



Volume 5, No: 3  
April 2022  
ISSN 2513-8359

# International Journal of Computer Science Education In Schools

## Editors

Dr Filiz Kalelioglu

Dr Yasemin Allsop

Andrew Paul Csizmadia

# International Journal of Computer Science Education in Schools

**April 2022, Vol 5, No 3**

**DOI: 10.21585/ijcses.v5i3**

## **Table of Contents**

	Page
<b>Hyunchang MOON<sup>1</sup>, Jongphil CHEON<sup>2</sup>, Kyungbin KWON<sup>3</sup></b> Difficult Concepts and Practices of Computational Thinking Using Block-Based Programming	3 - 16
<b>Osman Gazi YILDIRIM<sup>1</sup>, Nesrin OZDENER<sup>2</sup></b> The Development and Validation of the Programming Anxiety Scale	17 - 34
<b>Ünal ÇAKIROĞLU<sup>1</sup>, Şüheda MUMCU<sup>1</sup>, Melek ATABAY<sup>1</sup>, Merve AYDIN<sup>1</sup></b> Understanding Problem-Solving Processes of Preschool Children in CS- Unplugged Activities	35 - 53

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

**Hyunchang MOON<sup>1</sup>**  
**Jongphil CHEON<sup>2</sup>**  
**Kyungbin KWON<sup>3</sup>**

<sup>1</sup> Baylor University, Waco, USA

<sup>2</sup> Texas Tech University, Lubbock, USA

<sup>3</sup> 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 similar 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 learning 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
Computational Practices	Data: Storing, retrieving, and updating values in variables and lists
	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 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 Brennan 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 #2 Events #3 Loops	#1 Remixing & reusing #2 Being incremental and iterative
Challenge	#1 Data #2 Conditional #3 Parallelism	#1 Testing and debugging #2 Abstracting & modularizing
Improvement	#1 Conditional #2 Data #3 Operators	#1 Abstraction and modularity #2 Testing and debugging

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/comptinkingflyer.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](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](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](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](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](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](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>
- Sentence, S., & Csizmadia, A. (2015). Teachers' perspectives on successful strategies for teaching computing in school. In *IFIP TCS*. [https://www.researchgate.net/profile/Sue-Sentence/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](https://www.researchgate.net/profile/Sue-Sentence/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 behavioral 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>



# The Development and Validation of the Programming Anxiety Scale

Osman Gazi YILDIRIM<sup>1</sup>  
Nesrin OZDENER<sup>2</sup>

<sup>1</sup> National Defense University Army NCO Vocational College/Turkey

<sup>2</sup> Marmara University/Turkey

DOI: 10.21585/ijcses.v5i3.140

## Abstract

The main goal of the current study is to develop a reliable instrument to measure programming anxiety in university students. A pool of 33 items based on extensive literature review and experts' opinions were created by researchers. The draft scale comprised three factors applied to 392 university students from two different universities in Turkey for exploratory factor analysis. The number and character of the underlying components in the scale were determined using exploratory factor analysis. After exploratory factor analysis, confirmatory factor analysis was conducted on the draft scale using a sample of 295 university students. Confirmatory factor analysis was carried out to ensure that the data fit the retrieved factor structure. The internal consistency coefficient (Cronbach's alpha) was calculated for the full scale and each dimension for reliability analysis. For convergent validity, the factor loading of the indicator, the average variance extracted, composite reliability, and maximum share variance values were calculated. Additionally, convergent validity was tested through (1) comparison of mean values of factors and total programming anxiety depending on gender and (2) correlation analysis of factors, total programming anxiety, and course grade of students. The Fornell & Larcker criterion and the Heterotrait-Monotrait correlation ratio were utilized to assess discriminant validity. According to analysis results, the Programming Anxiety Scale (PAS) comprised 11 items in two factors: classmates and self-confidence. Similarly, results revealed that The PAS has good psychometric properties and can be used to assess the programming anxiety of university students.

**Keywords:** computer programming, programming anxiety, scale development, scale validation

## 1. Introduction

With the development of the internet and mobile technologies in the last two decades, breakthroughs have been experienced in many fields of computer science such as big data, artificial intelligence, blockchain, bioinformatics, wearable technologies, cloud computing, 3D printers, robotics, and virtual reality. Responsibilities of computer science such as automating the processes, facilitating communication, providing better products and services, assisting the world to be more productive have caused human beings to be more dependent on software (Santos, Tedesco, Borba, & Brito, 2020). As a result, all developed and developing countries are required to raise qualified individuals who can maintain the software used and produce practical solutions to new problems encountered in the future (Demirer & Sak, 2016). One of the conditions for the success of this task is to provide students with programming skills, which is considered one of the requirements of being a well-educated and knowledgeable citizen (Al-Makhzoomy, 2018; Kert & Uğraş, 2009). However, according to several studies, most computer science students regard programming courses as complicated and intimidating (Bennedsen & Caspersen, 2007; Connolly, Murphy, & Moore, 2009; Jenkins, 2002; Owolabi, Olanipekun, & Iwerima, 2014; Robins, Rountree, & Rountree, 2003; Wiedenbeck, Labelle, & Kain, 2004). Moreover, studies show that programming courses have high dropout and failure rates (Bennedsen & Caspersen, 2007; Luxton-Reilly et al., 2019).

Over the last decade, numerous research has been undertaken on the factors affecting learner success in programming courses. Previous studies indicate that programming background (Bunderson & Christensen, 1995; Byrne & Lyons, 2001), mathematical knowledge (Butcher & Muth, 1985; Wilson & Shrock, 2001), problem-solving skills (Gibbs, 2000; Hostetler, 1983), learning styles (Byrne & Lyons, 2001; Tan, Ting, & Ling, 2009), expectations of students for course outcome (Rountree, Rountree, & Robins 2002), comfort level (Bergin & Reilly, 2005) and self-efficacy (Ramalingam & Wiedenbeck, 1998) impact achievement of students in programming courses. Similarly, programming anxiety is also a significant predictor of achievement in programming (Connolly et al., 2009; Maguire, Maguire, & Kelly, 2017).

### 1.1 Related Work

Connolly, Murphy, and Moore (2007) define programming anxiety as a situation-specific psychological state caused by negative experiences or expectations in a computer programming situation. Connolly et al. (2007) also claim that programming anxiety is caused by the incorrect self-assessment of students' abilities when learning to program. According to Scott (2015), students often encounter programming anxiety at the initial stages of programming courses because programming courses involve concepts and materials that are "radically novel" (Dijkstra, 1989). Moreover, as beginning programming courses have become more abstract over the last few decades, programming anxiety has increased (Connolly et al., 2009). This particular content can evoke intense negative feelings (Huggard, 2004) such as confusion, frustration, and boredom (Bosch, D'Mello, and Mills, 2013) which is described as a phenomenon called "programming trauma" (Huggard, 2004).

Since learners' self-belief plays a fundamental role in intellectual development (Berland & Lee, 2011; Pajares, 1992), Jiang, Zhao, Wang, and Hu (2020) believe that this trauma happens when students lose their self-efficacy in programming, which negatively affects learning outcomes. Connolly et al. (2007) propose a cognitive model to explain how programming anxiety influences students' emotional, behavioral and physiological reactions (see Figure 1). The mental model asserts that students' automatic thoughts are activated in programming situations, directly influenced by their core and intermediate beliefs. Eventually, automatic thoughts affect their emotional, behavioral, and physiological reactions. According to Connolly et al. (2007), a fear of programming may commence caused by core beliefs for a student sensitive to programming anxiety. Then, intermediate thoughts of students could emerge as a fear of what other students might think about their performance and ability. Finally, automatic thoughts arise in programming situations and trigger negative thoughts and reactions.

In addition to the cognitive model for programming anxiety, Rogerson and Scott (2010) also depict an iceberg model to explain factors affecting fear of programming. According to the iceberg model, the fear of programming is induced due to the nature of programming. Rogerson and Scott (2010) cite that internal factors such as motivation, attitude, self-efficacy, and attribution often have a part in building negative perceptions of programming. At the same time, peers, teaching methodology, timing, lectures, and tutors constitute external factors.

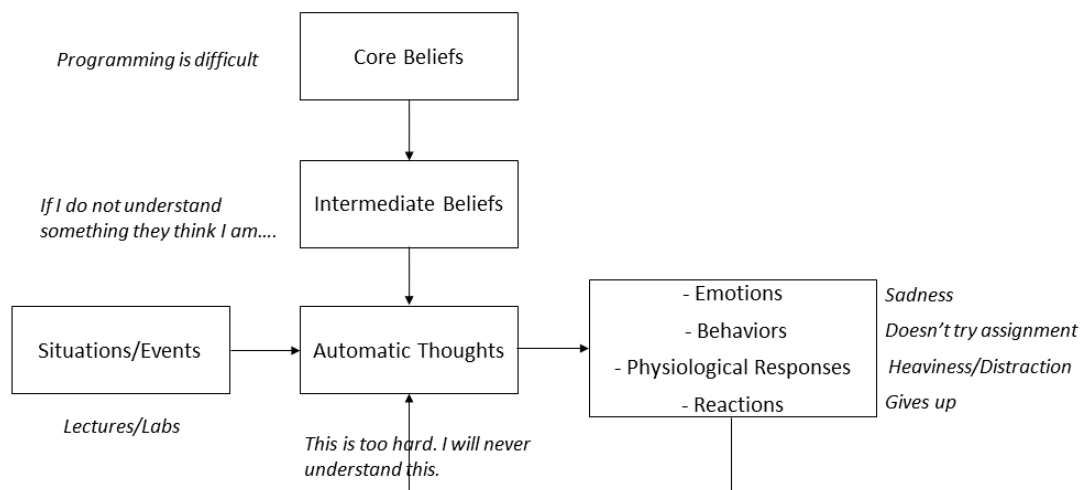


Figure 1. Proposed Cognitive Model for Programming Anxiety (Connolly et al., 2007)

The number of studies on programming anxiety has risen dramatically in the last decade. Some of these studies examined factors associated with programming anxiety, while others investigated the impact of programming anxiety on student performance and motivation. According to S Sinožić and Orehovaki (2018), the absence of programming experience, fear of programming, and a misperception of programming languages as very complex are all powerful determinants of programming anxiety among novices. Similarly, unfamiliar subjects in programming courses make students avoid programming, and programming makes them feel uncomfortable (Olipas, Leona, Villegas, Cunanan & Javate, 2021). According to studies, learners' programming anxiety levels aggregated as they were presented to programming concepts and principles. (Campbell, 2018; Dasuki & Quaye, 2016).

Several studies have also connected programming anxiety to academic performance, perceived self-efficacy, encountering errors when developing programs, gender, peers, test anxiety, mathematics, and computer anxiety. For example, Olipas et al. (2021) found a negative association between participants' academic performance and programming anxiety in a study of 348 students. Hsu and Gainsburg (2021) and Wilfong (2006) explain that self-efficacy plays a vital role in performance in programming courses, and self-efficacy has a mediating effect on the relationship between anxiety and performance. Results of a systematic review of the literature conducted by Nolan and Bergin (2016) illustrate correlates of programming anxiety as programming as a subject, test anxiety, computer anxiety (volume of computer usage), and using mathematics frequently in coding. Additionally, the students' incapacity to debug their programs increase their programming anxiety (Dasuki & Quaye, 2016; Nolan & Bergin, 2016).

Some researchers mention the effects of peers on programming anxiety. According to Nolan and Bergin (2016), when programming students learn to program in a laboratory with many peers, this circumstance can be stressful. Falkner, Falkner, and Vivian (2013) explored how collaborative practices in programming courses can cause fear and tension in learners. They concluded that working in groups prevented students from feeling comfortable in classes. There are also studies in the programming literature on the effects of gender on programming anxiety. According to Olipas and Luciano's (2020) study, female students show more programming anxiety than male students. Chang (2005) also explored a possible association between the perceived complexity of programming tasks and programming anxiety with 307 participants. According to the findings, there was a strong association between these two variables, indicating that as the perceived complexity of programming assignments increased, so did students' perceived programming anxiety levels.

Many studies in the literature state that programming anxiety is one of the factors that cause students to fail and lose interest in programming courses. It is reported that programming anxiety is critical in determining students' success in a programming course (Connolly et al., 2007; Figueroa & Amoloza, 2015; Kinnunen & Malmi, 2006; Nolan, Bergin & Mooney, 2019; Owolabi et al., 2014; Scott, 2015). With self-beliefs being the case, Kinnunen and Simon (2012) assert that learners' self-beliefs are developed due to the experiences students have while they engage in programming activities rather than the resulting quality of the programs they write. As a consequence of negative experiences and self-appraisals, learners lack the time or have no motivation to program (Kinnunen & Malmi, 2006; Scott, 2015). Similarly, Maguire et al. (2017) assert that programming anxiety causes a lack of confidence and plays a crucial role in discouraging students from carrying out programming independently. Results of the study of Özmen and Altun (2014) show that while students with a low level of programming anxiety spend extra time on programming and code more qualified programs, students with a high level of anxiety devote limited time on programming practices and avoid learning programming. Similar results have been cited by Scott (2015), concluding that programming anxiety inhibits time spent practicing programming and decreases course participation (Bergin & Reilly, 2005). Scott and Ghinea (2014) investigated the possible adverse effects of programming anxiety on students' programming practice. Participants of the study were 239 university students. The findings revealed that students are frequently concerned when undertaking debugging activities.

In the light of all these studies in the literature, it is essential to measure the programming anxiety levels of students with reliable and valid instruments to determine students' anxiety levels and help learners overcome their anxiety and frustration in programming courses. However, despite anxiety's critical role in programming, research on anxiety scale development has been deficient. Information about these measurement instruments is summarized in Table 1.

Table 1. A Summary of the Relevant Scales

Name of the Scale/Survey	Factors	The Total Number of Items
Programming Anxiety Survey (Figuroa & Amoloza, 2015)	- Not Applicable	6
The Computer Programming Anxiety Questionnaire (Connolly et al., 2009)	- Gaining Initial Computing Skills - Sense of Control - Computer Self Concept - State of Anxiety in Computer Situations	15
The Computer Programming Anxiety Scale (Choo & Cheung, 1991)	- Errors - Significant Others - Confidence	19

As presented in Table 1, three scales are prepared to measure primarily programming anxiety. All of the scales are based on self-reported data. In addition to these scales, it was noted that computer anxiety or information technology (IT) anxiety scales were adapted for measuring programming anxiety in several studies (see Olipas & Luciano, 2020; Scott & Ghinea, 2014, and Orehovacki, Radosevic & Konecki, 2012). Furthermore, Demir (2021) recently adapted Choo and Cheung's (1991) programming anxiety scale into Turkish.

