IJCSES

Volume 4, No: 1
August 2020
ISSN 2513-8359

International Journal of Computer Science Education In Schools

Editors

Dr Filiz Kalelioglu

Dr Yasemin Allsop

www.ijcses.org

International Journal of Computer Science Education in Schools

August 2020, Vol 4, No 1

DOI: 10.21585/ijcses.v4i1

Table of Contents

International Journal of Computer Science Education in Schools, August 2020, Vol. 4, No. 1
ISSN 2513-8359

Plugged or Unplugged Teaching: A Case Study of Students' Preferences for the Teaching Programming

Ali Kürşat Erümit[1]
Güven Şahin[2]

[1]Trabzon University
[2]Recep Tayyip Erdoğan University

DOI: DOI 10.21585/ijcses.v4i1.82

Abstract
This study is an investigation of the effects of plugged and unplugged activities in a programming course using the Programming in Seven Steps (PSS) model on pupils' satisfaction and activity type preferences. A case study method was used in the classroom was the case. Data included students' diary, their responses on semi-structured interview forms, and semi-structured interviews with a selected sub-set of students. The collected data were analyzed by content analysis technique. It has been found that there are different factors that positively affect student satisfaction for "Conditional Structures", "Variables" and "Loops" courses that are processed according to YAP model. In addition, students' preferences and reasons for activity type differ. Study results show that training students with their preferred activity types increases their satisfaction and enable them to overcome associated difficulties more easily. It is concluded that because the PSS model is effective with students with different learning environment preferences, it can be used as a model to increase learner satisfaction with programming instruction.

Keywords: teaching programming, PSS model, plugged activities, unplugged activities, student satisfaction

1. Introduction

With the rapid advance of the developments in the field of science and technology since the industrial revolution up to the present, the skills individuals need to have in the 21st century have evolved. Wagner (2008) lists such skills as critical thinking and problem solving, quick wit and adaptation, entrepreneurship and taking initiative, and inter-system and interpersonal cooperation as well as leadership, effective oral and written communication, curiosity and imagination, and ability to access and analyze information.

Studies on acquisition of these 21st century skills reveal that teaching programming significantly improves students' logical thinking (Shih, 2014), problem solving (Brown et al., 2013; Lai and Lai, 2012; Kalelioğlu and Gülbahar, 2014), computational thinking (Sáez-López et al., 2016), reflective thinking (Fesakis and Serafeim, 2009; Kobsiripat, 2015), metacognition (Fessakis, Gouli, and Mavrodi, 2013; Kafai and Burke, 2014), and collaborative work (Denner, Werner, and Ortiz, 2012). In addition to these skills, learning programming has been found to have a positive effect on students' academic achievement in a variety of subjects (computers, mathematics, science, language arts, and arts education) (Calder, 2010; Sáez-López et al., 2016).

The literature shows that teaching programming holds an important potential for helping students gain 21st century skills. Through programming teaching in the literature; students' logical thinking (Lai & Lai, 2012; Shih, 2014), problem solving (Brown et al., 2013; Lai & Lai, 2012), computational thinking (Sáez-López et al., 2016), higher level thinking There has been positive progress in skills (Kafai & Burke, 2014) and critical thinking (Erümit et al., 2019). Besides developing students' cognitive skills, however, programming requires other higher-level skills for the learning process itself (Law, Lee and Yu, 2010), which makes learning programming is quite challenging (Helminen and Malmi, 2010), and students have an overall lower level of achievement in this subject (Robins, Rountree, and Rountree, 2003). In order to increase the success of teaching programming and to facilitate students' understanding, it is necessary to first teach the logic of algorithms to students (Ala-Mutka, 2004). For this purpose, interesting and entertaining visual programming tools have been developed in order to facilitate learning for beginners in programming education (Schwartz, Stagner, and Morrison, 2006).

Teaching programming entails challenges in selecting suitable activities for a group of learners (Çatlak, Tekdal, and Baz, 2015), helping students understand and apply algorithms (Futschek and Moschitz, 2010), and helping them use programming languages in writing codes (Arabacıoğlu, Bülbül, and Filiz, 2007). The syntax of text-based programming is considered as one of the most challenging issues for students (Mannila, Peltomaki, and Salakoski, 2006; Özmen and Altun, 2014), while the code blocks presented in visual programming are easier for beginning level programmers to understand and apply (Wilson and Moffat, 2010). Calder (2010) argues that working with the block-based visual tools of programming software increases students' satisfaction and motivation to persist. In many studies, it was noted that the use of visual programming accelerates comprehension of the process (Naharro-Berrocal, Pareja-Flores, Urquiza-Fuentes, and Velazquez-Iturbide, 2002). For beginning learners, therefore, elementary level software programs such as Scratch (Malan and Leitner, 2007; Wu, Chang and He, 2010), Kodu (Stolee and Fristoe, 2011), StarLogo (Klopfer and Yoon, 2005), and Alice (Kelleher, Pausch, and Kiesler, 2007) are recommended as they allow students to perform coding by placing code blocks in order through drag and drop functions. In this way, the frequently faced problem of syntactic errors in text-based programming can be largely overcome.

In addition, attitudes toward programming (Gomes and Mendes, 2007; Hawi, 2010) and self-efficacy beliefs (Aşkar and Davenport, 2009) are some of the affective issues encountered in the teaching programming. These challenges often arise during the teaching of basic programming concepts such as the structure of a programming language (Lahtinen, Ala-Mutka, and Järvinen, 2005), loops (Ginat, 2004), and algorithm structures (Seppälä, Malmi, and Korhonen, 2006).

Another major challenge encountered in teaching programming relates to overall pedagogical approaches (Coull and Duncan, 2011; Lahtinen, Mutka, and Jarvinen, 2005), such as the following as found in the literature: Problem solving (Asad, Tibi, and Raiyn, 2016; Grover, Pea, and Cooper, 2016; Qian and Lehman, 2016), abstraction (Futschek and Moschitz, 2010), peer instruction (Denner, Werner, and Ortiz, 2012), collaborative learning (Denner, Werner, and Ortiz, 2012; Denner et al., 2014), writing algorithms (Futschek, 2006; Futschek and Moschitz, 2010; Milková and Hůlková, 2013), and drama or role plays (Karaosmanoğlu and Adıgüzel, 2017; Sarıoğlu and Kartal, 2017; Weigend, 2014).

In the implementation of these approaches, both plugged and unplugged activities (activities with and without use of computers) may be used. For example, Sáez-López, Román-González and Vázquez-Cano (2016) carried out a study with 107 middle schoolers using Scratch and computerized activities. Their purpose was to explore the differences, if any, their approach made in the students' attitudes and competencies. They found that the students made progress in satisfaction, attachment, computational thinking, and calculating skills. Taylor, Harlow and Forret (2010) implemented plugged activities with 9- and 10-year-old students using Scratch and found out that the learners were interested in Scratch as a programming tool. In addition, they pointed out that the tool provides an environment in which students employ problem solving processes such as idea generation, target setting and testing. Similar studies using visual programming tools like Scratch have shown that such tools improved students' satisfaction, interest, and enjoyment in learning (Kalelioğlu and Gülbahar, 2014; Kelleher et al., 2007; Malan and Leitner, 2007; Pinto and Escudeiro, 2014; Wu et al., 2010).

Especially at an early age, there are difficulties in learning abstract programming topics. As a solution, unplugged activities can be used to teach programming. Because unplugged activities contain the lowest level of technical information, they involve fewer cognitive challenges than other programming tasks and are commonly used in basic computer instruction as a starting point before student progress to plugged activities (Kotsopoulos et al., 2017). Starting teaching programming according to the activity type preferences (plugged/unplugged) of the students or planning the activities to include both types of activities can facilitate the teaching programming for younger students and affect their perceptions and attitudes towards programming positively (Şahin, 2018). For example, Futschek and Moschitz (2011) introduced unplugged activities prior to plugged activities to students at the basic level of programming. They found out that this sequence of teaching increased students' satisfaction with plugged activities, and even those who previously had not been interested in programming ended up being enthusiastic. Giannakos et al. (2013) also found that using both plugged and unplugged activities increased students' interest in and satisfaction with both programming and computer sciences as a subject. They also found that female students became more interested in computer sciences than their male peers. Hermans and Aivaloglou (2017) investigated which method, plugged or unplugged, is more effective as a starting activity in programming in order to facilitate learning the concepts of programming, support learners' self-efficacy in programming tasks, and motivate them to do research. As a result, the group starting with unplugged activities recorded higher self-efficacy. Also, those students used a larger range of Scratch blocks.

Pedagogical approaches and types of activities (plugged and unplugged) mentioned in the literature, however, tend to be applied independently in programming instruction. Moreover, there exists no specific model for these coordinating these pedagogical approaches in class and laboratory settings with the age of the target group, for making them workable steps, and thus for systematizing programming instruction. Targeting these problems, Erümit et al. (2019) devised a teaching model entitled Programming in Seven Steps (PSS) for teaching programming so as to improve students' algorithmic thinking and problem-solving skills independent of any specific programming environment.

The first four steps, “Understand the Problem,” “Devise a Plan,” “Compare the Strategies,” and “Devise an Algorithm,” use unplugged activities. The last three steps, “Code the Algorithm,” “Identify and Correct an Error in a Different Code,” and “Prepare and Code New Algorithms,” involve plugged activities. These steps are further elaborated below.

The seven steps of the model include plugged and unplugged activities as shown in Figure 1:

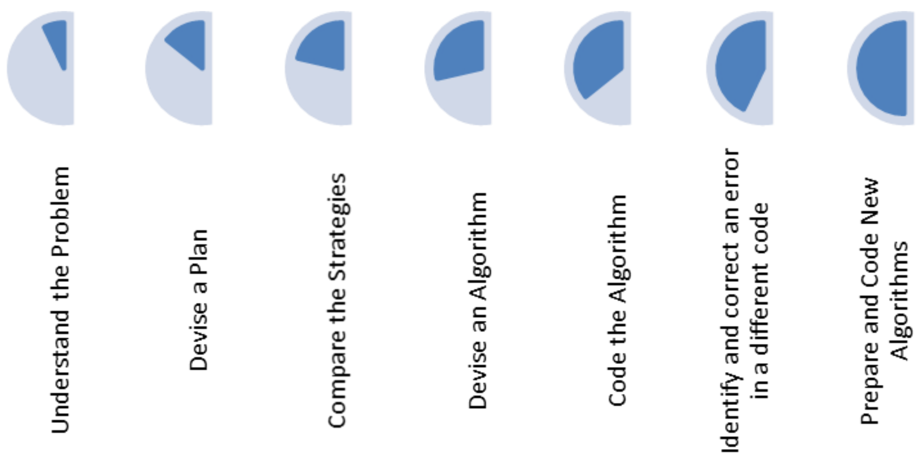


Figure 1. Steps of the PSS Model

The first step - understand the problem involves identifying what relevant information is given in the problem, what further information is needed, and what given information is irrelevant, more or less simultaneously. The next step, devise a plan, requires developing strategies appropriate to seeking the solution of the problem and deciding how to implement them. In the step comparing the strategies, the results of implementing different strategies are evaluated and compared to determine through negotiation the best path to the solution. In the following step, devise an algorithm, the most effective algorithm emerging from the previous step is determined, and in the following step, the generated algorithm is coded. Identify and correct an error is a debugging process independent of the other steps to detect errors in the coding of a program. Finally, prepare and code new algorithms, involves revising the algorithm and/or generating new algorithms suitable for the given problem and coding them.

