

	CT Concepts from CTt and Coding Journals	CT Practice from Coding Journals
Process	N/A	#1 Being incremental and iterative
		#2 Remixing & reusing
Success	#1 Sequences	#1 Remixing & reusing
	#2 Events	#2 Being incremental and iterative
	#3 Loops	
Challenge	#1 Data	#1 Testing and debugging
	#2 Conditional	#2 Abstracting & modularizing
	#3 Parallelism	
Improvement	#1 Conditional	#1 Abstraction and modularity
	#2 Data	#2 Testing and debugging
	#3 Operators	

First, participants had difficulty as the complexity of concepts increased. Since the biggest limitation of CT instruction is that CT is difficult to teach due to its abstract concepts (e.g., parallelism, conditionals, data, and operators), unplugged activities can help novice learners gain a deeper conceptual understanding of abstract computation concepts and develop an algorithmic solution on paper. For example, storyboard, decomposition sheet, flowchart, pseudo code, and/or journal entry can aid in understanding challenging computational concepts (e.g., Looi et al., 2018). These unplugged activities are suitable for novice programming learners to build difficult computational concepts and develop difficult computational practices gradually. Unplugged activities build student insight into the meaning of blocks, rather than copying a set of blocks and running it (e.g., Brackmann et al., 2017; Caeli & Yadav, 2020).

Second, explicit instruction can address challenges learners face when learning difficult computational concepts and practices. For example, direct instruction is a way to teach concepts and skills to novice students using direct and structured instruction that explains, demonstrates, and models what learners do. In particular, direct instruction is effective when background knowledge is low and the task is complex (Kroesbergen et al., 2004, Rupley et al., 2009). When complex computational concepts and practices are broken down into adaptable chunks, instructors can evaluate students' understanding more precisely by teaching codes one line at a time. Students can practice the skills to increase their understanding of concepts by observing and experimenting with the assistance of the teacher. After guided practice, students need to apply it independently in their use of the concept and skills.

Third, CT instructions should be differentiated for high- and low-achieving students when teaching complex concepts and practices. High-achieving learners are likely to have more prior knowledge and existing schemas for constructing new information. Low-achieving learners need support, repetition, and motivating activities, such as constructive feedback and gamification including choice, rewards, experience points, and level up (e.g., Standford et al., 2010). Besides, the scope and sequence of CT instruction should be presented depending on the difficulty level of domains and tasks (e.g., Tomlinson, 2012). Learners' knowledge background and proficiency should be considered in designing CT instructions with technology. Even non-CS college students need help to understand complex concepts in order to solve computational problems.

Fourth, novice learners should have opportunities to learn how to build computational practices. A complete understanding of computational concepts does not mean that computational practice can be acquired naturally. Since computational problem-solving requires an incremental and iterative process, novices need to learn relevant strategies (e.g., planning multiple phases of development, dividing functions or processes in a program). Debugging usually starts by looking into what should happen, but beginners may have a hard time locating the problem (McCauley et al., 2008). Debugging strategies (e.g., checking invalid values/operations, order of codes, time between blocks) help beginners troubleshoot the problems. Also, they should be encouraged to accept failures as part of their learning process and understand that such experiences help them find the right solution. As shown in Table 5, for students who have tried several different attempts to solve a problem but cannot successfully execute, debugging strategies and tips as a scaffolding should be in place in case they give up without solving the problem. Moreover, as novices advance their computational practices, the CT instructions should include exercises on abstraction and modularization strategies (e.g., simplifying a program, dividing code blocks).

Last, learners should be encouraged to reflect on and share their CT learning experiences with other classmates. Collaboration was incredibly beneficial, particularly to students with minimal programming experience (Denner et al., 2014). In activities related to reuse and remixing (see Table 5), students responded that they benefited from seeing other students' coding blocks or ideas when developing a solution. Collaborative experiences include brainstorming solutions, planning the uses of code blocks, developing algorithms, and fixing errors in pairs. The collaborative learning experience is advantageous not just for developing programming knowledge, but for building other skills critical to solving problems, especially considering that first programming experiences are not offered equally to all.