### *1.2 Purpose of the Study*

Studies show that reducing anxiety can enhance academic performance and achievement (Hattie, 2008). The same is true when it comes to improving the efficiency of programming courses. It is vital to identify learners' programming anxiety and work closely with students with high anxiety to develop the learning outcomes of programming courses at the highest level. In this sense, there is a need for reliable measurement tools designed to measure programming anxiety to make meaningful conclusions from the analysis. As a result, the current research aims to create a proper and reliable tool to measure programming anxiety in university students.

## **2. Method**

The Computer Programming Anxiety Scale was developed and validated in three phases, illustrated in Figure 2. In summary, dimensions of the draft scale were identified, and the item pool was generated in the first phase. In the second phase, content and phase validity were assessed. In the last stage, exploratory and confirmatory factor analysis was conducted, and construct validity was evaluated.

### *2.1 Phase 1: Identifying dimensions & item generation*

Clark and Watson (1995) recommend beginning scale development by clearly conceptualizing the target construct and clarifying its breadth and scope. The researchers conducted a comprehensive literature review and content analysis to identify different dimensions of programming anxiety. With this respect, models and explanations related to programming anxiety were examined to develop a clear conceptualization. Furthermore, related constructs including computer anxiety, math anxiety, and test anxiety were investigated. The Computer Programming Anxiety Scale (Choo & Cheung, 1991) was used to identify programming anxiety dimensions. At the same time, scales developed to measure students' anxiety, such as programming anxiety, computer anxiety, test anxiety, math anxiety, and foreign language learning anxiety, were investigated. Depending on the studies on programming anxiety, three dimensions were proposed: (1) classmates, (2) self-confidence, and (3) errors. The "Classmates" subscale measures students' anxiety in the presence of more proficient students. The "Programming confidence" subscale measures students' feelings of inadequacy while programming. "Errors" subscale measures students' anxiety when confronted with errors during programming.

Next, a pool of 33 items was constructed to capture negative emotions during program development and debugging. The rationale underpinning including as many items as possible in the draft scale was that the number of items at the start should be twice as numerous as the final scale (Nunnally, 1994). To obtain precise and unambiguous items that reflect the specified conceptual definitions, item wording rules suggested by

Carpenter (2018) were applied. This cyclical item development procedure yielded a total of 33 items, each rated on a 5-point scale from 1 ("never true") to 5 ("always true"). In this regard, "seldom true" was scored as 2, "sometimes true" was 3, and "often true" was 4.

## *2.2 Phase 2: Development of the scale*

### *2.2.1 Content validity*

Content validity of the draft scale was tested by interviewing five experts, three of whom were from the field of instructional technology, one from the field of Turkish language, and one from the field of psychological counseling and guidance. An expert opinion form was created in this phase. The experts were requested to rate each scale item using this form on a four-point rating scale (1 = not relevant; 2 = item requires so much revision that it is no longer relevant; 3 = item is suitable but needs minor changes; 4 = highly relevant). Data gathered from the expert opinion was used to quantify the content validity process and calculate Content Validity Index (I-CVI; Polit, Beck, & Owen, 2007). The I-CVI was calculated by dividing the experts who provided a 3 or 4 by the total number of experts for each item (Lynn, 1986). Nine items with an I-CVI-score of less than one were excluded from the draft scale using Lynn's (1986) criteria. In addition, based on the experts' recommendations, two of the retained items were revised to simplify the language. This operation yielded 24 items in the final pool (classmates: 8 items; self-confidence: 9 items; errors: 7 items). Table 2 depicts the item pool on the draft scale.

### *2.2.2 Face validity*

The questionnaire's face validity was assessed quantitatively. To evaluate the qualitative face validity, nine college students enrolled in a programming course were interviewed face to face and participants rated the items based on clarity and relevancy. Despite some minor errors, all of the interviewees concurred on the clarity and comprehensibility of all of the items.

### *2.2.3 Translation of the Scale*

The Programming Anxiety Scale items were created and written in Turkish at first. The data were collected utilizing this original scale. The translation of the original scale to English was carried out after the data collection process. A mixed translation strategy utilizing the back-translation method and the committee approach that was distinct from Jones, Lee, Phillips, Zhang & Jaceldo (2001) was used in the translation process. The researchers initially translated each item in the original version into English. Next, three Turkish/English bilingual professors thoroughly inspected each translated item. With the help of the multilingual teacher group, any necessary modifications to problematic items were performed. The bilingual experts agreed on the translated and original versions of the scale.

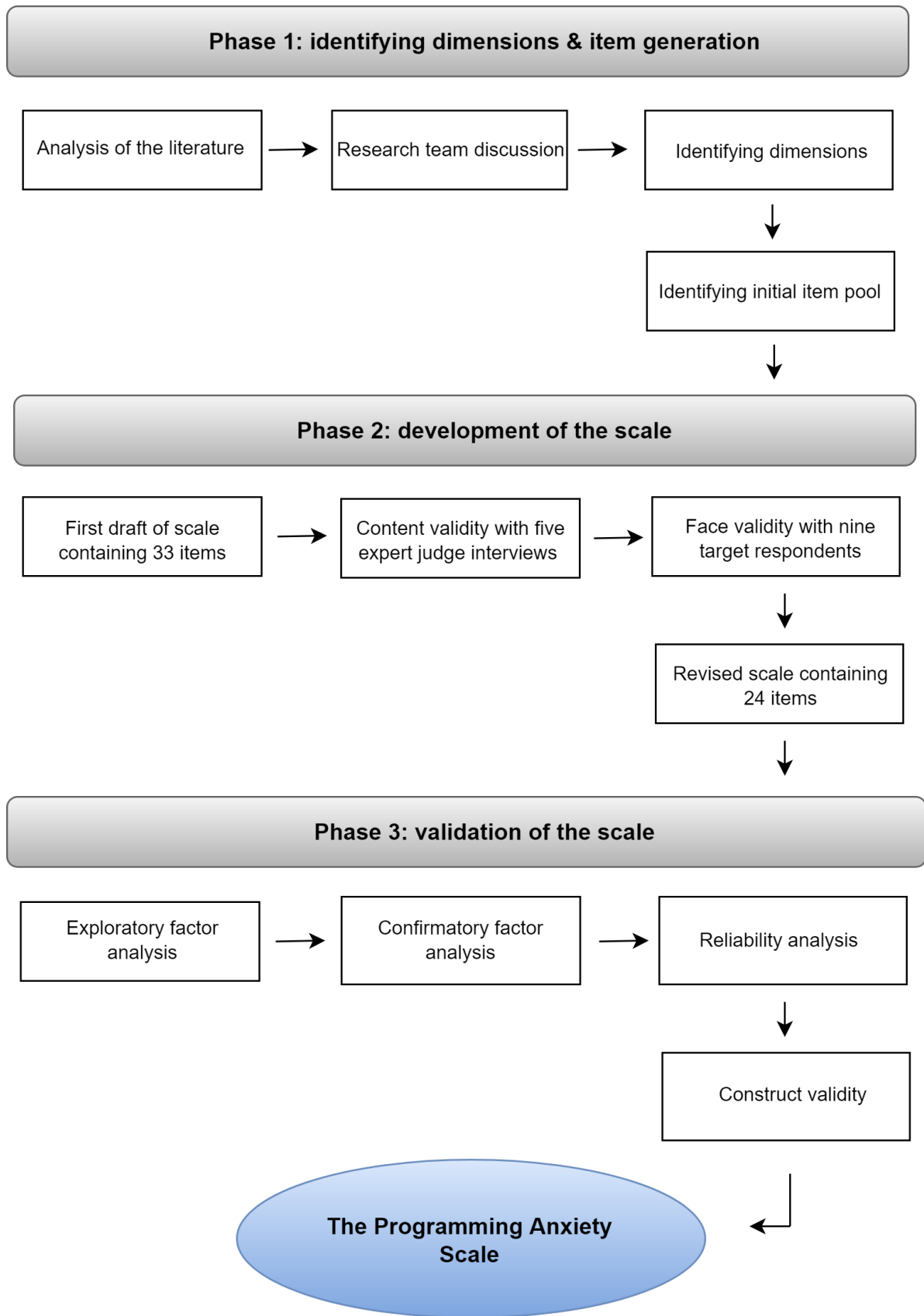


Figure 2. An overview of phases of the PAS development. (Adapted from Zarouali, Boerman & de Vreese, 2021)

Table 2. Item Pool of the Draft Scale

Factor	Items Code	Item
Classmates	Item1	I am concerned about not being able to stay calm like my classmates while coding a program.
	Item2	I feel humiliated if a classmate easily debugs an error on which I worked so hard.
	Item3	Believing that I cannot reach the level of my friends who have taken programming lessons before makes me anxious.
	Item4	The presence of my classmates who have previously taken programming courses makes me nervous.
	Item5	It makes me anxious when many of my classmates can write the code that I cannot write.
	Item6	I get worried if my classmates comprehend a programming topic and I don't.
	Item7	I feel tense when my friends talk about programming topics that I don't understand.
	Item8	I am concerned about not being able to write a program and being ridiculed by my classmates.
Self-confidence	Item9	I think I do not understand programming well.
	Item10	It makes me anxious to feel that I memorize programming topics instead of learning the logic.
	Item11	It makes me anxious to feel that I quickly forget what I have learned in programming lessons.
	Item12	I have doubts about creating the steps (algorithm) necessary for the solution while coding the program.
	Item13	I have concerns about my programming abilities.
	Item14	I feel confused when the program lines become complicated.
	Item15	I don't trust myself in writing programs.
	Item16	I get nervous when we talk about programming.
	Item17	It scares me that there are too many topics to learn in the programming lesson.
Errors	Item18	I get worried when I can not understand error messages.
	Item19	The number of errors in my program makes me worried.
	Item20	I am worried about encountering errors in my programs.
	Item21	I feel worried when my program fails to run.
	Item22	Debugging programs is a major worry for me.
	Item23	I get worried about debugging my software over and over again.
	Item24	It makes me worried to think that my codes will have bugs.

### 2.3 Phase 3: Validation of the scale

#### 2.3.1 Sample

The draft scale was validated primarily in two phases. Since using the same data set for exploratory factor analysis (EFA) and confirmatory factor analysis (CFA) is not generally accepted as the correct method in the literature (Fokkema & Greiff, 2017), data were collected from two distinct sample groups of university students (first and second sample). While the data from the first sample was used to investigate the scale's underlying

factor structure in the EFA phase, the data from the second sample was used in the CFA phase to cross-validate the EFA results. All subjects were recruited using a convenience sampling method.

The first sample comprised 392 university students (29 females, 363 males) taking programming courses at two different universities in Turkey. The female and male participants' mean age was 20.30 and 19.46 years, respectively. The first sample consists of students from the Department of Electronics Technology (82.4), Computer Education and Instructional Technology (CEIT) (10.2%), and Computer Technology (7.4). While 81.89% (n=321) of the students were experienced in a programming, 18.11% (n=71) were novices.

The second sample consisted of 295 college students recruited voluntarily from different programs (32 females, 263 males). The female and male participants' mean age was 21.12 and 19.89 years, respectively. The second sample consists of students from the Department of Electronics Technology (69.04), CEIT (16.01%), and Computer Technology (14.95). While 76.61% (n=226) of the students were experienced, 23.39% (n=69) had no prior programming experience. Data from both samples were collected using Google Forms.

### *2.3.2 Analytical Strategy*

The PAS was validated using Boateng, Neilands, Frongillo, Melgar-Quiñonez, & Young's (2018) scale development recommendations. The number and character of the underlying components in the scale were determined using EFA, which CFA followed to ensure that the data fit the retrieved factor structure. The construct validity was then examined after a reliability analysis. The Kaiser-Meyer-Olkin Measure of Sampling Adequacy (KMO) and Bartlett's test were implemented to assess the adequacy of the study group (Tabachnick & Fidell, 2001). The tests showed that the data was suitable for EFA. In addition, considering Hair, Black, Babin, & Anderson's (2010) suggestions on the cutoff value for factor loadings and commonalities, these values were determined as .50 and .30, respectively. Items that loaded only one factor without any cross-loadings were kept. The research employed SPSS 22 software for EFA. For CFA, Maximum likelihood estimation was adopted to calculate the structure parameters using AMOS 22 software.

The internal consistency coefficient (Cronbach's alpha) was calculated for the full scale and each dimension for reliability analysis. For convergent validity, the factor loading of the indicator, the average variance extracted (AVE), composite reliability (CR), and maximum share variance (MSV) values were calculated (Hair, Hult, Ringle, & Sarstedt, 2014). These calculations were conducted using Gaskin's (2016) AMOS MasterValidity Plugin. Furthermore, convergent validity was assessed through (1) comparison of mean values of factors and total programming anxiety depending on gender and (2) correlation analysis of factors, total programming anxiety, and course grade of students. The Fornell & Larcker criterion and the Heterotrait-Monotrait (HTMT) correlation ratio were utilized to examine discriminant validity (Ab Hamid, Sami & Sidek, 2017). HTMT was calculated using Excel 2016 software. Results of EFA, CFA, reliability, and validity analysis were presented in the *manuscript's results section*.

## **3. Results**

### *3.1 Exploratory Factor Analysis*

The data's appropriateness for factor analysis was assessed before doing EFA. For this reason, Mahalanobis distance values for probable multivariate outliers were determined. Thirty-nine instances were eliminated from the analysis because their Mahalanobis values were above the necessary chi-square value of 54.05 (df = 26, alpha=.001) (Pallant, 2007). In addition, KMO was used to determine the adequacy of the sample size, and Bartlett's Test of Sphericity was used to determine whether the datum was suitable for factor analysis. The KMO sampling adequacy metric was .951, higher than the acceptable value of .60. Bartlett's Test of Sphericity was also statistically significant ( $\chi^2=3790.593$ ,  $p=.000$ ), demonstrating that the data was considerably factorable (Pallant, 2007).

In the first stage of EFA, principal axis factoring was employed for 24 items with direct oblimin rotation. The direct oblimin rotation approach was chosen because of the associated factors (Costello & Osborne, 2005; Gorsuch, 1983). After the EFA process, the correlation matrix was investigated for multicollinearity issues. Correlations in the .80's or .90's (Field, 2018) were examined, and six items (Item18, Item19, Item20, Item21, Item23, and Item24) in the "errors" factor with a correlation coefficient greater than .8 were excluded from the scale. Next, EFA was conducted again for the remaining 18 items. Four items (Item2 and Item6 in classmates and Item13 and Item16 in self-confidence) with high factor loadings in multiple factors were excluded (Burns & Machin, 2009; Howard, 2016). EFA was executed with the remaining 14 items. Based on the cutoffs for the eigenvalues and inspection of the scree plot, a two-factor model was identified that explained 69.4% of the



variance in programming anxiety. The two-factor model was confirmed by a parallel analysis with 5000 randomly generated data matrices through Parallel Analysis Web Application (Patil, Surendra, Sanjay, & Donovan, 2017). These factors were labeled: (1) Classmates and (2) Self-Confidence. Table 3 includes the complete list of factor loadings. Internal consistency reliabilities (i.e., Cronbach's alpha coefficient) for the full scale and the subscales were .95, .90, and .94, respectively.