The purpose of the current study was to discover students' preferences for plugged and unplugged activities along with their justifications and also their satisfaction with the activities performed in the programming lessons taught with the PSS model. For this purpose, answer was sought for the following questions:

1) What is the effect of applying the PSS model in programming lessons l on students' satisfaction with the instruction provided?
2) What type(s) of activities do students prefer in programming lessons employing the PSS model and what are the reasons for their preferences?

2. Method

2.1. Research Model

Case study was the qualitative research method used in this study. Qualitative research is a research model which uses non-metric data collection methods such as observations, interviews, and document analysis, and follows procedures for eliciting perceptions and depicting events in a realistic and holistic way in their natural setting (Yıldırım and Şimşek, 2005). It also provides flexibility to the researcher during the design and execution of the research (Silverman, 2013). Among many qualitative research designs, McMillan (2000) describes case study as a method for in-depth exploration of one or more events, settings, programs, social groups, or other interconnected systems. It is a research pattern in which an entity is defined and customized with reference to time and space (Büyüköztürk, Çakmak, Akgün, Karadeniz, and Demirel, 2016). To allow in-depth investigation of different aspects of a particular case, a variety of data sources and collection techniques are used (Cohen, Manion, and Morrison, 2005).

In this research, case study was chosen to achieve the following:

- in-depth content analysis of the data collected from the students related to activities offered by the PSS model,
- better understanding feelings, thoughts and emotions through qualitative methods,
- evaluating the findings in a descriptive way.

2.2. Participants

Participants were 38 sixth grade students (20 girls and 18 boys) attending a public school, when the lesson plans were used in classes. Students were coded from K1 to K38. Students study in a classroom with average achievement in a school with a medium ranking among schools with computer lab. They had not previously used Scratch or received any programming training. The research-related activities were carried out by the teacher in charge of Information Technologies and Software (ITS) course, who had 15 years of professional experience and 5 years of experience in teaching Scratch programming. The researcher personally collected data while visiting the school throughout the implementation stage.

2.3. Data Collection Tools

Data included students' journals (Appendix B), periodic semi-structured interviews responses with a sub-set of students, and their responses to a semi-structured interview form (Appendix A), developed by the researcher to explore students' perceptions and evaluation of the process. While preparing the data collection tools, the researcher consulted with six field specialists (two lecturers in Computer Education and Instructional Technologies, two PhD candidates, and two master's students). Thus, the scope validity of the data collection tools was ensured. Data collection tools were applied to 10 different students (5 females, 5 males) in the sixth grade before the actual application and their opinions were obtained and the necessary arrangements were made, and their opinions were obtained again. In this way, the validity of meaning is also provided.

2.4. Data Collection and Analysis

All students wrote journal entries every week at the end of each week (nine times in total, see Appendix B). Also, the researcher and the teacher jointly conducted five semi-structured interviews with nine students (each student individually) at the juncture between one topic and the next. Interviewees were selected to represent a balance of mathematics and ITS proficiency levels (three low, three medium, three high) and genders (five females and four males). Interviews were audio-recorded, and all students' data were transferred to the computer for content analysis. Participants' informed consent was obtained prior to the interviews, and they participated on an absolutely voluntary basis. All participants' information was kept confidential.

2.5. Validity and Reliability of Study

The qualitative data were analyzed using content analysis methods, which included combining, reducing and interpreting what the participants said and what the researcher witnessed and read (Merriam and Tisdell, 2015).

Audio-recordings of the student interviews were recorded, digitalized, and analyzed which included editing, coding by themes, and supporting the themes with citations and excerpts (Creswell, 2013). In addition, during the interviews the researcher kept detailed notes, paying due diligence to capturing all information. All data were securely saved in case of need to verify the results. Three researchers specializing in the field of Computer Education and Instructional Technologies first analyzed the data independently and then aligned their results. Initially the researchers' analyses were 88% congruent, following which they convened and reached full consensus on the study themes and related areas. The data were confirmed by member-checks by participants when deemed necessary. The researchers played an active role in reporting as well as in analyzing data.

2.6. Implementation Approvals

Because the research involved human subjects and their data, an approval certificate was obtained from the Ethics Committee for Social and Human Sciences of University, documenting conformity with ethical principles of purpose, justification, method and data collection instruments and procedures. The document was then submitted to the Ministry of National Education (MoNE), which granted approval for research in schools. After both approvals were obtained, an information form about the study and consent form were sent to parents of the participant students. The study was conducted with students only after receiving their parents' consent.

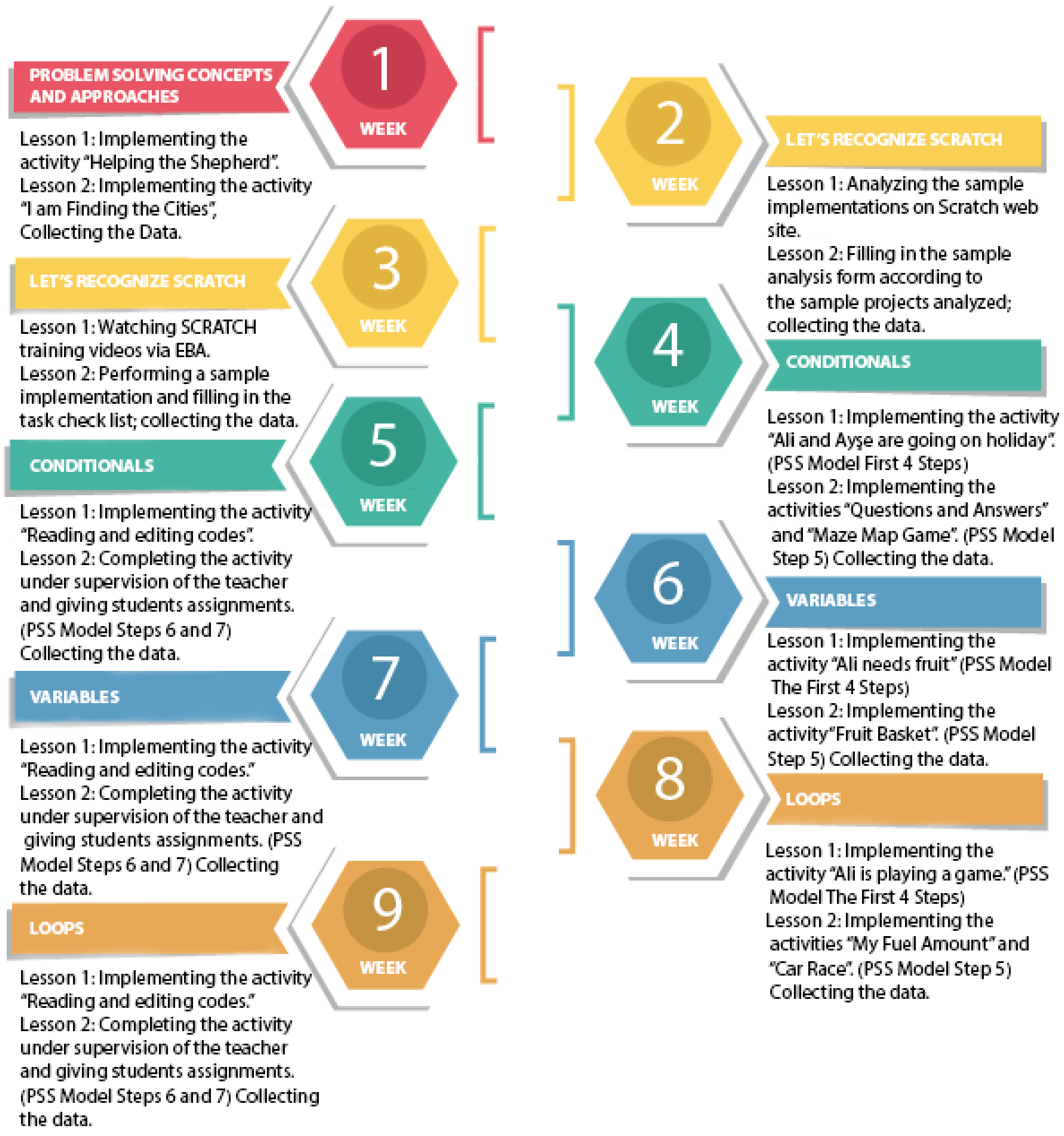
2.7. The Implementation

The research was carried out for 9 weeks (80 minutes each) with 38 students studying in the 6th grade. Each of the students in the class had his/her own computer, which was controlled from a central computer. In the first week, a researcher informed the students about the implementation process and its importance. Within the scope of the seven-step lesson plans, the first four steps were carried out without computers, while the other three steps were plugged. The weekly implementation process is shown in Figure 2.

7 steps of PSS model may not be applied in the same week. This is related to the duration of the lesson. The important thing is to use 7 steps to teach the same subject. In the same week, 7 steps of PSS may not catch up since the lesson time is limited. However, since each step constitutes a meaningful piece, it is not a problem that 7 steps are independent.

As shown in Figure 2, the first week covered Problem-Solving Concepts and Approaches. In the first lesson, the activity, “Helping the Shepherd” (Appendix C) was carried out with an activity sheet providing the scenario of the problem, a story visual, and a space in which to write the solution algorithm using pseudo-code. In the following lesson, the activity “I am finding the cities” (Appendix D) also included an activity sheet for students to analyze and solve a problem. Again, they used the blank area to provide the solution algorithm created through pseudo-code.

During the two classes of week two, the topic “Let's Recognize Scratch” was addressed. In the first lesson, the students studied sample projects on the Scratch web platform, guided by the course teacher to make sure they paid attention to the salient features of the projects. In order to ensure that the students pay attention to the characteristics of the projects they are studying, instructions were given by the course teacher (Appendix E). The next lesson was allocated for students to freely perform activities inspired from the projects examined. In the following week, the previous week's topic was continued. In the first lesson, the students watched Scratch training videos via the Education Informatics Network (EBA, http://www.eba.gov.tr/). In the next class by students completed some easy exercises.



PROBLEM SOLVING CONCEPTS AND APPROACHES

Lesson 1: Implementing the activity "Helping the Shepherd".
Lesson 2: Implementing the activity "I am Finding the Cities", Collecting the Data.

LET'S RECOGNIZE SCRATCH

Lesson 1: Analyzing the sample implementations on Scratch web site.
Lesson 2: Filling in the sample analysis form according to the sample projects analyzed; collecting the data.

LET'S RECOGNIZE SCRATCH

Lesson 1: Watching SCRATCH training videos via EBA.
Lesson 2: Performing a sample implementation and filling in the task check list; collecting the data.

CONDITIONALS

Lesson 1: Implementing the activity "Ali and Ayşe are going on holiday". (PSS Model First 4 Steps)
Lesson 2: Implementing the activities "Questions and Answers" and "Maze Map Game". (PSS Model Step 5) Collecting the data.

CONDITIONALS

Lesson 1: Implementing the activity "Reading and editing codes".
Lesson 2: Completing the activity under supervision of the teacher and giving students assignments. (PSS Model Steps 6 and 7) Collecting the data.

VARIABLES

Lesson 1: Implementing the activity "Ali needs fruit" (PSS Model The First 4 Steps)
Lesson 2: Implementing the activity "Fruit Basket". (PSS Model Step 5) Collecting the data.