7. Conclusion

As CS/CT education has gained growing recognition in many disciplines, it is necessary to carefully prepare for its integration to make the leap from block-based programming to problem-solving. However, educators were neither confident in the subject matter nor differentiated it sufficiently for a mixed-ability group (Sentence & Csizmadia, 2015). The evidence from this study confirmed what computational concepts and practices novice learners might struggle with. We discussed how instructions need to be shaped to assist novices in improving CT learning in an online environment. The findings of this study underlined that CT-related learning activities should offer additional instructional support to enhance the understanding of challenging computational concepts and practices. It is hoped that educators will close instructional gaps in what their students struggle with to construct difficult computational concepts and fully practice new solutions with what they already know. Further studies are needed to investigate the effects of instructional approaches to these identified CT components.

Limitations

The empirical results reported herein should be treated with caution. First, the study is limited in that the programming task did not require a design-based activity and did not ask for differences in perceptions of CT perspectives. Future studies, therefore, should focus on deepening our understanding of how CT learning processes occur in creative programming tasks and how the computational perspective helps teachers and learners understand themselves and their communities. Second, the CTt scale used in this study was found to partially measure the components of Brennan and Resnick's CT concept and practices. That is, the CTt did not contain or fit some computational concepts (e.g., parallelism, data, and operator) and computational practices (e.g, being incremental and iterating, reusing and remixing, abstracting and modularizing). Further research is needed to include these sub-components on the scale. Third, to be more valid with a different population and other settings, the study may need to be repeated to support the results.

References

Adams, C., Cutumisu, M., & Lu, C. (2019). Measuring K-12 computational thinking concepts, practices and perspectives: An examination of current CT assessments. In *Society for Information Technology & Teacher Education International Conference* (pp. 275-285). Association for the Advancement of Computing in Education (AACE). https://www.learntechlib.org/primary/p/207654

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, *55*(7), 832-835. https://doi.org/10.1093/comjnl/bxs074

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, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, *38*(6), 20-23. https://edtechbooks.org/-HQ

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?. *Inroads*, *2*(1), 48-54. https://doi.org/10.1145/1929887.1929905

Bati, K. (2018). Computational Thinking Test (CTT) for middle school students. *Akdeniz Eğitim Araştırmaları Dergisi*, *12*(23), 89-101. https://doi.org/10.29329/mjer.2018.138.6

Berland, M., & Wilensky, U. (2015). Comparing virtual and physical robotics environments for supporting complex systems and computational thinking. *Journal of Science Education and Technology*, *24*(5), 628-647. https://doi.org/10.1007/s10956-015-9552-x

Black, P., & Wiliam, D. (1998). Assessment and classroom learning. *Assessment in Education: principles, policy & practice*, *5*(1), 7-74. https://doi.org/10.1080/0969595980050102

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017, November). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (pp. 65-72). https://doi.org/10.1145/3137065.3137069

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada, 1,* 25. https://www.media.mit.edu/publications/new-frameworks-for-studying-and-assessing-the-development-of-computational-thinking

Cachero, C., Barra, P., Meliá, S., & López, O. (2020). Impact of programming exposure on the development of computational thinking capabilities: An empirical study. *IEEE Access*, *8*, 72316-72325. https://doi.org/10.1109/ACCESS.2020.2987254

Caeli, E. N., & Yadav, A. (2020). Unplugged approaches to computational thinking: A historical perspective. *TechTrends*, *64*(1), 29-36. https://doi.org/10.1007/s11528-019-00410-5

Cetin, I. (2016). Preservice teachers' introduction to computing: Exploring utilization of scratch. *Journal of Educational Computing Research*, *54*(7), 997-1021. https://doi.org/10.1177/0735633116642774

Chan, S. W., Looi, C. K., & Sumintono, B. (2021). Assessing computational thinking abilities among Singapore secondary students: A rasch model measurement analysis. *Journal of Computers in Education*, *8*(2), 213-236. https://doi.org/10.1007/s40692-020-00177-2

Creswell, J. W., & Clark, V. L. P. (2017). *Designing and conducting mixed methods research*. Sage publications.