Table 3. Factor Loadings of the PAS

Items Code	Item	Classmates	Self-Confidence
Item1	I am concerned about not being able to stay calm like my classmates while coding a program.	.633	
Item3	Believing that I cannot reach the level of my friends who have taken programming lessons before makes me anxious.	.806	
Item4	The presence of my classmates who have previously taken programming courses makes me nervous.	.918	
Item5	It makes me anxious when many of my classmates can write the code that I cannot write.	.643	
Item7	I feel tense when my friends talk about programming topics that I don't understand.	.634	
Item8	I am concerned about not being able to write a program and being ridiculed by my classmates.	.877	
Item9	I think I do not understand programming well.		.818
Item10	It makes me anxious to feel that I memorize programming topics instead of learning the logic.		.726
Item11	It makes me anxious to feel that I quickly forget what I have learned in programming lessons.		.851
Item12	I have doubts about creating the steps (algorithm) necessary for the solution while coding the program.		.844
Item14	I feel confused when the program lines become complicated.		.909
Item15	I don't trust myself in writing programs.		.780
Item17	It scares me that there are too many topics to learn in the programming lesson.		.852
Item22	Debugging programs is a major worry for me.		.822

### 3.2 Confirmatory Factor Analysis

The 14-item Programming Anxiety Scale's two-factor model was subjected to CFA using second sample data. Before the analysis, data were screened for missing values and outliers. With this respect, 13 respondents were detected unengaged in the scale evidence by getting the same response for every item. Thus, 14 cases were deleted from the sample. Next, CFA was conducted with 282 samples using maximum likelihood estimation. After CFA, three items (i.e., Item1, Item9, and Item11) had standardized parameter estimates smaller than the recommended value of .50 (Hair et al., 2010).

Furthermore, AVE values for factors below .50 were calculated, indicating the absence of convergent validity. After removing these three items, 11 item structure of the PAS was re-subjected to CFA. The diagram regarding the factor structure of programming anxiety with new item codes (See Appendix) and the parameter estimates was given in Figure 2.

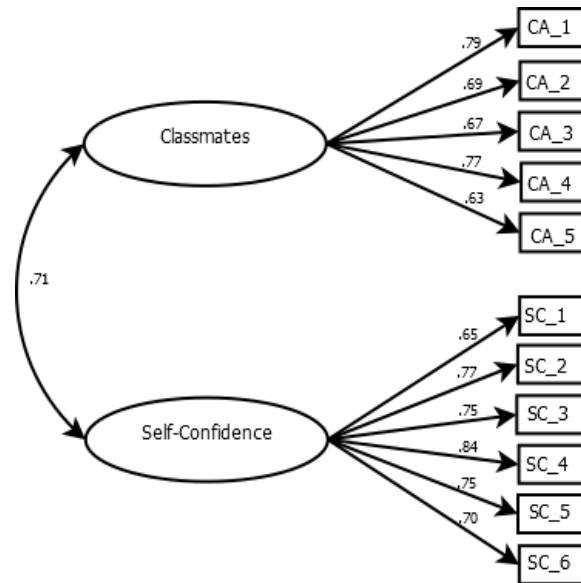


Figure 2. Standardized Coefficients for the Two-Factor Model of PAS

According to Hu and Bentler (1999), researchers employ numerous goodness-of-fit metrics to analyze a model. In this study, the Chi-square goodness of fit test, Root Mean Square of Error of Approximation (RMSEA), Standardized Root Mean Square Residuals (SRMR), Goodness of Fit Index (GFI), Normative Fit Index (NFI), and Comparative Fit Index (CFI) were employed to assess model fit (Brown, 2006; Browne & Cudeck, 1993; Hair et al., 2010; Kline, 1998). Table 4 presents the fit statistics for the confirmative factor analysis.

Table 4. Values of the Goodness-of-Fit Test for Programming Anxiety

$X^2$	$X^2/df$	p-value	RMSEA	SRMR	GFI	NFI	CFI
114.226*	2.79	.000	.071	0.032	.93	.93	.95

\*  $p < 0.01$

When Table 4 was analyzed, the chi-square value ( $X^2 = 114.226$ ,  $X^2/df = 2.79$ ,  $p = .000$ ) was found to be significant. RMSEA value of .071 indicate good adaptation (Brown, 2006; Browne & Cudeck, 1993). GFI, CFI, and NFI values greater than .90 indicate a good fit (Hair et al., 2010; Kline, 1998). The CFA results indicate that the structural model is a good fit. As shown in Fig. 2., each item loaded significantly on its particular dimension and was relatively large (.50 and above).

### 3.3 Assessment of Reliability and Validity

To assess convergent validity, standardized factor loadings, CR, and AVE were calculated (Hair et al., 2010). Cronbach alpha values for factors and the whole scale were calculated for reliability analysis (Taber, 2018). The condition that factor loading values greater than .5, Cronbach alpha values greater than .7, and AVE and CR values greater than .5 and .7 were taken into account (Taber, 2018; Hair et al., 2010). Table 5 illustrates that all constructs are reliable since they fulfill the above criteria. Furthermore, each construct's Cronbach's alpha exceeds the recommended value. The Cronbach's alpha value of the whole scale was calculated as .901, which fulfilled the reliability criterion.

Table 5. Evaluation of the Measurement Model

Factors	CR	AVE	MSV	Cronbach Alpha
Classmates	.835	.506	.498	.843
Self-Confidence	.881	.554	.498	.882

Furthermore, convergent validity was assessed through (1) comparison of mean values of factors and total programming anxiety depending on gender (Table 6) and (2) correlation analysis of factors, total programming anxiety, and course grade of students (Table 7). As shown in Table 6, female students had more programming anxiety than male respondents. Gender differences in programming anxiety were also investigated using a t-test. The test results showed that while there was no significant difference between male and female students in self-confidence factor ( $t = .767, p > .05$ ), there was a significant difference in classmates factor ( $t = 4.058, p = .000$ ) and total programming anxiety ( $t = 2.518, p = .012$ ). Correlation analysis results (Table 7) revealed that classmates ( $r = -.194$ ) and self-confidence ( $r = -.315$ ) factors, as well as total programming anxiety ( $r = -.284$ ), were negatively associated with course grade,  $p < .01$ . All of these correlations were found as weak (Schober, Boer & Schwarte, 2018; Senthilnathan, 2019).

Table 6. Gender differences in programming anxiety

Gender	Factors					Total Programming Anxiety	
	N	Classmates		Self-Confidence		M	SD
Female	30	M	SD	M	SD	M	SD
Male	252	16.00	4.69	16.87	4.75	32.87	8.45
		11.96	5.19	15.96	6.27	27.91	10.34

Table 7. Correlation analysis results

	Classmates	Self-Confidence	Total Programming Anxiety	Course Grade
Classmates	—			
Self-Confidence	.688**	—		
Total Programming Anxiety	.898**	.938**	—	
Course Grade	-.194**	-.315**	-.284**	—

\*\* . Correlation is significant at the 0.01 level (2-tailed).

The Fornell & Larcker and the HTMT criteria were used to test the scale's discriminant validity. Table 8 summarizes the results of the Fornell & Larcker criteria. In Table 8, each AVE's square root was given on the diagonal, and the correlation coefficients (off-diagonal) for each construct were displayed in the corresponding rows and columns. Fornell and Larcker (1981) state that the AVE values' square root should be higher than the correlations between the components included in the analysis. As shown in Table 8, this condition was satisfied, and the model met the Fornell & Larcker criterion for discriminant validity. In addition to the Fornell & Larcker criterion, discriminant validity was assessed through the HTMT coefficient. The HTMT coefficient was calculated as .705 in this model. According to Henseler, Ringle, and Sarstedt (2015), the HTMT coefficient should be less than .90 if the components to be evaluated are hypothetically close to one other. The HTMT coefficient was found to be below the threshold levels.

Table 8. The square root of the average variance extracted (AVE) and correlations matrix

Factors	Factors	
	Classmates	Self-Confidence
Classmates	<b>.711</b>	
Self-Confidence	.705	<b>.745</b>

#### **4. Discussion and Conclusion**

In computing education research, accurate measurement is critical (Scott, 2015). On the other hand, few measurement tools are available to computer education researchers (Scott & Ghinea, 2014). The current research fills a gap in the literature by developing and validating a measurement tool to assess the computer programming anxiety of university students. The development and validation of the programming anxiety scale in the current study were carried out in harmony with the scale studies recently published in several fields (Nasir, Adil, & Kumar, 2021; Rosario-Hernández, Rovira-Millán, & Blanco-Rovira, 2022; Sun et al., 2022; Zarouali, Boerman, & de Vreese, 2021). EFA, DFA, reliability, and validity analysis resulted in a scale including 11 items, five items for classmates, and six for self-confidence. The minimum obtainable score from the Programming Anxiety Scale is 11, while the maximum score is 55. As the score obtained from the scale increases, programming anxiety also increases. Choo and Cheung's (1991) Computer Programming Anxiety Scale served as a guide to develop the present scale.

In the current study, no factor related to errors was found, while there was a factor for error anxiety in the study of Choo & Cheung (1991) and Demir (2021), in which Choo & Cheung's (1991) scale was adapted into Turkish. Although seven items were included in the draft scale for this factor, six of these items showed high multicollinearity. They were removed from the draft scale due to the exploratory factor analysis, and one item was kept. Although the inclusion of an item about error anxiety shows that debugging is a source of anxiety for students (Dasuki & Quaye, 2016; Nolan & Bergin, 2016), it is worth examining why it is not included as a factor. Not having an "errors" factor may indicate that encountering errors is a source of anxiety regardless of the number of errors and the time spent for debugging, which was utilized as parameters in Choo & Cheung (1991) and Demir (2021). In other words, encountering errors in programs may exist as a single source of anxiety, regardless of the frequency of encountering errors, the number of errors, or the time it takes to debug. The fact that Demir's (2021) study does not contain any information about CFA makes it impossible to make inferences about whether the three-factor model shows a good fit in the Turkish version of the scale and make comparisons about the error factor.

Within the scope of the current study, the only item related to error anxiety was included in the self-confidence factor. One of the reasons for this result may be the differences in perception of debugging as a process in programming activities between the novices and the relatively more experienced individuals. From this point of view, while debugging may be perceived as an independent process for novice programmers, debugging may be perceived as an integral process of programming and an element of self-efficacy perception. Considering a high degree of relationship between self-confidence and self-efficacy perception (Blanco et al., 2020; Malureanu, Panisoara & Lazar, 2021; Tsai, 2019), it is not surprising that an item in the error factor is included in the self-confidence factor. It is evident that enhanced debugging skills develop a programmer's confidence, and fear of making mistakes may be related to programming skills (Ahmadzadeh, Elliman, & Higgins, 2005; Connolly et al., 2009; Nolan & Bergin, 2016). From this point of view, the experience of the participant group of Choo & Cheung's (1991) study on programming was not explained in detail. The participant group was specified only as of grade 12 level. However, participants in both the EFA and CFA stages of the current study were relatively more experienced with programming than the participants in Choo & Cheung's (1991) study. They have developed at least one project and were familiar with the debugging process. This result may indicate that perceptions of debugging are related to programming experience.

Another reason may be the attitudes of the participants towards programming. Choo & Cheung's (1991) participant group consisted of grade 12 junior high school students. In contrast, the current study participants comprised university students who perceived programming as a profession. The students' awareness that the programming profession will include debugging may have caused them to perceive debugging as a personal competence and an aspect of their career. The last reason for this result may be the software development environments (Integrated Development Environments-IDEs) and the resources and materials that can be facilitated for debugging. Scratch was used in Demir (2021) as the program development environment. On the contrary, Visual Studio was used in the current study. The nature of the errors encountered in Scratch and Visual Studio show differences. The IDE used by Choo & Cheung (1991) was not specified. However, the number of resources found on debugging in the 1990s and today's resources are very different. Today, the internet is used for interpretations of error messages, and previous experiences of other people are utilized for solutions. Nowadays, even IDEs that translate error messages into users' native language exist. Therefore, the characteristics of IDEs and the resources may alter the perception of debugging anxiety.

The validity of the developed scale was tested by comparing the results of studies examining the relationship between programming anxiety and theoretically related variables in the literature. In the current study, female students had more programming anxiety than male respondents. The t-test result also revealed that while there was no significant difference between male and female students in self-confidence, there was a substantial difference in classmates factor and total programming anxiety. This result is consistent with Olipas and Luciano's (2020) study. Similarly, consistent with the former findings ((Connolly et al., 2007; Figueroa & Amoloza, 2015; Kinnunen & Malmi, 2006; Nolan et al., 2019; Owolabi et al., 2014; Scott, 2015), factors of the PAS and total programming anxiety was correlated with course grades of students. These results indicate that the developed scale is valid and reliable.

Another issue is related to the fact that this scale was developed specifically for programming anxiety. Since this scale was designed specifically to measure programming anxiety, it differs from scales adapted to programming anxiety, such as computer anxiety, computer attitude, and IT anxiety (Chang, 2005; Olipas & Luciano, 2020; Owolabi et al., 2014). There were insufficient instruments to assess programming anxiety in the literature, and the current study offered a psychometrically reliable scale. With the help of the present scale, programming anxiety levels in students can be measured, methods and techniques that can reduce students' anxiety can be developed, and special attention can be paid to students with high programming anxiety. In addition, situations that increase programming anxiety in students can be investigated. The PAS developed in the current study is recommended for study groups that have somewhat experience in creating, coding, and debugging programming projects. Future research may concentrate on a different group of college students from various cultures and countries.

### **Acknowledgments**

This study was conducted in the framework of Osman Gazi YILDIRIM's Ph.D. dissertation at Marmara University/TURKEY, supervised by Professor Nesrin OZDENER DONMEZ.