VARIABLES

Lesson 1: Implementing the activity "Reading and editing codes."
Lesson 2: Completing the activity under supervision of the teacher and giving students assignments. (PSS Model Steps 6 and 7) Collecting the data.

LOOPS

Lesson 1: Implementing the activity "Ali is playing a game." (PSS Model The First 4 Steps)
Lesson 2: Implementing the activities "My Fuel Amount" and "Car Race". (PSS Model Step 5) Collecting the data.

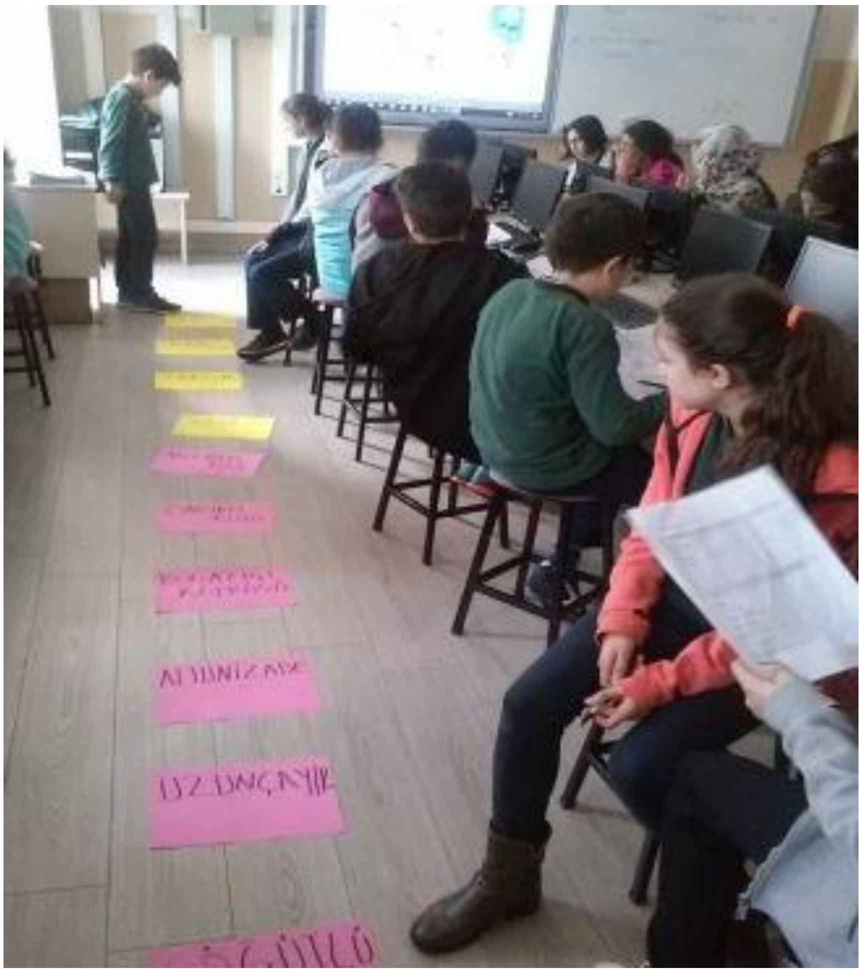
LOOPS

Lesson 1: Implementing the activity "Reading and editing codes."
Lesson 2: Completing the activity under supervision of the teacher and giving students assignments. (PSS Model Steps 6 and 7) Collecting the data.

Figure 2. The Implementation Processes

In week four the "Conditionals" were taught, again in two lessons beginning with the activity, "Ali and Ayşe are going on holiday," (Appendix F) in which the first four steps of the PSS model were applied. For "Understand the problem", the students answered questions about the problem situation using the Problem Acknowledgement Table (PAT) form (Appendix G). For "Devise a Plan", they used the Strategy Devise Form (SDF) (Appendix H) to decide on the most appropriate strategy and solved the problem.

For the following step, “Compare the Strategies,” the teacher worked with the students to identify the most appropriate strategy from those generated in the previous step. As the next step, the students wrote the algorithm for their own plan using pseudo-code. Later, some students voluntarily dramatized their algorithm using materials provided by the teacher (Picture 1).



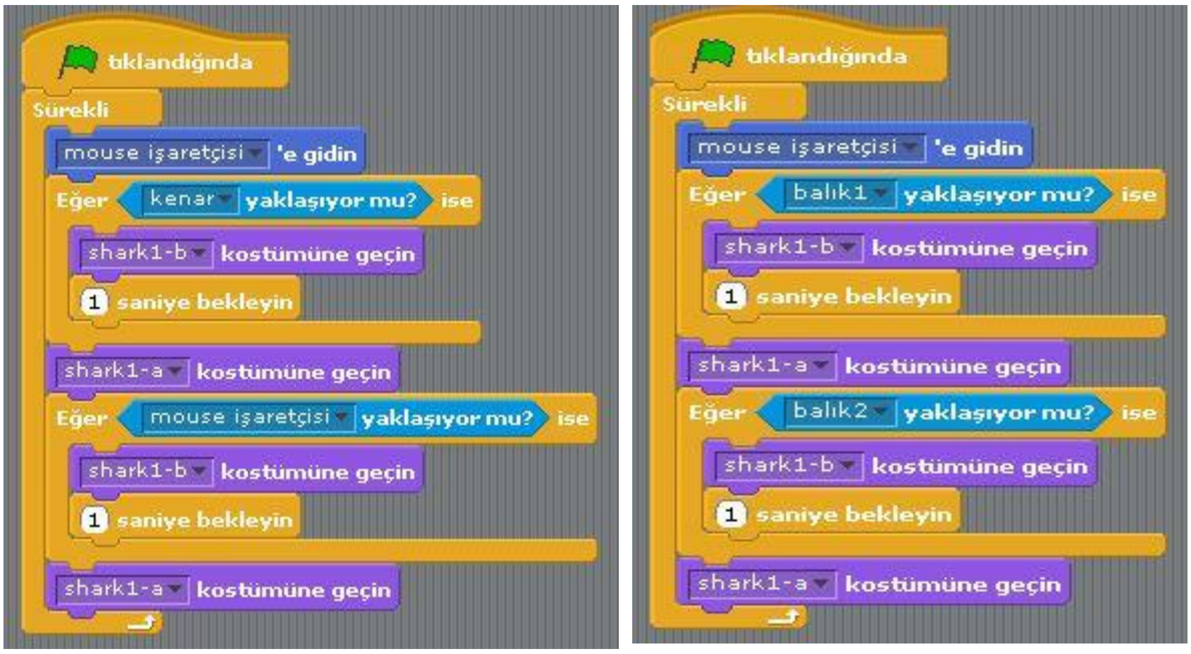
Picture 1. In Class Application

The next lesson began with a Scratch computer application, “Questions and Answers,” followed by “Maze Map” as the fifth and last step of the model in this phase.

Picture 2. “Maze Map” Scratch Activity

In the fifth week, the topic from the previous week was continued. In lesson one, the teacher presented a Scratch application with an incorrect code structure which the students studied to detect and correct the errors.

Incorrect Correct



```
tıklandığında
Sürekli
    mouse işaretçisi 'e gidin
    Eğer kenar yaklaşıyor mu? ise
        shark1-b kostümüne geçin
        1 saniye bekleyin
    shark1-a kostümüne geçin
    Eğer mouse işaretçisi yaklaşıyor mu? ise
        shark1-b kostümüne geçin
        1 saniye bekleyin
    shark1-a kostümüne geçin
```

```
tıklandığında
Sürekli
    mouse işaretçisi 'e gidin
    Eğer balık1 yaklaşıyor mu? ise
        shark1-b kostümüne geçin
        1 saniye bekleyin
    shark1-a kostümüne geçin
    Eğer balık2 yaklaşıyor mu? ise
        shark1-b kostümüne geçin
        1 saniye bekleyin
    shark1-a kostümüne geçin
```

Figure 3. Incorrect and Correct Code

During the next class, the students and the teacher finished the activity by modifying the faulty code structure together. After that, the teacher sent the students an assignment via the EBA system, which corresponds to steps 6 and 7 of the PSS Model.

In the sixth week, the topic of Variables was discussed. The first hour was used for an activity titled “Ali needs fruit,” in which students completed the first four steps of the model. The next lesson followed for the implementation of Step 5 of the PSS Model in the context of “Fruit Basket” as a Scratch application. The seventh week was a continuation of the previous week. The procedure was put into practice as described under “Conditionals” in the second week.

During week eight, “Loops” was discussed. The first lesson, which featured the activity “Ali playing a game,” was completed after the first four steps of the PSS model as described in the fourth week had been implemented. The following class started with the Scratch computer application “My fuel amount,” followed by “Auto racing,” finalizing stage 5 of the PSS model. In the ninth week of the implementation, the issue of “Loops” continued. The application in the ninth week was carried out in the same way as the application in the fifth week, and the whole application process was completed.

To recapitulate the scope of the PSS model, as part of the first step, understand the problem, the students answered the questions formed in accordance with the activity in the lesson plan. In this step, the students were asked to distinguish the necessary and unnecessary information in the problem presentation. During the second step, devise a plan, the students sought the best among multiple paths to reach a solution to a problem. In step three, comparing the strategies, the teacher constructed the optimum result by examining all the students’ proposed solutions together. Step four, code the algorithm, was completed in two parts. First, the students represented their solutions as algorithms written in pseudo-code. Later, volunteers acted out their algorithms in dramatic performances for the class. All of these four steps could fit into one class hour. During the next step, code the algorithm, the students undertook the task of transferring the previous activity onto Scratch. This step took about two hours.

In step six, identify and correct an error in a different code, the students were given Scratch applications with missing or incorrect code structures independent from the activity in the lesson plan and required to identify and eliminate the errors. After they completed the task individually, the teacher collaborated with the students to complete it on the smart board. This stage took one lesson period.

At the final stage, prepare and code new algorithms, the teacher mailed assignments to the students via EBA with specified criteria. In this assignment, the students were asked to first write the algorithm that met the criteria given and then to encode the algorithm in the programming environment. An overview and use of lesson plans are given in Table 1.