CSTA & ISTE. (2011). *Operational definition of computational thinking for k-12 education*. http://csta.acm.org/curriculum/sub/currfiles/compthinkingflyer.pdf

Czerkawski, B. C., & Lyman, E. W. (2015). Exploring issues about computational thinking in higher education. *TechTrends, 59*(2), 57-65. https://doi.org/10.1007/s11528-015-0840-3

Denning, P. J., & Tedre, M. (2021). Computational thinking: A disciplinary perspective. *Informatics in Education*, *20*(3), 361-390. https://10.15388/infedu.2021.21

de Paula, B. H., Burn, A., Noss, R., & Valente, J. A. (2018). Playing Beowulf: Bridging computational thinking, arts and literature through game-making. *International journal of child-computer interaction*, *16*, 39-46. https://doi.org/10.1016/j.ijcci.2017.11.003

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students?. *Journal of Research on Technology in Education*, *46*(3), 277-296. https://doi.org/10.1080/15391523.2014.888272

Duncan, C., & Bell, T. (2015). A pilot computer science and programming course for primary school students. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, 39-48. ACM. https://doi.org/10.1145/2818314.2818328

General, U. S. (2019). The age of digital interdependence. *Report of the UN Secretary-General's High-Level Panel on Digital Cooperation.* https://www.un.org/en/pdfs/DigitalCooperation-report-for%20web.pdf

Google. (2016). *Computational thinking for educators*. https://edu.google.com/resources/programs/exploring-computational-thinking

Grover, S., Cooper, S., & Pea, R. (2014). Assessing computational learning in K-12. In *Proceedings of the 2014 conference on innovation & technology in computer science education*. 57-62. ACM. https://doi.org/10.1145/2591708.2591713

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. (2018). Computational thinking: A competency whose time has come. *Computer Science Education: Perspectives on teaching and learning in school. London: Bloomsbury Academic*, 19-37. https://www.researchgate.net/profile/Shuchi-Grover-2/publication/322104135_Computational_Thinking_A_Competency_Whose_Time_Has_Come/links/5a457813a6fdcce1971a5ce5/Computational-Thinking-A-Competency-Whose-Time-Has-Come.pdf

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

Guggemos, J. (2021). On the predictors of computational thinking and its growth at the high-school level. *Computers & Education*, *161*, Article 104060. https://doi.org/10.1016/j.compedu.2020.104060

Guggemos, J., Seufert, S., & Román-González, M. (2019). Measuring computational thinking-Adapting a performance test and a self-assessment instrument for german-speaking countries. *International Association for Development of the Information Society*. https://eric.ed.gov/?id=ED608655

Kousis, A. (2019). The impact of educational robotics on teachers' computational thinking. *Educational Journal of the University of Patras UNESCO Chair*. https://doi.org/10.26220/une.3085

Kroesbergen, E. H., Van Luit, J. E., & Maas, C. J. (2004). Effectiveness of explicit and constructivist mathematics instruction for low-achieving students in the Netherlands. *The Elementary School Journal*, *104*(3), 233-251. https://doi.org/10.1086/499751

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

Jenkins, C. (2015). A work in progress paper: Evaluating a microworlds-based learning approach for developing literacy and computational thinking in cross-curricular contexts. In *Proceedings of the Workshop in Primary and Secondary Computing Education,* 61-64. ACM. https://doi.org/10.1145/2818314.2818316

Johnson, R. B., Onwuegbuzie, A. J., & Turner, L. A. (2007). Toward a definition of mixed methods research. *Journal of Mixed Methods Research, 1*, 112-133. https://doi.org/10.1177/1558689806298224

Looi, C. K., How, M. L., Longkai, W., Seow, P., & Liu, L. (2018). Analysis of linkages between an unplugged activity and the development of computational thinking. *Computer Science Education*, *28*(3), 255-279. https://doi.org/10.1080/08993408.2018.1533297

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: a review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67-92. https://doi.org/10.1080/08993400802114581

Miles, M. B., Huberman, A. M., Huberman, M. A., & Huberman, M. (1994). *Qualitative data analysis: An expanded sourcebook*. Sage publications. https://doi.org/10.1016/0149-7189(96)88232-2

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *Revista de Educación a Distancia*, 46, 1-23. https://doi.org/10.6018/red/46/10

National Research Council. (2010). *Report of a workshop on the scope and nature of computational thinking.* National Academies Press. https://doi.org/10.17226/12840

National Science Foundation. (2016) Computer science for all (CSforAll:RPP)(Dec. 1 2016). https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=505359