### **References**

- Ab Hamid, M. R., Sami, W., & Sidek, M. M. (2017, September). Discriminant validity assessment: Use of Fornell & Larcker criterion versus HTMT criterion. In *Journal of Physics: Conference Series* (Vol. 890, No. 1, p. 012163). IOP Publishing.
- Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An analysis of patterns of debugging among novice computer science students. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* (pp. 84-88).
- Al-Makhzoomy, A. K. (2018). *Effect of Game Development-Based Learning on the Ability of Information Technology Undergraduates to Learn Computer and Object-Oriented Programming* (Doctoral dissertation, Wayne State University).
- Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2), 32-36.
- Bergin, S., & Reilly, R. (2005). Programming: factors that influence success. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education* (pp. 411-415).
- Berland, M., & Lee, V. R. (2011). Collaborative strategic board games as a site for distributed computational thinking. *International Journal of Game-Based Learning (IJGBL)*, 1(2), 65-81.
- Blanco, Q. A., Carlota, M. L., Nasibog, A. J., Rodriguez, B., Saldaña, X. V., Vasquez, E. C., & Gagani, F. (2020). Probing on the relationship between students' self-confidence and self-efficacy while engaging in online learning amidst COVID-19. *Journal La Edusci*, 1(4), 16-25.
- Boateng, G. O., Neilands, T. B., Frongillo, E. A., Melgar-Quinonez, H. R., & Young, S. L. (2018). Best practices for developing and validating scales for health, social, and behavioral research: a primer. *Frontiers in public health*, 6, 149.
- Bosch, N., D'Mello, S., & Mills, C. (2013, July). What emotions do novices experience during their first computer programming learning session?. In *International Conference on Artificial Intelligence in Education* (pp. 11-20). Springer, Berlin, Heidelberg.

- Brown, T. A. (2006). *Confirmatory factor analysis for applied research*. New York: Guilford Publications.
- Browne, M. W., & Cudeck, R. (1992). Alternative ways of assessing model fit. *Sociological methods & research*, 21(2), 230-258.
- Bunderson, E. D., & Christensen, M. E. (1995). An analysis of retention problems for female students in university computer science programs. *Journal of Research on Computing in Education*, 28(1), 1-18.
- Burns, R. A., & Machin, M. A. (2009). Investigating the structural validity of Ryff's psychological well-being scales across two samples. *Social indicators research*, 93(2), 359-375.
- Butcher, D. F., & Muth, W. A. (1985). Predicting performance in an introductory computer science course. *Communications of the ACM*, 28(3), 263-268.
- Byrne, P., & Lyons, G. (2001, June). The effect of student attributes on success in programming. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education* (pp. 49-52).
- Campbell, S. L. (2018). *An evaluation of an automated, interactive learning method for a database query language*. The University of Iowa.
- Carpenter, S. (2018). Ten steps in scale development and reporting: A guide for researchers. *Communication Methods and Measures*, 12(1), 25-44.
- Chang, S. E. (2005). Computer anxiety and perception of task complexity in learning programming-related skills. *Computers in Human Behavior*, 21(5), 713-728.
- Choo, M. L., & Cheung, K. C. (1991). On meaningful measurement: Junior college pupils' anxiety towards computer programming. *Journal of Educational Technology Systems*, 19(4), 327-343.
- Clark, L. A., & Watson, D. (1995). Constructing validity: Basic issues in objective scale development. *Psychological Assessment*, 7(3), 309-319
- Connolly, C., Murphy, E., & Moore, S. (2007). Second chance learners, supporting adults learning computer programming. In *international conference on engineering education-ICEE*.
- Connolly, C., Murphy, E., & Moore, S. (2009). Programming anxiety amongst computing students—A key in the retention debate?. *IEEE Transactions on Education*, 52(1), 52-56.
- Costello, A. B., & Osborne, J. (2005). Best practices in exploratory factor analysis: Four recommendations for getting the most from your analysis. *Practical assessment, research, and evaluation*, 10(1), 7.
- Dasuki, S., & Quaye, A. (2016). Undergraduate students' failure in programming courses in institutions of higher education in developing countries: A Nigerian perspective. *The Electronic Journal of Information Systems in Developing Countries*, 76(1), 1-18.
- Demir, F. (2021). The effect of different usage of the educational programming language in programming education on the programming anxiety and achievement. *Education and Information Technologies*, 1-24.
- Demirer, V. & Sak, N. (2016). Dünyada ve Türkiye'de programlama eğitimi ve yeni yaklaşımlar. *Eğitimde Kuram ve Uygulama*, 12(3), 521-546.
- Dijkstra, E. W. (1989). On the cruelty of really teaching computing science. *Communications of the ACM*, 32(12), 1398-1404.
- Falkner, K., Falkner, N. J., & Vivian, R. (2013, March). Collaborative Learning and Anxiety: A phenomenographic study of collaborative learning activities. In *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 227-232).
- Field, A. (2018). *Discovering Statistics Using IBM SPSS Statistics*. Sage
- Figuroa Jr, R. B., & Amoloza, E. M. (2015). Addressing Programming Anxiety among Non-Computer Science Distance Learners: A UPOU Case Study. *International Journal*, 9(1), 56-67.
- Fokkema, M., & Greiff, S. (2017). How performing PCA and CFA on the same data equals trouble. *European Journal of Psychological Assessment*.

- Fornell, C., & Larcker, D. F. (1981). Evaluating structural equation models with unobservable variables and measurement error. *Journal of marketing research, 18*(1), 39-50.
- Gaskin, J. (2016), "MasterValidity", Gaskination's Statistics. <http://statwiki.gaskination.com>
- Gibbs, D. C. (2000). The effect of a constructivist learning environment for field-dependent/independent students on achievement in introductory computer programming. *ACM SIGCSE Bulletin, 32*(1), 207-211.
- Gorsuch, R. L. (1983). *Factor analysis (2nd Edition)*. Hillsdale, NJ: Erlbaum.
- Hair, J., Black, W., Babin, B., & Anderson, R. (2010). *Multivariate data analysis (7th ed.)*. Upper Saddle River, NJ: Prentice-Hall.
- Hair, J. F., Hult, G. T. M., Ringle, C., & Sarstedt, M. (2014). *A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM)*.
- Hattie, J. (2008). *Visible learning: A synthesis of over 800 meta-analyses relating to achievement*. Routledge.
- Henseler, J., Ringle, C. M., & Sarstedt, M. (2015). A new criterion for assessing discriminant validity in variance-based structural equation modeling. *Journal of the academy of marketing science, 43*(1), 115-135.
- Hostetler, T. R. (1983). Predicting student success in an introductory programming course. *ACM SIGCSE Bulletin, 15*(3), 40-43.
- Howard, M. C. (2016). A review of exploratory factor analysis decisions and overview of current practices: What we are doing and how can we improve? *International Journal of Human-Computer Interaction, 32*(1), 51-62.
- Hsu, W. C., & Gainsburg, J. (2021). Hybrid and Non-Hybrid Block-Based Programming Languages in an Introductory College Computer-Science Course. *Journal of Educational Computing Research, 59*(5), 817-843.
- Hu, L. T., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural equation modeling: a multidisciplinary journal, 6*(1), 1-55.
- Huggard, M. (2004). Programming trauma: can it be avoided. *Proceedings of the BCS Grand Challenges in Computing: Education, 50-51*.
- Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* (Vol. 4, No. 2002, pp. 53-58).
- Jiang, Y., Zhao, Z., Wang, L., & Hu, S. (2020, August). Research on the Influence of Technology-Enhanced Interactive Strategies on Programming Learning. In *2020 15th International Conference on Computer Science & Education (ICCSE)* (pp. 693-697). IEEE.
- Jones, P. S., Lee, J. W., Phillips, L. R., Zhang, X. E., & Jaceldo, K. B. (2001). An adaptation of Brislin's translation model for cross-cultural research. *Nursing research, 50*(5), 300-304.
- Kert, S. B. & Uğraş, T. (2009). Programlama eğitiminde sadelik ve eğlence: Scratch örneği. In *The First International Congress of Educational Research, Çanakkale, Turkey*.
- Kinnunen, P., & Malmi, L. (2006, September). Why students drop out CS1 course?. In *Proceedings of the second international workshop on Computing education research* (pp. 97-108).
- Kinnunen, P., & Simon, B. (2012). My program is ok—am I? Computing freshmen's experiences of doing programming assignments. *Computer Science Education, 22*(1), 1-28.
- Kline, R. B. (1998). *Principles and practice of structural equation modeling*. New York: Guildford
- Luxton-Reilly, A., Ajanovski, V. V., Fouh, E., Gonsalvez, C., Leinonen, J., Parkinson, J., ... & Thota, N. (2019). Pass rates in introductory programming and in other stem disciplines. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 53-71).
- Lynn, M. R. (1986). Determination and quantification of content validity. *Nursing research*.
- Maguire, P., Maguire, R., & Kelly, R. (2017). Using automatic machine assessment to teach computer programming. *Computer Science Education, 27*(3-4), 197-214.

- Malureanu, A., Panisoara, G., & Lazar, I. (2021). The relationship between self-confidence, self-efficacy, grit, usefulness, and ease of use of elearning platforms in corporate training during the COVID-19 pandemic. *Sustainability, 13*(12), 6633.
- Nasir, M., Adil, M., & Kumar, M. (2021). Phobic COVID-19 disorder scale: Development, dimensionality, and item-structure test. *International Journal of Mental Health and Addiction, 1*-13.
- Nolan, K., & Bergin, S. (2016, November). The role of anxiety when learning to program: a systematic review of the literature. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (pp. 61-70).
- Nolan, K., Bergin, S., & Mooney, A. (2019, September). An Insight Into the Relationship Between Confidence, Self-efficacy, Anxiety and Physiological Responses in a CS1 Exam-like Scenario. In *Proceedings of the 1st UK & Ireland Computing Education Research Conference* (pp. 1-7).
- Nunnally, J. C. (1994). *Psychometric theory* 3E. Tata McGraw-Hill Education.
- Olipas, C. N. P., & Luciano, R. G. (2020). Understanding The Impact Of Using Countdown Timer On The Academic Motivation And Computer Programming Anxiety Of IT Students: The Case Of A State University In The Philippines. *International Journal of Scientific and Technology Research, 9*.
- Olipas, C. N. P., Leona, R. F., Villegas, A. C. A., Cunanan Jr, A. I., & Javate, C. L. P. (2021). The Academic Performance and the Computer Programming Anxiety of BSIT Students: A Basis for Instructional Strategy Improvement.
- Orehovacki, T., Radosevic, D., & Konecki, M. (2012, June). Acceptance of Verificator by information science students. In *Proceedings of the ITI 2012 34th International Conference on Information Technology Interfaces* (pp. 223-230). IEEE.
- Owolabi, J., Olanipekun, P., & Iwerima, J. (2014). Mathematics ability and anxiety, computer and programming anxieties, age and gender as determinants of achievement in basic programming. *GSTF Journal on Computing (JoC), 3*(4), 109.
- Özmen, B., & Altun, A. (2014). Undergraduate Students' Experiences in Programming: Difficulties and Obstacles. *Turkish Online Journal of Qualitative Inquiry, 5*(3).
- Pajares, M. F. (1992). Teachers' beliefs and educational research: Cleaning up a messy construct. *Review of educational research, 62*(3), 307-332.
- Pallant, J. (2007). *SPSS survival manual: A step by step guide to data analysis using SPSS version 15* (3rd ed.). New York: Open University Press.
- Patil Vivek H, Surendra N. Singh, Sanjay Mishra, and D. Todd Donovan (2017). *Parallel Analysis Engine to Aid in Determining Number of Factors to Retain using R [Computer software]*, available from <https://analytics.gonzaga.edu/parallelenigne/>.
- Polit, D. F., Beck, C. T., & Owen, S. V. (2007). Is the CVI an acceptable indicator of content validity? Appraisal and recommendations. *Research in nursing & health, 30*(4), 459-467.
- Ramalingam, V., & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research, 19*(4), 367-381.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education, 13*(2), 137-172.
- Rogerson, C., & Scott, E. (2010). The fear factor: How it affects students learning to program in a tertiary environment. *Journal of Information Technology Education: Research, 9*(1), 147-171.
- Rosario-Hernández, E., Rovira-Millán, L. V., & Blanco-Rovira, R. A. (2022). Development and Validation of the Job Satisfaction Brief Scale. *Revista Caribeña de Psicología, e6191-e6191*.
- Rountree, N., Rountree, J., & Robins, A. (2002). Predictors of success and failure in a CS1 course. *ACM SIGCSE Bulletin, 34*(4), 121-124.



- Santos, S. C., Tedesco, P. A., Borba, M., & Brito, M. (2020). Innovative approaches in teaching programming: A systematic literature review. In *Proceedings of the 12th International Conference on Computer Supported Education* (Vol. 1, pp. 205-214).
- Schober, P., Boer, C., & Schwarte, L. A. (2018). Correlation coefficients: appropriate use and interpretation. *Anesthesia & Analgesia*, *126*(5), 1763-1768.
- Scott, M. (2015). *Self-Beliefs in the Introductory Programming Lab and Games-based Fantasy Role-Play* (Doctoral dissertation, Brunel University).
- Scott, M. J., & Ghinea, G. (2014, July). Measuring enrichment: the assembly and validation of an instrument to assess student self-beliefs in CS1. In *Proceedings of the tenth annual conference on International computing education research* (pp. 123-130).
- Senthilnathan, S. (2019). Usefulness of correlation analysis. *Available at SSRN 3416918*.
- Sinožić, S., & Orehovački, T. (2018, May). Using analytic hierarchy process to select the most appropriate tool for learning programming. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 0524-0529). IEEE.
- Sun, J., Jiang, X., Gao, Y., He, C., Wang, M., Wang, X., ... & Zhang, L. (2022). Subhealth Risk Perception Scale: Development and Validation of a New Measure. *Computational and Mathematical Methods in Medicine*, 2022.
- Tabachnick, B. G., & Fidell, L. S. (2001). *Using multivariate statistics (4th ed.)*. Needham Heights, MA: Allyn & Bacon
- Taber, K. S. (2018). The use of Cronbach's alpha when developing and reporting research instruments in science education. *Research in science education*, *48*(6), 1273-1296.
- Tan, P. H., Ting, C. Y., & Ling, S. W. (2009, November). Learning difficulties in programming courses: undergraduates' perspective and perception. In *2009 International Conference on Computer Technology and Development* (Vol. 1, pp. 42-46). IEEE.
- Tsai, C. Y. (2019). Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Computers in Human Behavior*, *95*, 224-232.
- Watson, C., & Li, F. W. (2014, June). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 39-44).
- Wiedenbeck, S., Labelle, D., & Kain, V. N. (2004). Factors affecting course outcomes in introductory programming. In *PPIG* (p. 11).
- Wilfong, J. D. (2006). Computer anxiety and anger: The impact of computer use, computer experience, and self-efficacy beliefs. *Computers in human behavior*, *22*(6), 1001-1011.
- Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *ACM SIGCSE Bulletin*, *33*(1), 184-188.
- Zarouali, B., Boerman, S. C., & de Vreese, C. H. (2021). Is this recommended by an algorithm? The development and validation of the algorithmic media content awareness scale (AMCA-scale). *Telematics and Informatics*, *62*, 101607.