Table 1. Summary Table for Lesson Plans

Topic	Attainment	Activity	PSS Step	Type	Individual or Group	Duration	Week
Problem Solving Concepts and Approaches	• Analyzes a problem. • Describes the importance of generating authentic solutions for problems. • Devises an algorithm for solving the problem. • Discusses the problem-solving process and basic concepts.	Helping the Shepherd	-	Unplugged	Individual or Group	40 mins	1
		Weaver Birds	-	Unplugged	Individual or Group	40 mins	
Let's Recognize the Programming Environment	• Adds characters to the screen. • Changes the screen background. • Gives movement to the characters. • Gives movement by changing character costume. • Recognizes the interface and features of the block-based programming tool. • Describes the function of a program in the block-based programming tool.	Leisure Time	-	Plugged	Individual	160 mins	2-3
Conditionals	• Analyzes a problem. • Describes the importance of generating authentic solutions for problems. • Discusses the problem-solving process and basic concepts. • Devises an algorithm for solving the problem. • Analyzes different algorithms and chooses the fastest and most accurate solution. • Generalizes the solution to similar problems. • Tests the solution of an algorithm. • Improves and edits a program in the block-based programming tool against given criteria. • Creates programs which include the linear logic structure. • Creates programs which include the condition structure. • Creates programs which include multiple condition structures. • Corrects a program in the block-based programming tool. • Tests and corrects programs involving the linear logic structure. • Tests and corrects programs involving the condition structure. • Tests and corrects programs involving the multiple condition structure.	"Ali and Ayşe are going on holiday"	1-4	Unplugged	Individual or Group	35 mins	4-5
		"Colourful Steps"	4	Unplugged	Group	10 mins	
		"Questions and Answers" "Maze map game"	5	Plugged	Individual	95 mins	
		"Reading and editing codes"	6	Plugged	Individual	20 mins	

	• Divides a problem into sub-problems.						
Variables	• Analyzes a problem. • Describes the importance of generating authentic solutions for problems. • Discusses the problem-solving process and basic concepts. • Devises an algorithm for solving the problem. • Analyzes different algorithms and chooses the fastest and most accurate solution. • Generalizes the solution to similar problems. • Classifies data by type. • Tests the solution of an algorithm. • Uses the constants and variables to solve the given problem. • Creates programs which include the linear logic structure. • Improves and edits a program in the block-based programming tool against given criteria. • Corrects a program in the block-based programming tool. • Tests and corrects programs involving the linear logic structure. • Divides a problem into sub-problems.	"Buying fruit at greengrocer's"	1-4	Unplugged	Individual or Group	30 mins	6-7
		"I choose fruit"	4	Unplugged	Group	10 mins	
		"My fruit basket"	5	Plugged	Individual	95 mins	
		"Reading and editing codes"	6	Plugged	Individual	20 mins	
Loops	• Analyzes a problem. • Describes the importance of generating authentic solutions for problems. • Discusses the problem-solving process and basic concepts. • Devises an algorithm for solving the problem. • Analyzes different algorithms and chooses the fastest and most accurate solution. • Generalizes the solution to similar problems. • Tests the solution of an algorithm. • Improves and edits a program in the block-based programming tool against given criteria. • Creates programs which include the linear logic structure. • Creates programs which include the loop structure. • Chooses the most appropriate condition structures to implement an algorithm. • Generates solutions for complex problems by using different programming structures. • Tests and corrects programs involving the loop structure.	"Ali playing a game"	1-4	Unplugged	Individual or Group	30 mins	8-9
		"Auto racing" in-class animation	4	Unplugged	Group	10 mins	
		"My fuel amount"	5	Plugged	Individual	95 mins	
		"Auto racing" Application					
		"Reading and editing codes" Application	6	Plugged	Individual	20 mins	

	• Determines the relation between mathematics and computer sciences. • Devises an authentic project which involves all programming structures. • Divides a problem into sub-problems.

All of the activities listed in Table 1 were developed in accordance with the PSS model. Şahin's (2018) thesis provided a reference for developing the lesson plans and activities used in the implementation stage and for testing the suitability of the activities for the students' cognitive levels, determining lesson durations, and coping with overcrowded classrooms. Also, the lesson plans were prepared in accordance with the Directive of the Ministry of National Education on Planned Implementation of Education and Instruction Activities (2003).

3. Results

3.1. Effects of programming lessons taught with the PSS model on students' satisfaction

To answer the first research question (What is the effect of applying the PSS model in programming lessons on students' satisfaction with the instruction provided), the researchers analyzed the data collected through students' semi-structured interviews and journals using content analysis, including such topics as "Conditionals," "Variables," and "Loops". Table 2 presents students' reflections on the teaching of "Conditionals".

Table 2. Students' Reflections on Learning about Conditionals through PSS

Students' Reflections on Learning about Conditionals through PSS	**f (n=29)**
Lesson was fun.	19
I'd like lessons in this way.	5
I find the lesson effective or efficient. I could understand the topic very well.	16
I find the teaching of the lesson a bit boring.	2

These highly positive responses are supported by the following excerpts:
Plugged and Unplugged Activities

> *"I thought it was going to be boring while I was starting. But when I started the class, I started to have fun slowly. I think it was a really fun lesson I enjoyed all the activities; I hope we will go on like this." (K8)*

> *"I think the teaching of the lesson was very nice. Thanks to the activities given to us, the topic was understood very well, and the process was fun." (K11)*

> *"I enjoyed it a lot and had fun. I think it is more efficient for us to be taught in this way. I want it to continue like this." (K18)*

Plugged Activities

> *"The plugged activity was far more fun." (K10)*

> *"It gets more fun on the computer." (K4)*

Unplugged Activities

> *"I had difficulty understanding the problem in the activity Ali and Ayşe. I think it was difficult, so I was bored."(K15)*

> *"I could understand better the problem and the solution in this way. I realized that I can understand problems more effectively by playing activities and games." (K6)*
>
> *"It helped me better understand the parts I didn't understand. It was a fun activity." (K20)*
>
> *"I could better understand the activity by seeing, and I believe it will stay longer in my mind in terms of time." (K29)*
>
> *"I think we have visualized the problems thanks to that activity. It also made the lesson more fun." (K31)*

As can be understood from Table 2 and the excerpts above, the students particularly enjoyed learning through the PSS model in connection with "Conditionals." Although some students found some of the relevant activities difficult, they did not make negative comments about the overall instructional process. Only two students said that they were bored by the lesson because they found the problem in unplugged activities difficult to understand. Table 3 presents students' views regarding the teaching of the topic "Variables".

Table 3. Students' Reflections on Learning of Variables through PSS

Students' Reflections on Learning of Variables through PSS	**f (n=30)**
Lesson was fun.	15
I liked that the lesson was taught in this way.	17
I learnt to devise a strategy.	3
It was a bit boring.	4

Table 3 demonstrates that students' perceptions of the lesson on variables were highly positive, as exemplified in the following excerpts:

Plugged and Unplugged Activities

> *"The animation was also nice. It was also easy to transfer this to the Scratch environment." (K31)*

Plugged Activities

> *"They were very nice and logical activities. I enjoy such activities." (K7)*
>
> *"It was enjoyable and instructive. Maybe it could have been a little harder. Other than that, I liked the faulty code activity the most." (K12)*

Unplugged Activities

> *"I had fun doing it, the activity was nice and instructive, but it was too easy." (K8)*
>
> *"I couldn't find any negative sides. As for positive, I learned strategy." (K19)*
>
> *"I had a lot of fun; I learned variables better and clearly." (K5)*

> *"It was nice. The activity was very easy for me, so I was a little bored." (K3)*
>
> *"I had so much fun. It was a lot of fun to act the fruits and Ali. My thinking ability has improved." (K7)*
>
> *"That was so fun. This game helped me understand variables." (K12)*

As indicated in Table 3 and the excerpts, , "Variables" was enjoyed the most as a result of teaching with the PSS model. However, four students stated that they were bored because they found the problem too easy.
Students' reflections on the topic "Loops" are given in Table 4.

Table 4. Students' Reflections on Learning Loops through PSS

Students' Reflections on Learning Loops through PSS	**f (n=24)**	**%**
Lesson was fun.	11	45.83%
I liked that the lesson was taught in this way.	13	54.16%
I could better understand the topic.	2	8.33%
It improved my skills of understanding the problem, devising a strategy.	2	8.33%

Table 4 shows that the students reported substantially positive views regarding the use of the PSS for learning loops. Some of the students' reflections on the process are excerpted below:

Plugged Activities

> *"I had more fun on the computer." (K23)*
>
> *"Overall, it was nice. I liked the coding part on Scratch the most. The negative side was that there were no different ideas. Other than that, it was nice." (K8)*
>
> *"It has no negative side. Its positive sides are about our editing the incorrect codes in the activity. In this way, I could understand much better what each operation is for." (K13)*

Unplugged Activities

> *"I found it good. It didn't have a negative side. As for the positive side, it helped me devise a strategy and understand better." (K4)*
>
> *"I like it. The positive sides have improved my ability to understand the problem, and the step of creating the algorithm reminded me of the algorithms we have recently created. I don't think there's a negative side." (K6)*
>
> *"The positive sides are about improving our strategy strength. They have no negative sides." (K18)*
>
> *"I found it fun and it made me understand the topic better." (K12)*

As reported in Table 4 and the excerpts, the participants found learning “Loops” using the PSS model to be a pleasant and entertaining experience. Notably, no students reported negative opinions about the course of the lesson.

3.2. Students’ activity type preferences in programming lessons taught with the PSS model along with their *justifications*

Students’ responses to “What types of activities do you prefer, plugged only, unplugged only, or both?” are summarized across three topics in Table 5:

Table 5. Students’ Preferences for Activity Type

Topic	**Activity Type**	**n**	**f**	**%**
Conditionals	Plugged	30	5	16.66
	Unplugged		5	16.66
	Both		20	66.66
Variables	Plugged	29	7	24.13
	Unplugged		2	6.89
	Both		20	68.96
Loops	Plugged	24	6	25
	Unplugged		4	16.66
	Both		14	58.33

Table 5 shows that although both types of activities were predominantly preferred, some students opted for only plugged or unplugged activities.

In order to find out the reasons for their preferences, the students were asked *“Why would you prefer a particular type of activity (plugged, unplugged, both) for teaching of the lessons?”* Table 6 summarizes the students’ reasons for their preferences along with supporting excerpts from their answers.

Table 6. Students’ Reasons for Activity Preferences

Type of Activity	**Reason for Preference**	**Student’s Comments**
Unplugged Activities	(A) • It presents a problem status with more than one path of solution. • The problem has an analogical design. • The solution of the problem offers “Devise a Plan” step. • It types the algorithm for the best plan available. • The plan can be performed with drama activities.	Conditionals – Colourful steps game- *“I could understand the problem and the solution better in this way. I realized that I can understand problems more effectively by activities and playing games.” (K6)* Conditionals - Colourful steps game- *“It helped me better understand the parts I didn't understand. It was a fun activity.” (K20)* Conditionals - Colourful steps game- *“I could better understand the activity by seeing, and I believe it will stay longer in my mind in terms of time.” (K29)*

		Conditionals- Colourful steps game- *"I think we have visualized the problems thanks to that activity. It also made the lesson more fun." (K31)* Variables- Fruit basket- *"I had a lot of fun; I learned variables better and clearly." (K5)* Variables- Fruit basket- *"I had so much fun. It was a lot of fun to act the fruits and Ali. What it has brought me is the convenience of doing this application in the Scratch program. My thinking ability has improved." (K7)* Variables- Fruit baskets- *"That was so fun. This game helped me understand variables." (K12)* Loops- *"Yes, it was fun. It better taught me the activity." (K4)* Loops- *"I found it fun and it made me better understand the topic." (K12)*
Plugged Activities	(B) • The visual programming tool Scratch is used. • It corrects the erroneous code structure. • There are interesting activities.	Conditionals - *"The plugged activity was more fun." (K10)* Conditionals - *"It becomes more fun on the computer." (K4)* Variables- *"They were very nice and logical activities. I enjoy such activities." (K7)* Variables - *"It was enjoyable and instructive. Maybe it could have been a little harder. Other than that, I liked the faulty code activity the most." (K12)* Loops- *"Overall, it was nice. I liked the coding part on Scratch the most. The negative side was that there were no different ideas. Other than that, it was nice." (K8)* Loops - *"I had more fun on the computer." (K23)*
Plugged and Unplugged Activities	(A)+(B)	Conditionals- *"The computer activities were as fun as the activities we did in the classroom." (K27)* Conditionals- *"Both were fun (with and without a computer), I had a lot of fun." (K28)* Variables- *"The activities were fun to me. I had a lot of fun." (K7)* Variables- *"They were very amusing and instructive activities." (K11)* Loops- *"I think in-class activities are very nice and fun." (K5)* Loops- *"The activities were very fun. I had fun." (K6)*

The results in Table 6 were based on themes obtained from the content analysis. As stated in the right column, some of the students appreciated plugged activities more, while some others favoured unplugged activities in a environment. Some others stated that they liked all the activities in the classroom (including plugged and unplugged) and found them amusing.