Ozoran, D., Cagiltay, N., & Topalli, D. (2012). Using scratch in introduction to programming course for engineering students. In *2nd International Engineering Education Conference. 2,* 125-132. https://www.academia.edu/25922529/Using_Scratch_in_introduction_to_programming_Course_for_Engineering_Students

Papert, S. (1980). *Mindstorms: Children, computers and powerful ideas*. New York: Basic Books. https://doi.org/10.5555/1095592

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. B. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60-67. https://web.media.mit.edu/~mres/papers/Scratch-CACM-final.pdf

Román-González, M. (2015). Computational thinking test: Design guidelines and content validation. In *Proceedings of EDULEARN15 Conference,* 2436-2444. https://doi.org/10.13140/RG.2.1.4203.4329

Román-González, M., Moreno-León, J., & Robles, G. (2017). Complementary tools for computational thinking assessment. In *Proceedings of International Conference on Computational Thinking Education, S. C Kong, J Sheldon, and K. Y Li (Eds.). The Education University of Hong Kong,* 154-159. https://doi.org/10.1007/978-981-13-6528-7_6

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in human behavior, 72*, 678-691. https://doi.org/10.1016/j.chb.2016.08.047

Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2016). Does computational thinking correlate with personality?: The non-cognitive side of computational thinking. In *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality,* 51-58. ACM. https://doi.org/10.1145/3012430.3012496

Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018). Can computational talent be detected? Predictive validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction, 18*, 47-58. https://doi.org/10.1016/j.ijcci.2018.06.004

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

Romero, M., Lepage, A., & Lille, B. (2017). Computational thinking development through creative programming in higher education. *International Journal of Educational Technology in Higher Education, 14*(1), 42. https://doi.org/10.1186/s41239-017-0080-z

Rupley, W. H., Blair, T. R., & Nichols, W. D. (2009). Effective reading instruction for struggling readers: The role of direct/explicit teaching. *Reading & Writing Quarterly*, *25*(2-3), 125-138. https://doi.org/10.1080/10573560802683523

Seiter, L., & Foreman, B. (2013). Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth annual international ACM conference on International computing education research*, 59-66. ACM. https://doi.org/10.1145/2493394.2493403

Sentance, S., & Csizmadia, A. (2015). Teachers' perspectives on successful strategies for teaching computing in school. In *IFIP TCS.* https://www.researchgate.net/profile/Sue-Sentance/publication/301525438_Teachers%27_perspectives_on_successful_strategies_for_teaching_Computing_in_school/links/57176e3708ae2679a8c76745/Teachers-perspectives-on-successful-strategies-for-teaching-Computing-in-school.pdf

Shepard, L. A. (2000). The role of assessment in a learning culture. *Educational researcher*, *29*(7), 4-14. https://doi.org/10.3102/0013189X029007004

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

Stanford, P., Crowe, M. W., & Flice, H. (2010). Differentiating with technology. *TEACHING exceptional children plus*, *6*(4), 4. https://files.eric.ed.gov/fulltext/EJ907030.pdf

Teddlie, C., & Tashakkori, A. (2003). Major issues and controversies in the use of mixed methods in the social and behvioral sciences. *Handbook of mixed methods in social & behavioral research*, 3-50. https://doi.org/10.4135/9781506335193

Tomlinson, C. A. (2012). *Differentiated instruction* (pp. 307-320). Routledge. http://www.casenex.com/casenex/ericReadings/DifferentiationOfInstruction.pdf

United Nations (2019). The age of digital interdependence. *Report of the UN Secretary-General's*

High-level Panel on Digital Cooperation. https://www.un.org/en/pdfs/DigitalCooperation-report-for%20web.pdf

Wiebe, E., London, J., Aksit, O., Mott, B. W., Boyer, K. E., & Lester, J. C. (2019). Development of a lean computational thinking abilities assessment for middle grades students. In *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 456-461). https://doi.org/10.1145/3287324.3287390

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. The Magazine of Carnegie Mellon University's School of Computer Science*. Carnegie Mellon University, School of Computer Science. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)*, *14*(1), 5. https://doi.org/10.1145/2576872

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research*, *53*(4), 562-590. https://doi.org/10.1177/0735633115608444