## Appendix

### The Programming Anxiety Scale in its Original and Translated Form

Original Items (in Turkish)	Translated Items (in English)
CA_1 Daha önce programlama dersi alan arkadaşlarımla seviyesine yetişemeyeceğime inanmak beni kaygılandırır.	Believing that I cannot reach the level of my friends who have taken programming lessons before makes me anxious.
CA_2 Daha önce programlama dersi alan sınıf arkadaşlarımla varlığı beni tedirgin eder.	The presence of my classmates who have previously taken programming courses makes me nervous.
CA_3 Benim yazmadığım bir kodu çoğu sınıf arkadaşım yazabilmesi beni kaygılandırır.	It makes me anxious when many of my classmates can write the code that I cannot write.
CA_4 Arkadaşlarımla benim anlamadığım programlama konularında konuştuğunda gergin hissederim.	I feel tense when my friends talk about programming topics that I don't understand.
CA_5 Program yazmayıp sınıf arkadaşlarımla önünde gülünç duruma düşeceğimden endişelenirim.	I am concerned about not being able to write a program and being ridiculed by my classmates.
SC_1 Programlamayı iyi anlayamadığımı düşünürüm.	I think I do not understand programming well.
SC_2 Program yazarken çözüm için gerekli olan basamakları (algoritmayı) doğru oluşturabileceğim konusunda şüphelerim var.	I have doubts about creating the steps (algorithm) necessary for the solution while coding the program.
SC_3 Program satırları karmaşık olmaya başladığında aklımın karıştığını hissederim.	I feel confused when the program lines become complicated.
SC_4 Program yazma konusunda kendime güvenmem.	I don't trust myself in writing programs.
SC_5 Programlama dersinde öğrenilecek çok fazla konunun olması beni korkutur.	It scares me that there are too many topics to learn in the programming lesson.
SC_6 Programlarımda hatalarla karşılaşmak benim için büyük bir endişe kaynağıdır.	Debugging programs is a major worry for me.

# Understanding Problem-Solving Processes of Preschool Children in CS- Unplugged Activities

Ünal ÇAKIROĞLU<sup>1</sup>  
Şüheda MUMCU<sup>1</sup>  
Melek ATABAY<sup>1</sup>  
Merve AYDIN<sup>1</sup>

<sup>1</sup>Trabzon University

DOI: 10.21585/ijcses.v5i3.133

## Abstract

This study aims to explore the influences of the CS-unplugged activities in developing problem-solving skills of preschool children. The participants were 11 children (4-5 aged) enrolled in a public preschool and Code.org activities were used as an instructional package. Activity evaluation form and interviews were used to understand children's problem-solving processes. In order to determine the problem-solving performances, the tasks were divided into the meaningful sub-tasks with regard to problem steps of Nance' problem solving model. The results indicated that CS-unplugged activities positively influenced students' understanding and planning performances more than doing and evaluation skills. Preschool children developmental characteristics and the nature of the problems somewhat hampered the development of their performances in doing and evaluation steps. It is hoped that the study may provide insights for the efforts on enhancing preschool children's problem-solving processes.

**Key words:** CS unplugged activities; problem solving; preschool children

## 1. Introduction

Over the past decade, computational thinking (CT) has become a very hot topic in educational research and practice. After Wing's (2006) declaration, a common idea for CT definition entails at least thinking in a way that formulating problems and their solutions are represented in a form that can be effectively carried out with an information-processing agent (Wing, 2011). It leded researchers to study on supporting young children to acquire thinking skills that are transferable to problem solving in computing related subjects (Bransford et al., 2000, Brackmann et al., 2017).

Wing (2006) argued that CT is a fundamental skill for everyone, not just for computer scientists. Researchers also highlighted that CT is an important skill that should be taught to the next generation (Barr et al., 2011; Brown et al., 2013; Grgurina et al., 2014; Grover & Pea, 2013; Hodhod et al., 2016; Kafai & Burke, 2013; Voogt et al., 2015; Wing, 2006). Thus, many countries have updated their computer science (CS) curriculum to teach children starting from young ages (Bargury et al., 2012; Bers et al., 2014; Grgurina et al., 2014; Grout & Houlden, 2014; Kalelioglu et al., 2014; Lee, Martin, & Apone, 2014; Repenning, Webb, & Ioannidou, 2010). In an attempt to increase interest in CS, much effort has gone into developing some preliminary learning materials, activities, methods or tools for teaching CT for young children. In these studies, programming (Dann, Cooper, & Pausch, 2009; Resnick et al., 2009), educational robotics and CS-unplugged activities (Bell, Witten, & Fellows, 2005, Wohl et al., 2015) are frequently used considering the educational levels.

Since it is difficult to teach CT to young children via programming or robotics, CS-unplugged activities are suggested as an introduction strategy (Battal, Adanır & Gülbahar, 2021). Additionally, not all children are lucky enough to access to powerful tools and toys and CS-unplugged activities yield equal opportunity across all learners as individuals that do not require so many technological tools. Unplugged activities also have potentials to learn the concepts without the need for technological devices or computers (Kalelioğlu & Keskinçilic, 2017).

Previous research has demonstrated beneficial effects of CS-unplugged initiative as a way of teaching CT in early ages. For example, Del Olmo-Munoz et al. (2020) found that in the early stages of primary education, it is more suitable to perform CT through unplugged programming activities before plugged-in activities. Bell and Vahrenhold (2018) found through a literature review that unplugged programming activities can help students and teachers stimulate motivation to explore CS in a meaningful and attractive way and can also help students to carry out subsequent 'plugged-in' learning. Wohl et al. (2015) compared Scratch, Cubelets, and unplugged activities in teaching CS to 5-7 aged children and found that unplugged activities are more powerful in teaching concept of algorithm than others. On the other hand, Caelien and Yadav (2020) pointed out that unplugged programming activities can support students' participation in plugged-in programming activities in the future. Some other researchers also emphasized the role of CS-unplugged activities as a priming step to help students understand algorithmic steps before they write code (Gardeli & Vosinakis, 2017; Uchida et al., 2015). Following conclusions from the previous studies we aim at gaining an insight into the relationships between the nature of CS-unplugged activities and the problem-solving process of preschool students.

### *1.1. CS-unplugged Activities for Developing Problem Solving Skills*

According to Wing (2008) CS-unplugged activities are various kinds of problems that do not directly involve coding tasks. CS-unplugged is defined as a widely used collection of activities and ideas to engage a variety of audiences with great ideas from CS, without having to learn programming or even use a digital device (Bell & Vahrenhold, 2018, p. 497). Research have shown that CS-unplugged activities contribute to the acquisition of basic CS concepts (Hermans & Aivaloglou 2017; Wohl et al. 2015; Taub et al., 2009), support improvement of CT (Leifheit et al., 2018; Jaguš et al., 2018; Rodriguez, 2015), provide entertainment for the lesson (Bell & Vahrenhold 2018; Curzon, 2014) and help to overcome misconceptions or negative attitudes towards programming (Bell & Vahrenhold 2018). Researchers argued that using CS-unplugged activities would break the wall between CS and using computers in real life problem solving for children of young ages (Nishida et al., 2008; Lambert and Guiffre, 2009; Bell & Vahrenhold, 2018). Ahn, Sung, and Black (2021) also reported that CS-unplugged activities enhance younger students' problem-solving skills, debugging, and confidence, and to reduce the obstacles that programming can present for novice learners. Besides, unplugged approaches may be less intimidating to teachers without a background in CS or programming and avoid the high costs of teaching coding or dealing with hardware (Huang & Looi, 2021). Taking their advantages in learning with games, trial-and-error with real objects and learning within groups have made CS-unplugged popular in problem solving activities (Nishida et al., 2008).

There is a significant research effort invested on discussing the effects of CS-unplugged activities on problem solving skills to convey fundamental CS concepts to children without any computer skills in the schools (Bell et al., 2009; Prottsman, 2014; Wohl et al., 2015). According to Dwyer et al. (2014), while acting in CS-unplugged activities young children can describe problems, identify the requisites to solve the problem, break the problem into small logical steps, use these steps to meaningful problem-solving process, and then evaluate this process. In this sense, Alamer et al. (2015) used CS-unplugged in a camp to introduce programming concepts. In another study, Dwyer et al. (2014) implemented CS-unplugged to measure students' ability to work with systematic instructions in algorithms. In secondary education, Thies and Vahrenhold (2012) used CS-unplugged activities and addressed positive results in CT skills of students. There have been some projects undertaken to propose the potentials of CS-unplugged. For instance, Bebras is a test of computational problem solving that does not require the use of a programming language (Dagienė, et al., 2016; Gujberova & Kalas, 2013). In addition, (csunplugged.org) by CS Education Research Group in New Zealand introduces CT principles without using a computer (Bell, Witten, & Fellows, 2015). Another popular CS-unplugged project is Code.org. It introduces a blocky coding platform for preschool students through the 8th grade and older. It also covers a variety of algorithmic concepts that are connected to everyday life dedicated for children from the 4th year.

In order to improve problem-solving skills of young children, researchers suggested preparing activities focusing on children's developmental characteristics (Çetin, 2016). Although an increasing number of nations have plans for introducing CS-unplugged activities in early childhood, problem solving activities within CS-unplugged activities are not formally integrated into the preschool curriculum. Thus, a need exists to present models to guide the educators. Following the idea that CS-unplugged activities can promote young children to engage better in problem solving activities; this study seeks to examine the influences of CS-unplugged activities on their problem-solving skills.

### *1.2. Research Problem*

The aim of this research is exploring the development of problem-solving skills of preschool children with CS-unplugged activities. More specifically, the research question is “To what extent do the CS activities-unplugged develop young children’s problem-solving skills?” was investigated.

## **2. Method**

In this study, an instructional package including CS-unplugged activities was used for developing problem solving skills was implemented. A sequential explanatory mixed method was implemented in this research. This study undertakes the sequential approach where the quantitative phase is followed by the qualitative phase and the qualitative findings are used to contextualize the quantitative ones (Creswell, et al., 2003). Activity evaluation forms were used for evaluating the problem-solving skills quantitatively and interview data were used to explain the reasons of developments in problem solving skills through the students’ experiences.

During implementation, one of the researchers was observed the students’ behaviors in their learning environment as a participant researcher and tried to understand the atmosphere, language, or views of the group. At the same time, observation data were also used in the analysis of qualitative data by observing the group in depth through this participated researcher.

### *2.1. Participants*

This study was carried out with 11 children (aged 48-60 months) enrolled in a preschool. They did not take similar activities focusing directly problem-solving skills. They normally show the basic developmental characteristics behaviors of their young age during implementations which took place in the class environment.

### *2.2. Process*

The study lasted 5 weeks, 2 class hours per week with Code.org unplugged activities. Children at the age of 48-72 months can perform activities such as matching, establishing cause-effect relations, reading object graph and creating graphics with regard to their developmental characteristics (Piaget, 1976). Accordingly, the activities in Code.org were selected considering the motor, linguistics, cognitive and social development of young children characteristics. The activities covered direction, rhythm and classification skills including loops, conditionals or patterns. The children can provide different decisions, learn to carry out the iterative process to achieve tasks and produce tangible artifacts. The acquisitions covered in the activities are presented in Appendix 1. Three experts (2 preschool and 1 IT experts) reviewed the activities for content validity regarding the covered skills. Activities were selected based on in the 2013 preschool education program of the MoNE for 36-72 months old children considering the grouping and thinking skills on Code.org (Table 1). All activities in the Code.org have developmental foundations and evaluation worksheets, as well as various daily life problems appropriate for all age groups.

Code.org worksheet assessment forms were followed to develop a detailed lesson plan that would be applied in a 30-min. class period. The teacher introduced materials and basic problem-solving activities by following the lesson plans. The children worked on the tasks around common tables and followed the worksheets to complete the activity individually. The researcher only guided the children when they did not understand the tasks but did not explain how to solve the problem. The researcher as an observer took notes by observing the children’s behaviors and filled the evaluation form regarding their problem-solving performances. The research process is summarized in Figure 1.

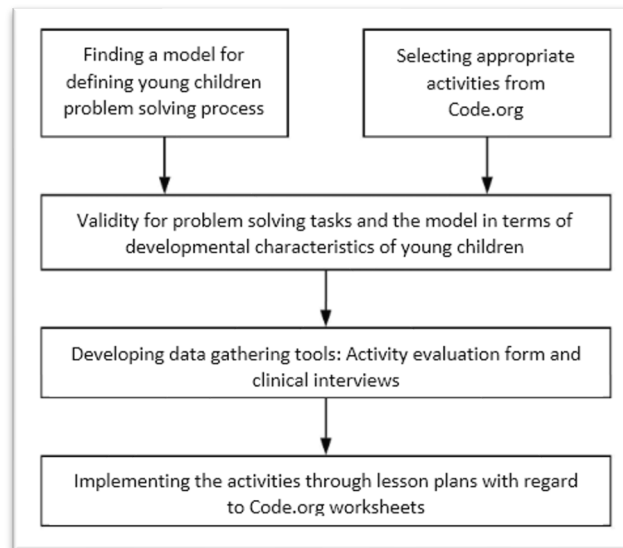
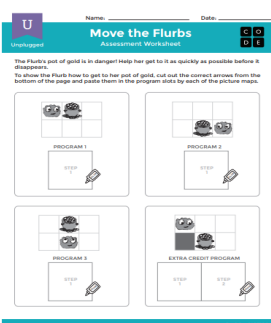


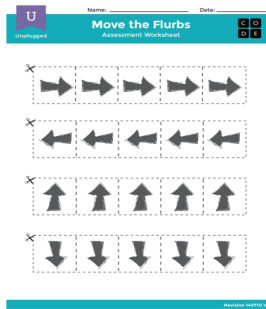
Figure1. Research Procedure

### 2.3. Selected CS-unplugged Activities for this Research

The “36-48 and 48-60-month-old Preschool Students' Developmental Characteristics Guide” was guided us to test the compatibility of the activities with the developmental characteristics of the children (Ministry of Education (MoNE), 2013). Table1 presents students’ developmental characteristics and the activities. The tasks in the activities are assigned into four steps (understanding, planning, implementation, and evaluation).