4. Discussion and Conclusion

This case study was carried out to shed light on students' satisfaction with the plugged and unplugged activities provided in programming lessons using the PSS model and their preferences for activity type along with their reasons. As a result, it was found that some participants preferred plugged activities and others, though not as many, preferred unplugged activities, but most frequent response was that they were positive about both. Furthermore, the study shows that neither plugged nor unplugged activities affect all students in the same way. It can be concluded that our model holds the potential to meet the needs of all students since it encompasses both unplugged and computer activities in the first four and the last three steps, respectively. On the whole, the students' views regarding the learning process with the PSS model using such terms as fun, liking, effective, and efficient indicate positive effects of the implementation on their interest and motivation.

In the literature, it is reported that students' satisfaction is bound to multiple factors. Sáez-López et al. (2016), in a study on fifth and sixth graders, found out that students were pleased to work with Scratch visual programming tools, enjoyed the activities, found the Scratch programming environment amusing, and felt motivated. Calder (2010) in his study involving sixth graders found that Scratch programming is motivating and interesting for learners. In their research with nine- and ten-year-old students, Taylor, Harlow, and Forret's (2010) also found that plugged activities with the Scratch programming tool were interesting for students. These results are consistent with our finding that students learning programming through the PSS model showed a greater preference for plugged activities more. Using Scratch as a visual and block-based programming tool appeared to increase their interest in and satisfaction with plugged activities and account the findings in the present study. Likewise, previous researchers point out that the use of visual programming tools increases positive reactions such as interest, satisfaction, and amusement (Kalelioğlu, 2015; Kalelioğlu and Gülbahar, 2014; Kelleher et al., 2007; Malan and Leitner, 2007; Pinto and Escudeiro, 2014; Wu et al., 2010).

Previous studies have also established that students enjoy and are motivated by learning through unplugged activities. For instance, Futschek and Moschitz (2011) included entertaining activities at different levels in an unplugged environment for teaching students the basic concepts of algorithm and developing their algorithmic thinking skills prior to switching to a programming environment. In the same vein, Tsalapatas et al. (2012) found that there was an increase in primary students' satisfaction with instruction to develop algorithmic thinking skills when they were offered activities and drama games in an unplugged environment. Moreover, it was noted that including drama activities in ITS lessons beneficial, more amusing, and increased satisfaction with the lessons and beneficial and entertaining (Karaosmanoğlu and Adıgüzel, 2017). Some researchers further defended role playing as a way to lure students to take an active part in the lesson and successfully learn by doing and interacting with peers, which supports collaborative learning (Atalay, 2010).

In addition, the literature suggests that analogies increase the interest, curiosity, and satisfaction of learners (Keller, 1983 as cited by Seyhan, 2015, p.18; Sarıoğlu and Kartal, 2017). Such findings are in congruence with our finding that students learning programming through PSS tend to prefer unplugged activities. It is thought the students in our case favoured unplugged activities mainly because such steps have an analogical basis, drama activities are entertaining, necessary and unnecessary data are separated in the step of understanding the problem and devising a strategy for a solution to the problem increased students’ interest and satisfaction.

Gomes, Falcão and Tedesco (2018) investigated how children can be taught programming concepts with digital games. A primary finding was that students had difficulty understanding the topic of loops and needed unplugged activities to reinforce their understanding of this concept, leading the researchers to recommend including in-class activities besides plugged activities in programming instruction.

Tsarava et al. (2017) conducted a curriculum development study to improve the computational thinking skills of primary school students. In the target curriculum, concepts common to computational thinking, computers, and programming were highlighted. While developing the curriculum, they intended to promote the cognitive operations that underlie computational thinking skills by approaching programming from a cognitive perspective. The curriculum included both plugged and unplugged game-based activities aiming to motivate students.

Leifheit, Jabs, Ninaus, Moeller and Ostermann (2018) also assessed the appropriateness of a game-based approach to teaching programming in primary school to develop students’ computational thinking skills such as abstraction, generalization, algorithms, and ability to systematically solve complex problems. In the lessons, algorithm concepts were covered first, and then game-based learning materials were used to introduce the other programming concepts using tangible objects rather than direct coding. In the following lessons, students' learning was reinforced with plugged programming exercises. It was concluded that the game-based, non-digital activity approach had a positive effect on teaching computational thinking and related skills.

Weigend (2014) reports that plugged and unplugged activities together can complement each other and hold students’ interest and enthusiasm. Futschek and Moschitz (2011) found that using unplugged exercises at the basic level of teaching programming resulted in students’ higher levels of satisfaction for programming on the computer as they experience increasing levels of programming, and even those who had no interest in programming at early stages developed positive attitudes toward programming. Giannakos et al. (2013) pointed out that students' interest in and satisfaction with computer sciences and programming increased as a result of engaging in both unplugged and plugged activities, especially for female students.

Thus the literature supports the effectiveness of a combination of plugged and unplugged activities for enhancing both conceptual understanding of programming and computational thinking skills. It has been found that unplugged tasks support understanding of a problem, devising and implementing a strategy, creating an algorithm, and dramatizing activities while plugged activities involving the digital media (Scratch, code.org, Lightbot etc.) are motivating.

In this study it was found out that the type of activity had a significant effect on students' satisfaction in the teaching programming as a means of developing cognitive skills. Students’ satisfaction increased when they were engaged in programming lessons in their preferred activity type. Thus, combining the two approaches can help students overcome the difficulties they encounter face in programming teaching. Therefore the PSS model, which includes both types of activity, is highly likely to support and appeal to all learners in a programming course.

5. Recommendations

For the teaching programming, we recommend that plugged activities should be included in order to

- Determine students' levels of interest in using a computer before starting teaching programming for increase satisfaction with programming, and
- Find out students' views regarding their preferred learning environment.

We recommend unplugged activities that

- include a problem situation that may have more than one solution.
- are formulated analogically.
- Include role-playing or other drama methods.

References

Ala-Mutka, K. (2004). Problems in learning and teaching programming. *Institute of Soft ware Systems, Tampere University of Technology*. Retrieved June 15, 2018 from https://www.cs.tut.fi/~edge/literature_study.pdf

Arabacıoğlu, T., Bülbül, H. İ., & Filiz, A. (2007, January). A New Approach to Computer Programming Teaching. *In Proceedings of the 9th Academic Informatics Conference*, Dumlupınar University, Kütahya, Turkey.

Aşkar, P., & Davenport, D. (2009). An investigation of factors related to self-efficacy for Java programming among engineering students. TOJET: *The Turkish Online Journal of Educational Technology,* 8(1), 26-27.

Asad, K., Tibi, M., & Raiyn, J. (2016). Primary school pupils' attitudes toward learning programming through visual interactive environments. *World journal of education,* 6(5), 20. doi:10.5430/wje.v6n5p20

Atalay, O.(2010). *Effect of the use of drama method on the 5th grade students success in information technologies course*. Unpublished Master Thesis. Gazi University, Education Sciences Institute, Ankara.

Brennan, K., & Resnick, M. (2012, April). 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).

Brown, Q., Mongan, W., Kusic, D., Garbarine, E., Fromm, E., & Fontecchio, A. (2013). *Computer aided instruction as a vehicle for problem solving: Scratch programming environment in the middle years classroom*. Retrieved September 22, 2017 from http://www.pages.drexel.edu/~dmk25/ASEE_08.pdf

Büyüköztürk, Ş., Çakmak, E. K., Akgün, Ö. E., Karadeniz, Ş., & Demirel, F. (2016). *Bilimsel araştırma yöntemleri*. Ankara: Pegem Akademi.

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *ACM SIGCSE Bulletin, 33*(3), 49-52. doi: 10.1145/377435.377467

Calder, N. (2010). Using Scratch: An integrated problem-solving approach to mathematical thinking. *Australian Primary Mathematics Classroom, 15*(4), 9-14.

Cohen, L., Manion, L., & Morrison, K. (2005). *Research methods in education* (5th ed.). London: Routledge Falmer.

Coull, N. J., & Duncan, I. M. (2011). Emergent requirements for supporting introductory programming. *Innovation in Teaching and Learning in Information and Computer Sciences*, 10(1), 78-85. doi: 10.11120/ital.2011.10010078

Creswell, J. W. (2013). Steps in conducting a scholarly mixed methods study. DBER Speaker Series. 48.

Çatlak, Ş., Tekdal, M., & Baz, F. Ç. (2015). The Status of Teaching Programming with Scratch: A Document Review Work. *Journal of Instructional Technologies & Teacher Education*, *4*(3), 13-25.

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. doi: 10.1080/15391523.2014.888272

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

Erümit A. K., Karal H., Şahin G., Aksoy D.A., Aksoy A., & Benzer A.İ. (2019). A Model Suggested for Programming Teaching: Programming in Seven Steps. *Education & Science*. 44(197), 155-183. doi: 10.15390/EB.2018.7678

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87-97. doi: 10.1016/j.compedu.2012.11.016

Fesakis, G., & Serafeim, K. (2009). Influence of the familiarization with scratch on future teachers' opinions and attitudes about programming and ICT in education. *ACM SIGCSE Bulletin*, 41(3), 258-262. doi: 10.1145/1562877.1562957

Futschek, G., & Moschits, J. (2010). *Developing algorithmic thinking by inventing and playing algorithms*. Retrieved December 9, 2016 from https://publik.tuwien.ac.at/ files/PubDat_187461.pdf

Futschek, G. (2006). Algorithmic thinking: The key for understanding computer science. In: Mittermeir, R. (Ed.), Informatics Education The Bridge between Using and Understanding Computers. Vol. 4226 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, ss. 159 – 168.

Futschek, G., & Moschitz, J. (2011, October). Learning algorithmic thinking with tangible objects eases transition to computer programming. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, 155-164. Springer, Berlin, Heidelberg.

Giannakos, M. N., Jaccheri, L., & Proto, R. (2013, April). Teaching computer science to young children through creativity: Lessons learned from the case of Norway. *In Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, Heerlen.

Ginat, D. (2004). On novice loop boundaries and range conceptions. *Computer Science Education*, 14(3), 165-181. doi: 10.1080/0899340042000302709

Gomes, T. C. S., Falcão, T. P., & Tedesco, P. C. D. A. R. (2018). Exploring an approach based on digital games for teaching programming concepts to young children. *International journal of child-computer interaction*, *16*, 77-84. doi: 10.1016/j.ijcci.2017.12.005

Gomes, A., & Mendes, A. J. (2007, September). Learning to program - difficulties and solutions. *In International Conference on Engineering Education*, Coimbra.

Grover, S., Pea, R., & Cooper, S. (2016, March). Factors influencing computer science learning in middle school. *In Proceedings of the 47th ACM technical symposium on computing science education,* New York. doi: 10.1145/2839509.2844564

Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers & Education, 54*(4), 1127-1136. doi: 10.1016/j.compedu.2009.10.020

Helminen, J., & Malmi, L. (2010, October). Jype-a program visualization and programming exercise tool for Python. *In Proceedings of the SOFTVIS '10 Proceedings of the 5th international symposium on Software visualization,* 153-162, Utah, USA. doi: 10.1145/1879211.1879234

Hermans, F., & Aivaloglou, E. (2017, November). To Scratch or not to Scratch?: A controlled experiment comparing plugged first and unplugged first programming lessons. *In Proceedings of WiPSCE '17 the 12th Workshop on Primary and Secondary Computing Education*, 49-56. doi: 10.1145/3137065.3137072

Kafai, Y. B., & Q. Burke. (2014). Connected code: Why children need to learn programming. MIT Press.

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

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior, 52,* 200-210. doi: 10.1016/j.chb.2015.05.047

Karaosmanoğlu, G., & Adıgüzel, Ö. (2017). Effects of Creative Drama Method On Students Academic Achievements In Ict Lessons Of Sixth Grades. *Elementary Education Online, 16*(2), 693-712.

Kelleher, C., Pausch, R., & Kiesler, S. (2007, April). Storytelling Alice motivates middle school girls to learn computer programming. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM*, 1455-1464. doi: 10.1145/1240624.1240844

Klopfer, E., & Yoon, S. (2005). Developing games and simulations for today and tomorrow's tech savvy youth. *TechTrends*, 49(3), 33-41. doi: 10.1007/BF02763645

Kobsiripat, W. (2015). Effects of the media to promote the scratch programming capabilities creativity of elementary school students. *Procedia-Social and Behavioral Sciences*, 174, 227- 232. doi: 10.1016/j.sbspro.2015.01.651

Kotsopoulos, D., Floyd, L. M., Khan, S. S., Namukasa, I. K., Somanath, S., Weber, J. L., & Yiu, C. (2017). A Pedagogical Framework for Computational Thinking. *Digital Experiences in Mathematics Education.* 3(2), 154-171. doi: 10.1007/s40751-017-0031-2

Lahtinen, E., Ala-Mutka, K., & Jarvinen, H. (2005) A Study of Difficulties of Novice Programmers. In Acm Sigcse Bulletin, ACM, 37(3), 14-18. doi: 10.1145/1067445.1067453

Lai, C. S., & Lai, M. H. (2012, June). Using computer programming to enhance science learning for 5th graders in Taipei. *International Symposium on computer, consumer and control,* 146-148. Taiwan. IEEE. doi: 10.1109/IS3C.2012.45

Lai, A., & Yang, S. (2011, September). The learning effect of visualized programming learning on 6th graders' problem solving and logical reasoning abilities. *In Electrical and Control Engineering (ICECE), 2011 International Conference on, IEEE*, 6940-6944. doi: 10.1109/ICECENG.2011.6056908

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

Leifheit, L., Jabs, J., Ninaus, M., Moeller, K., & Ostermann, K. (2018, October). Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School. *In ECGBL 2018 12th European Conference on Game-Based Learning* (p. 344). Academic Conferences and publishing limited.

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, 39(1), 223-227. doi: 10.1145/1227310.1227388

Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*, *16*(3), 211-227. doi: 10.1080/08993400600912384

McMillan, J. H. (2000). *Educational research: Fundamentals for the consumer* (3th ed.). New York: Longman.

Merriam, S. B., & Tisdell, E. J. (2015). *Qualitative research: A guide to design and implementation*. San Francisco: John Wiley & Sons.

Milková, E., & Hulkova, A. (2013). Algorithmic and logical thinking development: base of programming skills. *WSEAS Transactions on Computers*, *12*(2), 41-51.

Naharro-Berrocal, F., Pareja-Flores, C., Urquiza-Fuentes, J., & Velazquez-Iturbide, J. A. (2002). Approaches to comprehension-preserving graphical reduction of program visualizations. In Proceedings of the 2002 ACM symposium on Applied computing, ACM, 771-777. doi: 10.1145/508791.508941

Özmen, B., & Altun, A. (2014). Undergraduate students' experiences in programming: difficulties and obstacles. *Turkish Online Journal of Qualitative Inquiry*, 5(3), 1-27. doi: 10.17569/tojqi.20328

Pinto, A., & Escudeiro, P. (2014, June). The use of scratch for the development of 21st century learning skills in ICT. Information Systems and Technologies, *9th Iberian Conference*, (pp. 1-4). IEEE. doi: 10.1109/CISTI.2014.6877061

Qian, Y., & Lehman, J. D. (2016). Correlates of success in introductory programming: A study with middle school students. *Journal of Education and Learning*, *5*(2), 73. doi: :10.5539/jel.v5n2p73

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

Sarıoğlu, T., & Kartal, G. (2017). Drama as Method—An Alternative in Teaching Information and Communication Technologies? *Elementary Education Online*, *16*(1), 366-376.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129-141. doi: 10.1016/j.compedu.2016.03.003

Schwartz J., Stagner J., & Morrison, W. (2006). Kid's programming language (KPL). *In ACM SIGGRAPH 2006 Educators program, ACM.* doi: 10.1145/1179295.1179348

Seppälä, O., Malmi, L., & Korhonen, A. (2006). Observations on student misconceptions—A case study of the Build–Heap Algorithm. *Computer Science Education*, 16(3), 241-255. doi: 10.1080/08993400600913523

Seyhan, H. G. (2015). Okul öncesi fen eğitiminde analoji kullanımının önemi ve analoji örnekleri. *Cumhuriyet International Journal of Education*, *4*(2), 15-28.

Shih, I.-J. (2014). *The effect of scratch programming on the seventh graders' mathematics abilities and problem solving attitudes.* Unpublished master's thesis. Taipei University, Taiwan.

Silverman, D. (2013). *Doing qualitative research: A practical handbook*. London: SAGE Publications.

Stolee, K. T., & Fristoe, T. (2011, March). Expressing computer science concepts through Kodu game lab. *In Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 99-104. Dallas, USA. doi: 10.1145/1953163.1953197

Şahin, G. (2018). *A methodology for teaching programming at middle school level.* Unpublished Master Thesis. Karadeniz Technical University, Education Sciences Institute, Trabzon.

Taylor, M., Harlow, A., & Forret, M. (2010). Using a computer programming environment and an interactive whiteboard to investigate Some Mathematical Thinking. Procedia Social and Behavioral Sciences, 8 (2010), 561-570. doi: 10.1016/j.sbspro.2010.12.078

Tsalapatas, H., Heidmann, O., Alimisi, R., & Houstis, E. (2012). Game-based programming towards developing algorithmic thinking skills in primary education. *Scientific Bulletin of the" Petru Maior" University of Targu Mures*, *9*(1), 56-63.

Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017, October). Training computational thinking: Game-based unplugged and plugged-in activities in primary school. In *European Conference on Games Based Learning*, 687-695. Nijmegen, Netherlands.

Wagner, T. (2008). *The global achievement gap: Why even our best schools don't teach the new survival skills our children need-and what we can do about it.* New York: Basic Books.

Weigend, M. (2014). The digital woodlouse--scaffolding in science-related scratch projects. *Informatics in Education*, 13(2), 293-305. doi: 10.15388/infedu.2014.09

Wilson, A., & Moffat, D.C. (2010, September). Evaluating Scratch to introduce younger school children to programming. *In Proceedings of the 22nd Annual Psychology of Programming Interest Group*, Leganés, Spain.

Wu, W., Chang, C., & He, Y. (2010). Using Scratch as game-based learning tool to reduce learning anxiety in programming course. *In Proceedings of Global Learn Asia Pacific*, 1845- 1852.

Yıldırım, A., & Şimşek, H. (2005). *Sosyal bilimlerde nitel araştırma yöntemleri.* Ankara: Seçkin Yayıncılık.

Zanetti, H., Borges, M., & Ricarte, I. (2016, November). Pensamento Computacional no Ensino de Programação: Uma Revisão Sistemática da Literatura Brasileira. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, 27(1), 21. doi: 10.5753/cbie.sbie.2016.21

Appendix A

Semi-Structured Interview Questions for Informatics Technologies and Software Teachers

INTERVIEW QUESTIONS REGARDING THE LESSON PLAN TEACHING PROGRAMMING TO MIDDLE SCHOOL 6TH GRADE STUDENTS

Interviewee: ………………………………………… Interviewer:……………….

Date and Time: …………/………………2017 at …………….:………………

Duration: ………………………………………………………………………………

1. How suitable do you think the activity in the lesson plan for teaching programming to middle school 6th graders is for the students' levels?
2. How suitable do you think the computer activities in the lesson plan for teaching programming to middle school 6th graders are for the students' levels?
3. Do you find the duration allocated for the activities in the lesson plan for teaching programming to middle school 6th graders sufficient? Why or why not?
4. Do you find the duration allocated for the computer activities in the lesson plan for teaching programming to middle school 6th graders sufficient? Why or why not?
5. How applicable do you think the activities in the lesson plan for teaching programming to middle school 6th graders are in overcrowded classrooms?
6. How applicable do you think the computer activities in the lesson plan for teaching programming to middle school 6th graders are in overcrowded classrooms?

Appendix B

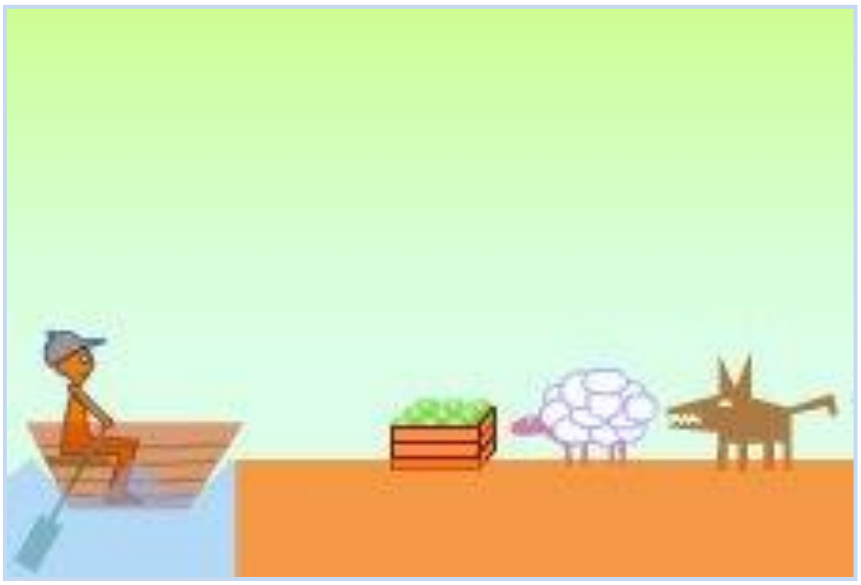
Students' Diaries on Problem Solving Concepts and Approaches

1. On the whole, how did you find the teaching of the lesson? (Please specify the good and bad sides, and what you liked and disliked.)	
2. What are the good sides of the activities in the lesson?	
3. What are the bad sides of the activities in the lesson?	
4. What do you think an algorithm resembles?	
5. Remembering the activities performed in the lesson, what do you think an algorithm is?	
6. Please discuss the activities about algorithm which were performed in the lesson.	
7. What do you think a problem resembles?	
8. Remembering the activities performed in the lesson, what do you think a problem is?	