Table1. Activities Associated with Students’ Characteristics

Code.org Unplugged Activities	Pre-School Level	Language Development	Cognitive Development	Social Development	Motor Development
<b>1- Happy Maps</b> 	<b>36-48 Months Old Groups</b>	Expresses their feelings verbally.	Creates one-to-one matching.	Participates in group games.	Cuts the given simple shapes.
		Follows the rules under adult supervision.	Performs the printing paste operation.		
	<b>48-60 Months Old Groups</b>	Answers questions such as Why? How? Who?	Answers questions about the object/person/picture that he/she has seen a short while ago.	Adapts to adult/peer leadership.	Bounces the ball on the ground three times.



The activity involves constructing an algorithm that takes the character to the desired goal by cutting out simple shapes about instructions.

*CT Skill:*

Logical thinking,  
Algorithmic thinking,  
Problem solving

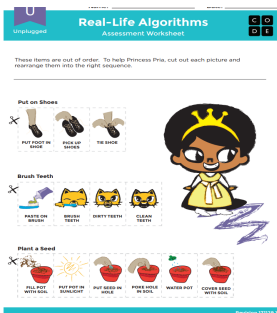
Answers questions about shortly simple stories.

Completes the missing parts in the pictures by looking at the example.

Answers questions involving cause-effect relationship.

**2-Real Life Algorithms**

**36-48 Months Old Groups**



In this activity, the algorithm flow of daily life examples is given in a mixed order and must be ordered correctly.

*CT Skill:* Algorithmic thinking, Efficiency, Problem solving

Expresses their feelings verbally.

Creates one-to-one matching.

Participates in group games.

Cuts the given simple shapes.

Describes two events in the order in which they occurred.

Identifies the object whose picture she sees.

Follows the rules under adult supervision.

Performs the printing paste operation.

Answers questions about him/her daily routine.

Continues the pattern consisting of two objects by looking at the model.

**48-60 Months Old Groups**


Answers questions such as Why? How? Who?

Sorts an event in the order in which it occurred.










Adapts to adult/peer leadership.

Answers questions about shortly

Answers questions about the object/person/pict

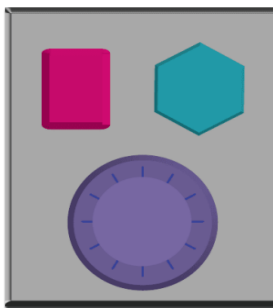
		simple stories.	ure that he/she has seen a short while ago.		
			Creates a story from the shown pictures.		
			Answers questions involving cause-effect relationship.		
<b>3-Getting Loopy</b>	<b>36-48 Months Old Groups</b>	Expresses their feelings verbally.	Shows the parts of their body which are said to her/him.	Participates in group games.	Cuts the given simple shapes.
		Describes two events in the order in which they occurred.		Follows the rules under adult supervision.	Performs the printing paste operation.
This activity includes recognizing repetitive steps and performing loops in order of flow with body movements including the language, self-care, cognition and motor development skills of children.	<b>48-60 Months Old Groups</b>	Performs two or three consecutive instructions.	Groups 1-5 objects according to their common properties.	Adapts to adult/peer leadership.	Makes simple dance steps.
<i>CT Skill:</i> Algorithmic thinking, Innovative thinking			Answers questions about the object/person/picture that he/she has seen a short while ago.	Efforts to go on the work he/she started.	
			Completes the missing parts in the pictures by looking at the example.		
			Answers questions		



		involving cause-effect relationship.			
<p><b>4-My Robotic Friends</b></p>  <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  Pick Up Cup         </div> <div style="text-align: center;">  Put Down Cup         </div> </div> <div style="display: flex; justify-content: space-around; align-items: center; margin-top: 10px;"> <div style="text-align: center;">  Step Forward         </div> <div style="text-align: center;">  Step Backward         </div> </div> <div style="display: flex; justify-content: space-around; align-items: center; margin-top: 10px;">   </div> <div style="display: flex; justify-content: space-around; align-items: center; margin-top: 10px;">   </div>	<p><b>36-48 Months Old</b></p> <p><b>Groups</b></p>	<p>Expresses their feelings verbally.</p> <hr/> <p>Describes two events in the order in which they occurred.</p>	<p>Sorts an event in the order in which it occurred.</p> <hr/> <p>Follows the rules under adult supervision.</p> <hr/> <p>Fulfills simple responsibilities</p> <hr/> <p>Answers questions about him/her.</p>	<p>Participates in group games.</p> <hr/> <p>Performs the printing paste operation.</p> <hr/> <p>Builds a tower by 8 cubes.</p>	<p>Cuts the given simple shapes.</p>
	<p>This activity includes making the different shapes of towers with plastic cups by following the given instructions.</p> <p><i>CT Skill:</i> Algorithmic thinking, Innovative thinking</p>	<p><b>48-60 Months Old</b></p> <p><b>Groups</b></p>	<p>Performs two or three consecutive instructions.</p> <hr/> <p>Performs tasks related to objects out of own sight.</p>	<p>Groups 1-5 objects according to their common properties.</p> <hr/> <p>Answers questions about the object/person/picture that he/she has seen a short while ago.</p> <hr/> <p>Completes the missing parts in the pictures by looking at the example.</p> <hr/> <p>Answers questions involving cause-effect relationship.</p>	<p>Adapts to adult/peer leadership.</p> <hr/> <p>Builds a tower by 10 cubes.</p>

**5-The Big Event**

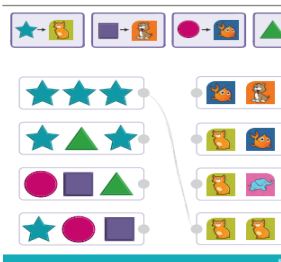
- Project the Event Controller onto your classroom screen



- Decide with your class what each button does. We s
  - Pink Button -> Say "Woooooo!"
  - Teal Button -> "Yeah!"
  - Purple Dial -> "Boom!"



You've been given a magical controller that changes the picture on the frame. Take a look below to see what each button does. Can you figure out which set events will cause your frame to show the pictures on the right? Draw a line fit of pictures to the button combination that causes it. The first one has been do



In this activity, the teacher asks the students when the teacher touches different shapes to make different sounds to get the meaning of the shapes. Then, with the same logic, the students are expected to match the shapes' representative animal characters.

*CT Skills:* Algorithmic thinking, Innovative thinking, Problem solving, Critical thinking

**36-48 Months Old Groups**

Expresses their feelings verbally.

Creates one-to-one matching.

Participates in group games.

Draws the model shown by looking at the example.

Performs two or three consecutive instructions.

Identifies the object whose picture she sees.

Follows the rules under adult supervision.

Continues the pattern consisting of two objects by looking at the model.

**48-60 Months Old Groups**

It tells what the source of the sound is.

Sorts an event in the order in which it occurred.

Adapts to adult/peer leadership.

Answers questions such as Why? How? Who?

Groups 1-5 objects according to their common properties.

Performs two or three consecutive instructions.

Completes the missing parts in the pictures by looking at the example.

Answers questions involving cause-effect relationship.

Compares objects according to their various properties.

*2.4. Data Collection Tools*

In this study, activity evaluation form and interviews were two main data collection tools.

*2.4.1. Activity Evaluation Form (AEF)*

AEF was developed for evaluating the problem-solving skills by monitoring the children's behaviors in the

problem-solving tasks and considering their perspectives about their experiences. Nance (2016) problem solving steps understanding, planning, doing, and looking back were taken as a framework for each activity.

*Understand:* The researcher asked, “What does this activity ask you to do?” and achievement of the student's ability to understand and verbally express the problem was evaluated.

*Plan:* The children were asked the question of “What will you do for this activity, what do you need to achieve the result?” to determine their planning of the problem.

*Do:* Children were asked to perform specific tasks for each activity. For example, in Happy map activity, they were expected to complete the task of the “Finding correct and short way arrows”.

*Look Back:* The questions “Do you think you got this activity right? Do you think what you did was right? Do you have any idea how to fix it if you think you did wrong?” were asked to the children to reveal how they check their solutions.

Considering Nance (2016) framework, each activity was divided into sub-tasks and indicators were defined for each of the tasks. The problem-solving performances were evaluated through these indicators. Students’ answers were scored as “satisfactory”, “partially-satisfactory” and “unsatisfactory” for each activity. The behaviors of the children in the activity were observed and confirmed with the interview data. Two researchers scored the students’ answers in the AEF individually. Then, they discussed the scores together until they came to an agreement about the scores. The scores were also confirmed via the teachers’ opinions. So, a triangulation is done with the quantitative and qualitative data handled to reveal the problem-solving skill development of the students.

For instance, in “Understanding” step, AEF was filled for the Activity-1 as described below.

*Satisfactory:* Using a correct sense of the expected expressions of the activity. For example, *using the arrows to bring the character to the apple etc.*

*Partially Satisfactory:* Although not emphasizing the expected concepts, short but meaningful expressions are explained. For example, *do not eat apple, do not go to the apple etc.*

*Unsatisfactory:* Wrong representation of different expressions or failure to fully understanding about the task. For example, *independent expressions or took apples to the character (vice versa).*

Activity	Appropriate Tasks for the event (Based on Nance's problem-solving steps)		Evaluation Criteria		
	<i>Problem Solving Step:</i> Understanding		Satisfactory	Partially-Satisfactory	Unsatisfactory
Happy map activity	<i>Question (Teacher):</i> What will you do in this event according to your opinion?				
	What will you do in this game?	use to "go to toward to apples via using arrows" or use a similar expression	use to "go to toward to apples or eat apples" use a similar but right expression		use to different expressions such as "apple should go to character"
	Task-1: To express the logical flow of the game verbally.  (Saying something similar to “character should go towards				

apple")

---

A view from the AEF including the “Understanding” step for the Happy map is presented in Table2.

Table2. A view from the activity evaluation form

In order to calculate the total scores obtained from all activities, if the “satisfactory” and “partially-satisfactory” scores for the activity were more than “unsatisfactory” scores, the children’s performance was assigned as “satisfactory” for that activity. If the scores obtained from all activities were “partially-satisfactory” and “unsatisfactory”, the score is assigned as “partially-satisfactory” for the activity. If the scores assigned for almost all activities were “unsatisfactory”, the student’s performance was defined “unsatisfactory” for the activity. The criteria for determining the total scores are presented in Table3.

Table 3. Criteria for determining the total programming performance of the children

---

<b>Evaluation Criteria</b>	<b>Assigned Scores</b>
Satisfactory + Partially Satisfactory score is more	Satisfactory
Partially- Satisfactory + Unsatisfactory scored is more	Partially- Satisfactory
Unsatisfactory scored is more	Unsatisfactory

---

Values from evaluation criteria when analyzing data obtained from this form are scored as Satisfactory=2, Partially Satisfactory =1, Unsatisfactory=0.

#### *2. 4. 2. Interviews*

Interviews were conducted one by one and lasted 10-15 minutes. The details of students’ artifacts such as “Why did you do that? Do you think you did right? How did you decide?” were asked to children to understand what the student thought when they were doing the tasks. Qualitative data were analyzed via content analysis by transcribing the interviews. To develop categories and codes, two coders read the children responses carefully. The codes were put into categories depending on the programming steps to address the programming performances.

### **3. Results**

In this section, children’s problem-solving skills were discussed regarding their performances and their behaviors.

#### *3.1. Problem solving performances in CS unplugged activities*

The children’s total problem-solving performances by taking into account the evaluation criteria (unsatisfactory, partially satisfactory, and satisfactory) in the AEF were shown in Figure 2.

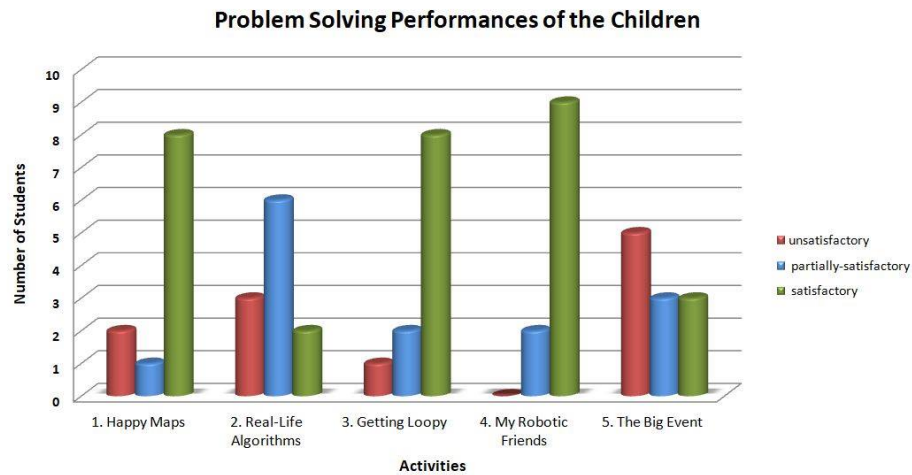


Figure 2. Problem solving performances of the children

Figure 2 shows that the lowest average problem solving performance that was observed in Activity 2, and the highest score as 2. The average scores regarding the steps of programming performances are normalized and presented as percentages is shown in Figure 3.

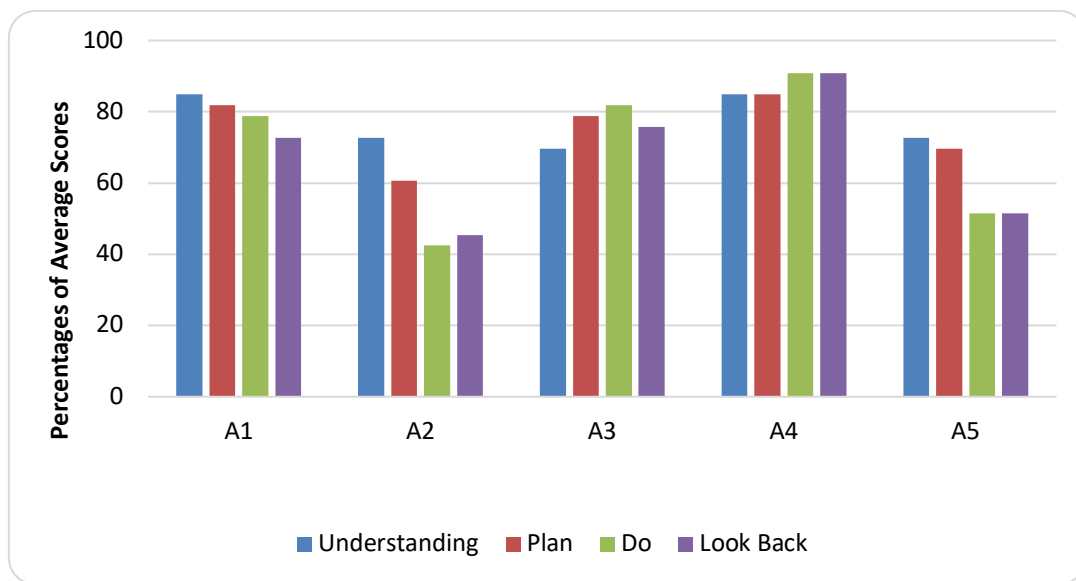


Figure 3. Average achievement scores in the problem-solving steps

Figure 3 shows the percentages of students' average problem-solving performances in all the four sub-steps scores (understanding, plan, do, and look back). For example, for the A1 activity, average score from 11 students was calculated as 2.54, and then this score converted to the percentage of 84.8% for representing A1 activity understanding sub-level. It is seen that, while the average problem-solving performances are high at A1 and A4, and those are relatively low in A2 and A5 which includes mostly ordering and matching activities.