9. Please discuss the activities about the problem status which were performed in the lesson.

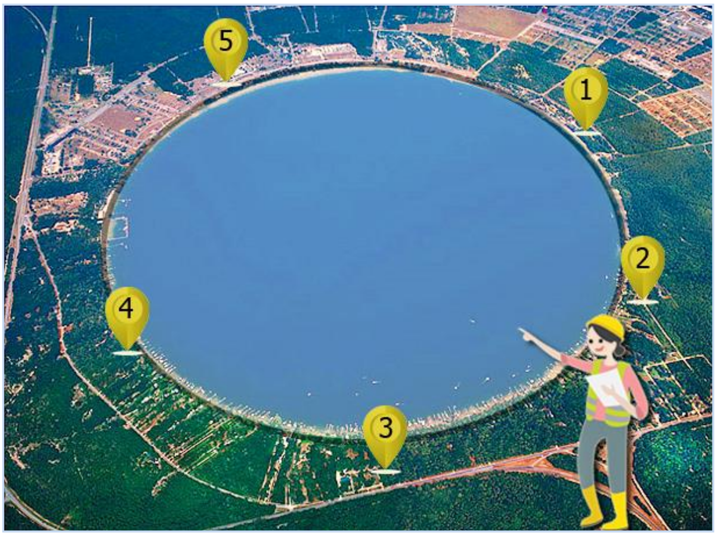
Appendix C
Activity: "Helping the Shepherd"

Scenario: There was a shepherd. Next to the shepherd was a wolf, a grass and a sheep. They'il cross the river with a raft. If the wolf passes the sheep eats grass, the grass passes, the wolf eats the sheep, the sheep wolves, the sheep crosses without eating sheep.



Appendix D
Activitity: "I am finding the cities"

Scenario: Friends, we see the map engineer Ayşe on the picture. Her new job is to map the cities in the region where a circular lake is located. Can you help her with this? The distances between these cities are given in the table. Match cities A, B, C, D and E in the table with the numbers 1, 2, 3, 4 and 5 in the photo.

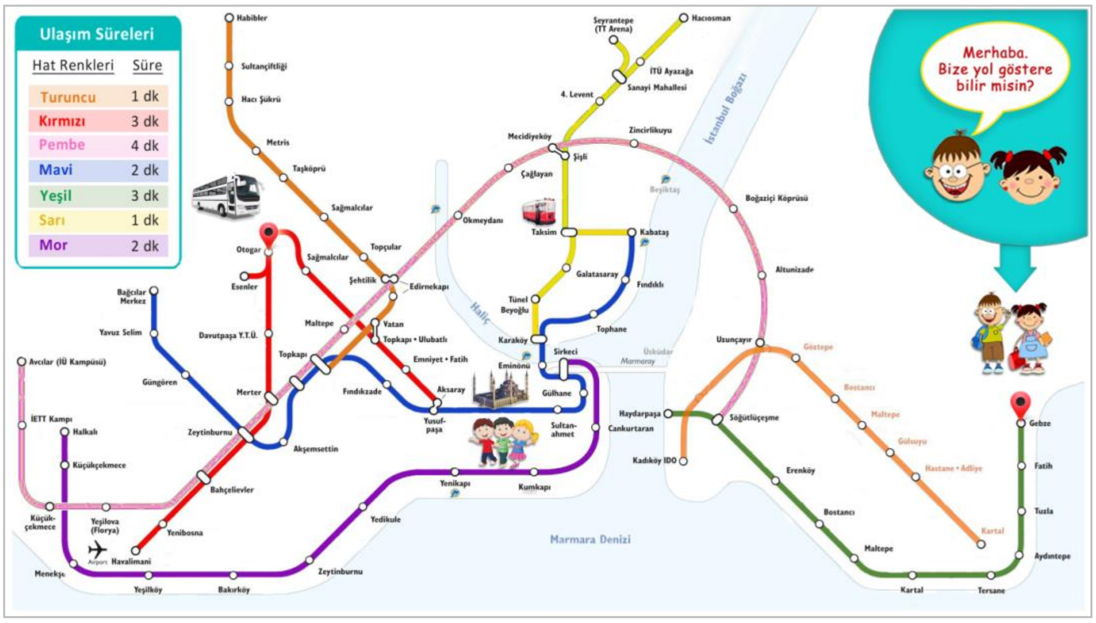


Appendix E
Guidelines to Examination Scratch Projects for Students

Category	Reviewed application name	Feature				
		Movement	Sound	Changing scene	Talking balloon	Move with keyboard
Games						
Animations						
Stories						
Music and dance						
Art						
Your application idea						

Appendix F
Activitity: “Ali and Ayşe are going on holiday”

Scenario: Ali and Ayşe live in Gebze. They'il go to their grandmother with bus. But before they go, they want to say goodbye to their friends. So, they will meet their friends in Sultanahmet Square. But they don't know which way to go. Can you show Ali and Ayşe the way to Sultanahmet and then to the bus terminal as soon as possible?



Appendix G

“Ali and Ayşe are going on holiday” activity Problem Acknowledgement Table (PAT) form

#	Questions	Answers
1.	Where do Ali and Ayşe live?	Gebze
2.	Who will Ali and Ayşe go to?	Grandmother
3.	Where will Ali and Ayşe meet with their friends and where will they go after they meet?	They will meet at Sultanahmet Square. They'il go to the bus station after saying goodbye to their friends.
4.	How do you describe the transfer points on the map?	The intersection of two different colors the path.
5.	What do transportation times mean at the event?	The elapsed time between two stops in each line color.
6.	How is the time elapsed from one stop to another stop in the activity calculated?	With given transportation times.

Appendix H

“Ali and Ayşe are going on holiday” activity Strategy Devise Form (SDF)

#	Boarding Stop	Landing Stop	Count of Stops	Line Color	Time
1.	Gebze	Söğütlü Çeşme	9	Green	27 min
2.	Söğütlü Çeşme	Mecidiyeköy	5	Pink	20 min
3.	Şişli	Karaköy	4	Yellow	4 min
4.	Karaköy	Sultanahmet	4	Blue	8 min
5.	Sultanahmet	Yusuf-paşa	1	Blue	2 min
6.	Aksaray	Otogar	4	Red	12 min
7.					
8.					
9.					
10.					
Total Time:					73 min
Total Number of Transfers:					4

Secondary Computer Science Teachers’ Pedagogical Needs

Olgun Sadik[1]
Anne Todd Ottenbreit-Leftwich[2]
Thomas Andrew Brush[3]

[1]Inonu University
[2]Indiana University
[3]Indiana University

DOI: 10.21585/ijcses.v4i1.79

Abstract
The purpose of this study is to identify secondary computer science (CS) teachers’ pedagogical needs in the United States. Participants were selected from secondary teachers who were teaching CS courses or content in a school setting (public, private, or charter) or an after-school program during the time of data collection. This is a qualitative study using CS teachers’ discussions in the Computer Science Teachers Association’s (CSTA) email listserv, responses to open-ended questions in a questionnaire, and discussions in follow-up interviews. Content analysis, thematic analysis and constant comparative method of qualitative data analysis were used to analyze the data. The most common pedagogical need expressed was learning student-centered strategies for teaching CS and guiding students’ understanding with the use of scaffolding and team-management strategies in CS classes. Furthermore, addressing students’ beliefs in CS and their preconceptions in math and reading were important factors influencing teaching CS effectively in secondary schools.

Keywords: computer science education, computing education, teachers’ needs, teachers’ challenges, computer science pedagogy, learning computer science, teaching computer science, teacher education

1 Introduction

Within the United States, more stakeholders have pushed for increased opportunities for computer science (CS) for K-12 students (Code.org Advocacy Coalitian, 2019; Smith, 2016). To meet this need, state departments of education and legislation have implemented numerous policies and resources to broaden K-12 student access to CS (Code.org, 2019). As of 2019, 15 states have made it a requirement that all high schools need to offer at least one computer science class each year (e.g., Indiana, Texas, West Virginia, Arkansas) (Code.org, 2019). Some have even pushed further, requiring all high school students to complete at least one CS course (e.g., Arkansas and West

Virginia). However, integrating CS in K-12 schools is a systemic change (Barr & Stephenson, 2011; DeLyser & Wright, 2019) and this change requires well-prepared teachers (Lang et al., 2013). With the push for more CS classes, many states have been trying to identify how to meet this need and improve CS teachers' content, curriculum and pedagogical knowledge through teacher education and professional development programs. According to the Code.org Advocacy Coalition's State of Computer Science Education (2019) report, the number of teacher certification programs has been increasing in the United States from 27 in 2017, 33 in 2018 to 38 in 2019. Another way this need for more CS teachers has been met is by training teachers from other content areas (e.g., math, science, foreign languages, etc.) through professional development programs (Menekse, 2015). Other programs have created CS teachers through supporting industry. For example, the Technology Education and Literacy in Schools Program supports teachers by partnering industry computer scientists with practicing teachers (DeLyser & Preston, 2015). However, studies have shown that many secondary CS teachers express needing more support beyond the certification programs and professional development programs provided (Giannakos, Doukakis, Pappas, Adamopoulos, & Giannopoulou, 2015; Qian, Hambrusch, Yadav, & Gretter, 2018). This leads to the problem of what CS teachers' challenges and needs are to help them improve their craft of teaching CS.

1.1 Teachers' Challenges and Needs in CS Education

In spite of current efforts to offer high quality CS classes in K-12 schools, many teachers have challenges and needs to better teach CS in their classes. These challenges can be categorized as knowledge and skills needs, curricular needs, contextual needs and pedagogical needs (Sadik, 2017). In terms of knowledge and skills, CS teachers share their limited CS content knowledge and skills in various studies (Angeli et al., 2016; Yadav, Gretter, Hambrusch, & Sands, 2016). Yadav et al. (2016) expressed that some CS teachers came from different backgrounds and had to learn CS alone from books and online resources. When these teachers were asked about their knowledge and skills needs, they reported the need for better understanding of programming constructs and more experience on computer programming (Yadav et al., 2016). Furthermore, understanding the principles of coputational thinking emerged as another important knowledge and skills need. CS teachers shared their needs for professional development programs to learn and implement the principles of computational thinking in their CS classes (Fessakis & Prantsoudi, 2019). In terms of curriculum, CS teachers reported needs for more resources for content delivery and assessment (Sentance & Csizmadia, 2016). Most CS teachers reported creating their own curricular resources alone (e.g. books, materials, presentationss) (Brown & Kölling, 2013; Yadav et al., 2016). Finding or creating quality assessment is especially difficult due to problem based and collaborative nature of CS projects (Wilson & Guzdial, 2010). Therefore, teachers expressed the need for assessment materials to guide and grade students' work in CS classes (Vivian et al., 2020). In terms of contextual needs, one of the important barriers was no to limited collaboration opportunities between CS teachers in schools (Tenenberg & Fincher, 2007). Most teachers expressed the feeling of loneliness and asked for a colleague to share ideas and resources (Yadav et al., 2016). Even though there are online communities for CS teachers to collaborate (Sadik, 2017), CS teachers need other CS teacher colleagues to regularly discuss and share ideas and resources in their subject (Cutts, Robertson, Donaldson, & O'Donnell, 2017). Furthermore, CS teachers need more support from their schools in terms of accessing up-to-date computer hardware and software. This show the need for technical support staff who can take care of software and hardware updates and maintanance in schools (Sadik, 2017).