The total problem-solving scores for all activities is shown in Figure 4.

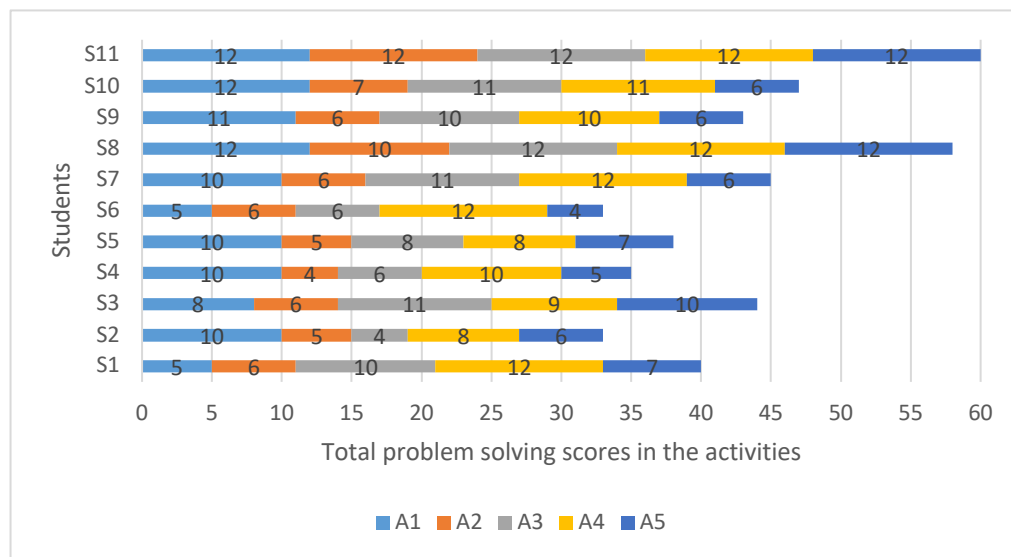


Figure 4. Total problem-solving scores in the activities

Figure 4 indicates that the scores from the activities vary for each child. For instance, while S2 got 10 from A1, she got low scores from other activities.

### 3.2. Students' Experiences in Problem Solving Tasks

In this section, the problem-solving performances in the activities are explained with regard to their experiences.

**3.2.1. Activity 1. Happy Maps:** The performances in the implementation and evaluation steps were slightly lower. In this activity, while determining the correct move within the relevant column for each step, some children *focused on only one column and chose all their moves from that column*.

In *planning step*: The majority of the children were able to express their actions in a specific order. However, in *doing step*, instead of completing the tasks as they planned, *they moved away from their planned solution ways*; namely, they did not follow their plans. Some of them attempted to continue with the cross moves. In this sense, S4 expressed that *"I used the short way because I thought that the character was hungry, and it was tired and needed to eat"*. This may be related to the imaginations about using an object for different purposes. This kind of imagination is frequently seen in young that they sometimes focus on other objects rather than the goal (Yeşilyaprak, 2018). Some of them also within their imaginations assigned some new meanings for the tasks and acted in the activities like playing games. In this sense, S7 expressed that *"I need to stick the arrows in the right direction and take this character to dinner by following the path (showing with his finger)"*. Although he expressed his plan by identifying correct steps, during the activity he tried many wrong ways. He explained this case with this statement *"...I know the right way for the solution, but I wanted him (character) to get confused so I didn't show it"*. Similarly, S3 explained the reasons why he did not apply his plan in the activity as *"I didn't want the beast to eat that food, because it always eats all food and is going to be very fat"*. In *looking back step*, S6 stated that *"I wanted it to go this way"* and he did not follow the directives and he did not look back to the situation. In general, although the children worked in the tasks as expected in this activity, the labyrinths, which were gradually getting difficult, made it difficult to apply the plans.

**3.2.2. Activity 2. Real Life:** In this activity, children were asked to arrange activities such as tying their shoes, brushing teeth and planting seed in a sequential manner. The tasks include ordering the pictures presented in a mixed order to form an integrity in accordance with the related games. In the Activity 2, the children performed high in *understanding* and *planning* steps, but the average scores taken from the tasks given in the *doing* and *looking back* steps were low. In the tooth brushing activity the children experienced the tasks in their daily lives. Similarly, in the shoe-tying activity as they previously experienced, children got high scores in understanding and planning tasks. Regarding this activity, some of the children when answered the questions like *"Why did you put a picture of clean-toothed cat at the end when sorting pictures in this event?"*, *"Do you think you did this activity right?"* An example answer is *"... because our teeth are clean when we brush our teeth, so I put it in the end (showing picture of a clean tooth cat)."*

In the seed planting activity, the performance of the children was relatively low especially during the *doing* step. In this activity, it was seen that some of the children copied other children's behaviors while performing the tasks. In addition, the symbols of the seeds used in planting seed and potted seedlings were not clear. It is thought that it is difficult to establish the relations among the pieces of a whole.

It is observed that children's experiences in the tasks significantly influenced their performance particularly in doing step. In fact, children who did not know the symbols in planting seed activity found it difficult to solve the problem. Considering the average scores of the three activities, the lower scores of the children can be thought as a reflection of the lower scores of the doing step. In addition, the fact that the planting seed (6 stages) activity was completed in more stages than shoe-tying (3 stages), tooth brushing (4 stages) activities may have influenced the low performance in implementing their plans.

**3.2.3. Activity3. Getting Loopy:** In this Activity, performances related to the tasks given for the *understanding* and *doing* steps were high but low at the *looking back* step. It was observed that all of the children in their plans provided repeated some actions (loop). However, the majority of the children could not determine the number of repetitions in this process, and they could not show the number of loops in the plans correctly. In this sense, S5 addressed that "...I will repeat clapping, clapping, clapping as my teacher doing" while repeating processes. Another student S3 stated, "...I'll repeat the same picture, but I don't know how many times".

On the other hand, the tasks in the doing step were done by following the teacher's presentation. The presentations helped children to get high scores even though the children had deficiencies in their plans. At the looking back step, some of the children could not perform the tasks by assigning the number of repetitions correctly.

**3.2.4. Activity4. My Robotic Friends:** In this activity, the children were asked to put the cups in order as in the pattern given on the worksheet. A view from students' actions is seen in Figure 5.

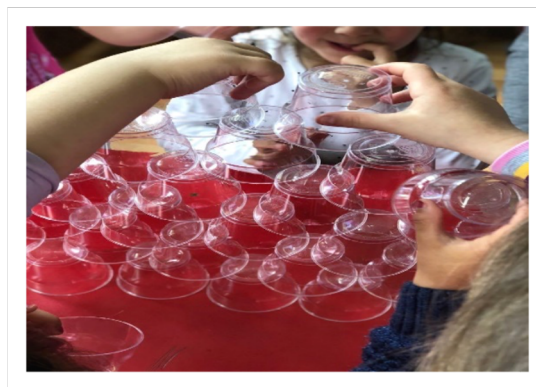


Figure 5. A view from My Robotic Activity

In this activity, almost all of the children showed high performance in *understanding* and *planning*, and all of the children were successful in *doing* and *looking back*. For instance, S4 expressed that "... I will put the cups in order as shown in picture", "We were like robots in this activity". Also, a number of children were able to decide the number of cups for the correct solution in the planning step. In addition, it is seen that in all of the tasks in the doing step, the children were able to put the cups in order as expected pattern. Using concrete object such as plastic cups may be considered as one of the reasons of high performance in this step. Using daily-recognized objects and allowing these objects to create the patterns by heuristic approach might have been contributed to this achievement. Accordingly, S1 identified that "sometimes I'm confused while I'm putting the cups in order, but then I just lined up like a tower." One other reason for the high performance of the children at the looking back step may be related to the developing a concrete product. In this sense, the children could review the paths they needed to follow when they could not do it correctly.

**3.2.5. Activity5. Big Event:** In this activity, the children were asked to match the geometric shapes with the appropriate animal figures.

Figure 3 shows that the majority of the children performed high in *understanding* and *planning* steps in Activity5. The performances in the doing and looking back steps were quite low. The average scores in the *doing* and *looking back* steps were lower than the other steps. Children's perspectives reflect that one reason for the low scores may be the fact that the children cannot remember more than one pattern. For instance, S5 expressed that

“...When I forgot the pattern, I looked up again and I waited to put my finger on it to not forget” S4 also explained his action as “...I've confused which shape corresponds to which animal”. As in other activities, it can be thought that the decrease in the performance of the problem solving is regarded to the complexity of cognitive tasks such as keeping the one more pattern in mind and creating multiple patterns. In addition, in this activity it is expected that while matching, first; understand the hint box given in the introduction, and then keep this information in mind, then adapt the following question according to the situation, and lastly, use this information to reach the desired result. Since this activity requires sequential follow-up and mental processes, it is not easy to perform the expected tasks in this age group sequentially.

Figure 3 indicates that in Activity 1, Activity 2 and Activity 5, the understanding steps were completed more successfully than the other steps, whereas the doing step were more successful than the other steps in the Activity 3 and Activity 4. It is seen that the planning step is constructed accordingly depending on how well the student completes the understanding step. It is noteworthy that children performed lower in the looking back steps except for Activity 2 and Activity 4 than other steps. At this point, the result may be about the nature of the activities Activity 2 and Activity 4. Because Activity 2 is more directly related to the daily life than others and Activity 4 (My robotic friends) addressed more motor skills.

#### **4. Discussion**

This study attempted to determine the effect of CS-unplugged activities on problem solving skills of preschool children. The results indicated that the significant increase in problem solving skills may be due to the activities designed in accordance with the learning objectives. In this study, in all of the activities the children performed high in understanding and planning steps compared to the other steps. The doing step, which is usually used to cut, paste, match, sort, etc. resulted in lower scores due to cognitive skills as well as hand skills. It is surprising that although the scores during the looking back steps were low, the children began to perform the tasks correctly in this step.

The current study confirmed that CS-unplugged activities can support the young children to establish a relationship between activity and real life. Namely, the activities including concrete events such as putting the cups in order together activity can provide high performance rates at the looking back steps. In Activity 2, although the children did not experience a problem before, they could understand the task, but mostly they could not perform high scores in doing step. In this context, one can infer that the activities that the children experienced before can support children's performances in doing step as in the understanding step. For instance, in Activity 1 (Happy Maps) and Activity 5 (Big Event), children's understanding and planning performances were high, but they could not perform the similar performance in the doing and looking back steps. In these two activities, keeping multiple moves in mind, matching multiple images and performing sequential operations are some examples requiring the advanced cognitive skills in which the children could understand the problem but not perform high in the doing step.

The findings showed that study concluded that the design of the activities and the roles attained to the children influenced the development of their problem-solving skills. Similarly, another study suggested that the activities in CS-unplugged activities should be explicitly linked to central concepts in CS (Taub, Armoni, & Ben-Ari, 2012). In accord to this study, Faber et al. (2017) found that the unplugged materials seem to elicit positive reactions from children. Another reason for the achievement in activities is the nature of the activities that preschool students are generally considered to have high performance due to their similarity to cut-and-paste activities. In accord to this finding; a comparative experiment by Montes-Leon et al. (2020) found that the introduction of unplugged programming activities could help students improve their CT skills and have a positive effect on their follow-up programming learning.

It is important for children to engage in the tasks of problem-solving activities. Actually, it is also known that the attention of the young children can be distracted quickly, and it is difficult to engage them in different tasks during the activities (Radesky & Christakis, 2016; Rodriguez et al, 2016). In the current study, attractive potentials of the tasks in CS-unplugged can be seen as engaging children in the activities and being active in problem solving. As suggested in prior studies, we ensured that all children had a role to act in the activities (any amount of down time potentially results in bored, disengaged children and lower assessment results).

Research has shown that unplugged programming activities can effectively develop 5–7-year-old children CT skills and help them transfer CT skills to other problem-solving scenarios (Conde et al., 2017; Wohl et al., 2015). Unplugged debugging activities without the use of programming tools provide more content-focused learning experiences for younger students by reducing the cognitive demands for using technological tools (Kotsopoulos et al., 2017). This study also confirmed that CS-unplugged activities including objects or concepts that the



children have experienced before positively influence on their planning and doing performances. The results also indicated that, the activities should also be designed considering the “imaginary world” of the children. Namely, something in the activity may remind them of some different events in their mind. Hence, the activities including tangible and basic materials may eliminate this and may provide better problem-solving outcomes.

This study highlights that CS-unplugged activities may provide successful outcomes for problem solving of preschool students. The preschool children can engage cognitively, socially, and creatively in the CS-unplugged activities. In this study we separated the problems into the tasks by associating them with the substages of problem solving. In this way, we could evaluate the problem-solving processes regarding their achievements in the tasks in the understanding, planning, doing and looking back steps. With both findings provide potential avenues for future problem solving, this study moves us one step closer to uncovering a way to evaluate the young children’s problem-solving process.

This research is not exempt from limitations. The most important of which is its exploratory nature. It is difficult to provide quantitative data about the young children’s problem-solving processes. It should be noted that this study focused on only 5 activities to evaluate the problem-solving performances in CS-unplugged activities. Implementing activities by taking objectives in the preschool curriculum and students’ developmental characteristics into consideration played a positive role about the implementation process. This study used only students’ answers and observations assess the problem-solving processes. In future studies, data from the video records including the interactions among children would support evaluating the problem-solving processes more accurately. Moreover, Taub, Armoni and Ben-Ari (2012) pointed out that it is difficult to demonstrate that CS-unplugged activities actually achieve long-term goals about directing young children’s interest in CS concepts. Hence, further longitudinal studies may be helpful in clarifying the effect of CS-unplugged activities to the achievements in CS. One other limitation was the small sample size; thus, a larger sample size would increase the sensitivity of the analysis.

## **5. Conclusion**

This study considered the cognitive development of the young children by directing the roles of children to problem solving and evaluated developments in their problem-solving skills. The results indicate that even if the children's plans about the tasks is correct, sometimes the problem-solving process cannot be fully completed as expected in the doing and looking back steps. The tasks in the activities were also found influential on achieving problem solving steps. Preschool children’s developmental and working memory characteristics and their previous experiences about the objects and the tasks in the activities also influenced their problem-solving process.

Overall, the contribution of the findings of this study is in two folds. One is about the design attributes of the problem-solving activities for preschool children. The second is about the evaluation process of CS-unplugged activities in terms of problem solving. It is recommended to design worksheets that are both engaging for the children and directing them to problem solving process. Incorporating worksheets or assessment techniques into lesson plans of preschools is also crucial to take the advantage of problem solving in the early ages. Instructional designers should take care when deciding to design certain types of learning activities considering children’s developmental characteristics. Educators can adapt CS-unplugged activities to their lessons are to build and maintain a collaborative classroom environment and refer them when teaching abstract concepts and solving daily problems in preschool classrooms. We hope that the findings of this study would assist in future design and implementation of CS- unplugged activities for young children.

## **References**

- Ahn, J., Sung, W., & Black, J. B. (2021). Unplugged debugging activities for developing young learners’ debugging skills. *Journal of Research in Childhood Education*, 1-17. <https://doi.org/10.1080/02568543.2021.1981503>
- AlAmer, R. A., Al-Doweesh, W. A., Al-Khalifa, H. S., & Al-Razgan, M. S. (2015). Programming unplugged: bridging CS unplugged activities gap for learning key programming concepts. In *2015 Fifth International Conference on e-Learning (econf)* (pp. 97-103). IEEE. <https://doi.org/10.1109/ECONF.2015.27>
- Barr, D., Harrison, J. and Conery, L. (2011). Computational thinking: A dijital age skill for everyone. *Learning & Leading with Technology*, 38(6), 20-23. [Google Scholar](#)

- Battal, A., Afacan Adanır, G., & Gülbahar, Y. (2021). Computer Science Unplugged: A Systematic Literature Review. *Journal of Educational Technology Systems*, 50(1), 25-47. <https://doi.org/10.1177/00472395211018801>
- Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13(1), 20-29. [Google Scholar](#)
- Bell, T., & Vahrenhold, J. (2018). CS unplugged-how is it used, and does it work? In H.J.Böckenhauer, D. Komm & W. Unger (Eds.), *Adventures between lower bounds and higher altitudes* (pp. 497–521). Springer. [https://doi.org/10.1007/978-3-319-98355-4\\_29](https://doi.org/10.1007/978-3-319-98355-4_29)
- Bell, T., Witten, I., & Fellows, M. (2015). *CS Unplugged: An enrichment and extension programme for primary-aged students*. Retrieved 10 April 2018 from [http://csunplugged.org/wp-content/uploads/2015/03/CSUnplugged\\_OS\\_2015\\_v3.1.pdf](http://csunplugged.org/wp-content/uploads/2015/03/CSUnplugged_OS_2015_v3.1.pdf)
- Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, 72, 145-157. <https://doi.org/10.1016/j.compedu.2013.10.020>
- Brown, Q., Mongan, W., Kusic, D., Garbarine, E., Fromm, E. and Fontecchio, A. (2013). *Computer aided instruction as a vehicle for problem solving: Scratch programming environment in the middle years classroom*. *Proceedings of the Annual Conference and Exposition*, Pittsburg, Pensilvania USA. [Google Scholar](#)
- 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). ACM. <https://doi.org/10.1145/3137065.3137069>
- Bransford, J. D., Zech, L., Schwartz, D., Barron, B., Vye, N., & CTGV. (2000). Designs for environments that invite and sustain mathematical thinking. In Cobb, P. (Ed.), *Symbolizing, communicating, and mathematizing: Perspectives on discourse, tools, and instructional design*. (pp. 275-324). Mahwah, NJ: Erlbaum
- Caelien, 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>
- Conde, M. A., Ferná ´ndez-Llamas, C., Rodriguez-Sedano, F., Guerrero-Higuera, A. M., Matella ´n-Olivera, V., & Garcia-Penalvo, F. J. (2017). Promoting computational thinking in K - 12 students by applying unplugged methods and robotics. In J. M. Doder, M. S. I. Sa ´iz, & I. R. Rube (Eds.), *Proceedings of the Fifth International Conference on Technological Ecosystems for Enhancing Multiculturalism* (TEEM 2017). Ca ´diz, Spain: ACM. <https://doi.org/10.1145/3144826.3145355>
- Creswell, J. W., Plano Clark, V. L., Gutmann, M. L. and Hanson, W. E. (2003). Advanced mixed methods research designs. In A. Tashakkori, C. Teddlie (Eds.), *Handbook of mixed methods in social and behavioral research* (pp.209–240). Thousand Oaks, CA: Sage,
- Curzon, P. (2014). Unplugged computational thinking for fun. Retrieved April 25, 2020 from: [https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/8257/file/cid07\\_S15-27.pdf](https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/8257/file/cid07_S15-27.pdf)
- Çetin, E. (2016). *A case study for the use of technology aided graphical organizers in preschool children's problem solving process*. Unpublished PhD. Dissertation, Gazi University, Ankara.
- Daan, W., Cooper, S., & Pausch, R. (2009). *Learning to Program with ALICE*. Pearson education. [Google Scholar](#)
- Dagien ´e, V., Stupurien ´e, G., Vinikien ´e, L. (2016). Promoting inclusive informatics education through the Bebras Challenge to all K-12 students. In: *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016* (pp. 407–414). ACM. <https://doi.org/10.1145/2983468.2983517>
- Del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of Primary Education. *Computers & Education*, 150, 103832. <https://doi.org/10.1016/j.compedu.2020.103832>

- Dwyer, H., Hill, C., Carpenter, S., Harlow, D., & Franklin, D. (2014). Identifying elementary students' pre-instructional ability to develop algorithms and step-by-step instructions. In *Proceedings of the 45th ACM Technical Symposium On Computer Science Education* (pp. 511-516). ACM. <https://doi.org/10.1145/2538862.2538905>
- Faber, H. H., Wierdsma, M. D., Doornbos, R. P., van der Ven, J. S., & de Vette, K. (2017). Teaching computational thinking to primary school students via unplugged programming lessons. *Journal of the European Teacher Education Network*, 12, 13-24.
- Gardeli, A., & Vosinakis, S. (2017). Creating the computer player: An engaging and collaborative approach to introduce computational thinking by combining unplugged activities with visual programming. *Italian Journal of Educational Technology*, 25(2), 36–50. <https://doi.org/10.17471/2499-4324/910>
- Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Stoker, I. (2014). Computational thinking skills in Dutch secondary education: exploring pedagogical content knowledge. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research* (pp. 173-174). ACM. <https://doi.org/10.1145/2674683.2674704>
- Grout, V., & Houlden, N. (2014). Taking computer science and programming into schools: The Glyndŵr/BCS Turing project. *Procedia-Social and Behavioral Sciences*, 141, 680-685. <https://doi.org/10.1016/j.sbspro.2014.05.119>
- 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>
- Gujberova, M., & Kalas, I. (2013). Designing productive gradations of tasks in primary programming education. In: *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, pp. 108–117. <https://doi.org/10.1145/2532748.2532750>
- Hermans, F., Aivaloglou, E. (2017): To scratch or not to scratch?: a controlled experiment comparing plugged first and unplugged first programming lessons. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education, WiPSCE 2017*, pp. 49–56. ACM, New York. <https://doi.org/10.1145/3137065.3137072>
- Hodhod, R., Khan, S., Kurt-Peker, Y., & Ray, L. (2016). Training teachers to integrate computational thinking into K-12 teaching. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 156-157). ACM. <https://doi.org/10.1145/2839509.2844675>
- Huang, W., & Looi, C. K. (2021). A critical review of literature on “unplugged” pedagogies in K-12 computer science and computational thinking education. *Computer Science Education*, 31(1), 83-111. <https://doi.org/10.1080/08993408.2020.1789411>
- Jagušt, T., Krzic, A. S., Gledec, G., Grgić, M., & Bojic, I. (2018). Exploring different unplugged game-like activities for teaching computational thinking. In *2018 IEEE Frontiers in Education Conference (FIE)* (pp.1-5). IEEE. <https://doi.org/10.1109/FIE.2018.8659077>
- Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan*, 95(1), 61-65. <https://doi.org/10.1177/003172171309500111>
- Kalelioğlu, F., & Keskinılıç, F. (2017). Teaching methods for computer science education. In Y. Gülbahar (Ed.), *From computational thinking to programming* (1st ed. pp. 155–182). Ankara.
- Kalelioğlu, F., Gülbahar, Y., Akçay, S., & Doğan, D. (2014). Curriculum integration ideas for improving the computational thinking skills of learners through programming via scratch. In *Local Proceedings of the 7th International Conference on Informatics in Schools: Situation, Evolution and Perspectives* (pp. 101-112). [Google Scholar](#)
- Kotsopoulos, D., Floyd, L., Khan, S., Namukasa, I. K., Somanath, S., Weber, J., & Yiu, C. (2017). A pedagogical framework for computational thinking. *Digital Experiences in Mathematics Education*, 3(2), 154-171. <https://doi.org/10.1007/s40751-017-0031-2>
- Lambert, L., & Guiffre, H. (2009). Computer science outreach in an elementary school. *Journal of Computing Sciences in Colleges*, 24(3), 118-124. [Google Scholar](#)

- Lee, I., Martin, F., & Apone, K. (2014). Integrating computational thinking across the K—8 curriculum. *AcM Inroads*, 5(4), 64-71. <https://doi.org/10.1145/2684721.2684736>
- Leifheit, L., Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018). Programming unplugged: An evaluation of game-based methods for teaching computational thinking in primary school. In M. Ciussi (Ed.), *ECGBL 2018 12th European Conference on Game-Based Learning* (p. 344). Academic Conferences and Publishing Limited. [Google Scholar](#)
- [Milli Eğitim Bakanlığı (MEB)], 2013. Okulöncesi eğitim programı. Retrieved 10 May 2018 from <http://tegm.meb.gov.tr/dosya/okuloncesi/ooproram.pdf>
- Montes-León, H., Hijón-Neira, R., Pérez-Marín, D., & Montes-León, R. (2020). Mejora del pensamiento computacional en estudiantes de secundaria con tareas unplugged. *Education in the Knowledge Society*, 21, 24. [Google Scholar](#)
- Nance, S. (2016). *Using computer programming to enhance problem-solving skills of fifth grade students*. Unpublished PhD. Thesis, University of Florida, Gainesville.
- Nishida, T., Idosaka, Y., Hofuku, Y., Kanemune, S., & Kuno, Y. (2008). New methodology of information education with “Computer science unplugged”. In *International Conference on Informatics in Secondary Schools-Evolution and Perspectives* (pp. 241-252). Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-540-69924-8\\_22](https://doi.org/10.1007/978-3-540-69924-8_22)
- Piaget, J. (1954). *The construction of reality in the child* (M. Cook, Trans.). New York, NY, US.
- Piaget, J. (1976). Piaget’s theory. In *Piaget and his school* (pp. 11-23). Springer, Berlin, Heidelberg.
- Protsman, K. (2014). Computer science for the elementary classroom. *ACM Inroads*, 5(4), 60-63. <https://doi.org/10.1145/2684721.2684735>
- Radesky, J. S., & Christakis, D. A. (2016). Keeping children’s attention: The problem with bells and whistles. *JAMA pediatrics*, 170(2), 112-113. <https://doi.org/10.1001/jamapediatrics.2015.3877>
- Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 265-269). ACM. <https://doi.org/10.1145/1734263.1734357>
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... and Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60-67. [Google Scholar](#)
- Rodriguez, B. R. (2015). *Assessing computational thinking in computer science unplugged activities*. Unpublished Masters thesis. Golden: Colorado School of Mines.
- Rodriguez, B., Rader, C., & Camp, T. (2016). Using student performance to assess CS unplugged activities in a classroom environment. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 95-100). ACM. <https://doi.org/10.1145/2899415.2899465>
- Taub, R., Ben-Ari, M., & Armoni, M. (2009). The effect of CS unplugged on middle-school students’ views of CS. *ACM SIGCSE Bulletin*, 41(3), 99–103. <https://doi.org/10.1145/1595496>
- Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS unplugged and middle-school students’ views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education (TOCE)*, 12(2), 8. <https://doi.org/10.1145/2160547.2160551>
- Thies, R., & Vahrenhold, J. (2012). Reflections on outreach programs in CS classes: learning objectives for unplugged activities. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 487-492). ACM. <https://doi.org/10.1145/2157136.2157281>
- Uchida, Y., Matsuno, S., Ito, T., & Sakamoto, M. (2015). A proposal for teaching programming through the five-step method. *Journal of Robotics, Networking and Artificial Life*, 2(3), 153. <https://doi.org/10.2991/jrnal.2015.2.3.4>
- Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 20(4), 715-728. <https://doi.org/10.1007/s10639-015-9412-6>

- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35. [Google Scholar](#).
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 366(1881), 3717-3725. <https://doi.org/10.1098/rsta.2008.0118>
- Wing, J. (2011). Research notebook: Computational thinking-What and why? *The Link Newsletter*, 6, 1-32. Retrieved from [http://link.cs.cmu.edu/files/11-399\\_The\\_Link\\_Newspaper-3.pdf](http://link.cs.cmu.edu/files/11-399_The_Link_Newspaper-3.pdf). [Google Scholar](#).
- Wohl, B., Porter, B., & Clinch, S. (2015, November). Teaching Computer Science to 5-7 year-olds: An initial study with Scratch, Cubelets and unplugged computing. In *Proceedings of the Workshop In Primary And Secondary Computing Education* (pp. 55-60). <https://doi.org/10.1145/2818314.2818340>
- Yeşilyaprak, B. (2006). *Education Psychology*. Pagema Publisher, 376s, Ankara.
- Zur Bargury, I., Haberman, B., Cohen, A., Muller, O., Zohar, D., Levy, D., & Hotoveli, R. (2012, October). Implementing a new Computer Science Curriculum for middle school in Israel. In *2012 Frontiers in Education Conference Proceedings* (pp. 1-6). IEEE. <https://doi.org/10.1109/FIE.2012.6462365>

### **Appendix1**

- Completes puzzle with 10-25 pieces.
- Creates new shapes by combining geometric shapes
- Groups the objects
- Performs addition and subtraction operations of at least 1-10 numbers.
- Explains how to do match, associate and group Establish cause-effect relationships
- Uses comparison statements