In order to address and help CS teachers' needs for content and curriculum, The Association for Computing Machinery (ACM), Code.org, the Computer Science Teachers Association (CSTA), the Cyber Innovation Center, National Math and the Science Initiative developed the K-12 Computer Science Framework. However, recent research emphasized CS pedagogy as the most significant aspect of teaching CS effectively in K-12 schools (Giannakos et al., 2015; Sentance & Humphreys, 2018; Yadav, et al., 2016). Due to the project and/or problem-based nature of CS education courses (Yadav et al., 2016), planning the lessons, keeping the students active and guiding them in their learning process can be difficult (Davenport, 2000). Furthermore, diverse student needs and interests make it more complicated to teach CS in their classes (Schulte & Knobelsdorf, 2007). Therefore, recent studies reported CS teachers' limited experience with student centered practices and teaching students with diverse needs (Che, Kraemer, & Sitaraman, 2019) and suggested conducting more research on CS teachers' pedagogical needs. In order to understand CS teachers' pedagogical needs, an interested reader needs to know what effective teaching and learning means in CS education as well as the successful instructional strategies in CS classrooms.

1.2 CS Pedagogy

Teaching is a complex field that requires strong pedagogical knowledge for planning, leading and mentoring dynamic classroom environments and students' learning experience. The International Society for Technology in Education (ISTE) (2011) highlighted that effective teaching and learning in CS education requires knowledge of various instructional strategies and materials. Research on CS education pedagogy has been a topic of interest since early 1980. Four of the successful strategies used in CS education include problem-based learning (Kay et al., 2000; Yadav, Subedi, Lundeberg, & Bunting, 2011), project-based learning (Mills & Treagust, 2003), pair programming (McDowell, Werner, Bullock, & Fernald, 2006), and media computation (Guzdial, 2003). For instance, in problem-based learning (PBL), students work in groups to solve a complex problem using various types of scaffolds (Hmelo-Silver, 2003). In CS education PBL, students are given a complex CS problem in a computer lab environment with tutorials to facilitate their problem-solving process. Project-based learning has also been implemented in CS education. Tasks in project-based learning are designed similar to projects in real life and give learners the opportunity to apply their knowledge in product design and development (Mills & Treagust, 2003). Even though there are similarities with PBL, project-based learning is product focused and requires students to be careful about resources and time, while PBL gives more flexibility in this process. Another important practice, pair programming, is a technique in which two programmers work on the same programming task design and development using one computer simultaneously. This technique was derived from CS industry practices and has been used as an instructional method in both K–12 and higher education. Media computation was a recently developed instructional technique that emphasized learning computing concepts and skills in digital media design (Guzdial, 2003). Guzdial has argued that digital media (e.g. images, videos, audios) can be modified and redesigned using programming and this process can help students learn computation in a more meaningful way. In addition to the instructional methods discussed here, previous studies documented the success of using various other tools and environments in CS education such as computer games (Papastergiou, 2009), virtual learning environments (Esteves, Fonseca, Morgado, & Martins, 2011) and robotics kits (Bers, Ponte, Juelich, Viera, & Schenker, 2002).

Even though CS education research has provided evidence of learning gains with all these strategies, tools and contexts, recent research expressed the need for understanding in-service teachers' explicit challenges in CS pedagogy, especially in student centered learning environments. Due to limited population in earlier levels, this study only targets secondary CS teachers and aims to understand their needs in effective teaching in CS education. The present study proposes that K-12 teachers may have crucial needs in pedagogy (Hazzan, Lapidot, & Ragonis, 2015; Yadav et al., 2016) and any efforts aiming to prepare CS teachers, initially, need to identify CS teachers' pedagogical needs and answer the following research question:

1. What pedagogical needs do U.S. secondary school teachers share to better teach computer science classes or content in their classes?

The study aims to reach teachers who are teaching CS content in formal and informal learning environments and explore secondary education teachers' pedagogical needs in teaching CS in the US. It is argued that by identifying needs in this grade range, this study will help:

1. Address the specific pedagogical needs of secondary CS teachers,
2. Inform administrators and scholars as they develop data-driven professional development programs and resources and inform teacher education programs about in-service secondary CS teachers' pedagogical needs and assist them in preparing courses that address those needs.

2 Method

Recent literature emphasized the need for improving CS teachers’ knowledge and skills in CS pedagogy; however, there is limited exploratory research that explains what that need entails. Using multiple data collection methods for data complementarity and triangulation with rich data (Creswell & Clark, 2017), this study employs general qualitative research design to explore and explain CS teachers’ pedagogical needs in detail.

2.1 Participants and Setting

Although previous research has recommended beginning CS education as early as kindergarten (Fessakis, Gouli, & Mavroudi, 2013; Kelleher, Pausch, Pausch, & Kiesler, 2007), due to the limited number of CS teachers and courses at the K-5 level at the time of the present study, the researchers focused only on secondary school level. Participants were selected from secondary teachers who were teaching CS courses or content in a school setting (public, private, or charter) or an after-school program during the time of data collection. In this study, secondary education refers to both middle and high school between grades 6-12. Even though there was a discussion in the literature about the title of the field of study as computing education versus computer science education (Guzdial, 2015), the second was selected because of its broader use in K-12 education and public society. “Model Curriculum for K-12 Computer Science” defined CS as “an academic discipline that encompasses the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society” (Tucker, 1996, p. 6). With this description, teachers who mentioned teaching the following and related areas were identified and considered as CS teachers for the purposes of this study:

> *programming, hardware design, networks, graphics, databases and information retrieval, computer security, software design, programming languages and paradigms, logic, translation between levels of abstraction, artificial intelligence, the limits of computations (what computers cannot do), applications in information technology and information systems, and social issues (Internet security, privacy, intellectual property (Barr & Stephenson, 2011, p. 113).*

2.2 Data Collection and Analysis

This research was completed in three phases:

1. The first phase included analysis of CS teachers' publicly available email communications in Computer

Science Teachers Association (CSTA) listserv. The preliminary step of phase 1 was organizing all the emails in the listserv, which included creating an Excel spreadsheet that included the email subjects, email content and the dates of each email sent. As members responded at different days and times to various topics, all the communications in the spreadsheet were grouped by subject and then sorted by date and time for each subject. After organizing the data, the spreadsheet was imported to Nvivo qualitative data analysis software.

2. The second phase included conducting a questionnaire with open ended questions to all the teachers in the CSTA membership database.
3. In the third stage, interviews with eight purposefully selected teachers were conducted to understand CS teachers' pedagogical needs in more detail.

2.2.1 Email Listserv Analysis

In the first phase of the research, due to extensive data in the email listserv, the teachers' communications were analyzed in two steps. In the first step, the researcher used inductive content analysis to identify the conversations related to pedagogy using the teacher communications (3 years, *N*=1706) in the email listserv (Weber, 1990). Following the content analysis, in the second step of the analysis in this phase, the email conversations related to pedagogy were analyzed using the thematic analysis technique (Braun & Clarke, 2006) to identify CS teachers' specific pedagogical needs with evidence. Every email identified as pedagogical needs was coded and each of these was categorized into one of seven themes. Table 1 shows examples of codes emerged from the email conversations and depicts how the researchers categorized the codes into the themes. Peer debriefing technique was used in the analysis stage to manage subjectivity, challenge researcher assumptions and discuss alternative interpretations. Initially, one researcher analyzed the email data and other researchers reviewed and provided alternative views to his interpretations.

Table 1. Examples of codes and the themes emerged

CODE EXAMPLES	**THEMES EMERGED**
• How to introduce programming • Help students learn programming • Guiding students' learning	• Instructional Strategies for Teaching CS
• Connections between programming languages • Pedagogical parallels between programming languages • Transfer learning to new contexts	• Transferring Skills between Programming Languages and Platforms
• Debugging students code • Efficient strategies to answer students' coding questions • Strategies that guide students solve their own problems	• Answering Students' Questions
• Effective teamwork strategies • Creating environments for students' collaboration • Allowing students to work together	• Facilitating Student Interaction and Collaboration

• Looking for resources that increase interest in CS • Increasing interest • Kids give up easily	• Teaching Students with Low Interest in CS
• Limited fundamental skills • Defining set of skills necessary to become a programmer Students different development levels in high school	• Teaching Students Who Lack Fundamental Skills

2.2.2 Questionnaire Analysis

In this phase, the questionnaire was disseminated to CS teacher members of CSTA through their email addresses. 121 members responded to the open-ended questions with rich comments and examples. The open-ended responses were included and analyzed using the constant comparative method of qualitative data analysis (Glaser, 1965). Glaser defines the purpose of the constant comparative method as providing an alternative to analysis, comparing themes and looking for agreements and conflicts (“negative cases or a consideration of alternative hypotheses”), and increasing credibility in the study results. The researcher imported the teachers' responses to Nvivo software and conducted comparison of the email listserv analysis results with the qualitative questionnaire responses. At the end of the questionnaire, the participants were asked to contribute voluntarily in a follow-up, semi-structured interview (3rd phase), in order to elaborate upon their pedagogical needs with examples from their practices.

2.2.3 Interview Data Collection and Analysis

A semi-structured interview protocol was used (Fraenkel, Wallen, & Hyun, 2011) and the interview questions were developed from the questionnaire responses to gather more information on secondary teachers' needs. For example, in the questionnaire, when a participant shared his or her need for learning more strategies on student centered learning in CS classes, s/he was asked to define the need and give examples from his/her classroom. Eight purposefully selected teachers (based on their responses to the questionnaire) participated in the follow-up interviews. The researchers selected participants who reported diverse needs and aimed to enrich the data with detailed explanations and examples. For example, teachers who mentioned scaffolding as a need or strong need in the questionnaire were purposefully selected to the interviews to enrich the data with better explanations and more examples.

The interview data collection ended when the researchers decided that the data was saturated, as suggested by Guba and Lincoln (1985): “exhaustion of sources, saturation of categories, emergence of regularities, and over-extension.” The interviews were fully transcribed. The teacher names and all the identifying information were replaced with pseudonyms. The data was analyzed using the constant-comparative method in the same way as conducted in phase 2 to enhance the findings. This phase also helped the researchers provide more descriptive information about CS teachers' needs with examples from the participants' explanations of their practices.

2.3 Issues of Reliability-Validity and Limitations of the Study

The researchers employed various techniques to ensure the standard of trustworthiness of this research (Guba & Lincoln, 1985). In the early stages of the research, multiple researcher meetings were held face to face to establish our research questions, identify our criteria for participant selection and develop and clarify the data collection methods. The data collection process was started using existing teacher discussions in the email listserv. This gave researchers access and opportunity to interpret teacher discussions in a real-life context. This data is triangulated using multiple forms of data sources (questionnaire and interviews) to answer the research question with rich data. This helped the researchers ensure that the data is credible. This reduced the risk for researcher bias. The interviews in the final phase were fully transcribed and analyzed with multiple researchers' input and agreement. Furthermore, the researchers sent the results to the participants and asked their confirmation of the researchers' interpretations. This technique is called member checking and have been used in qualitative research to ensure confirmability of the research results (Birt, Scott, Cavers, Campbell, & Walter, 2016). Nevertheless, the study has limitations. Even though all the possible efforts have been made for trustworthiness, the participants were coming from secondary teacher members of one target organization and did not represent all the secondary CS teachers in the US. Furthermore, there is always risk for researcher bias in qualitative research.

3 Findings

The pedagogical needs were identified from CS teachers' communications in both the listserv and the questionnaire and expanded on with the interview data. Within the findings, email quotations are marked with "E," questionnaire responses are marked with "Q," and the follow up interviews are marked with "I." Each quotation is also identified with a number indicating a unique participant. For the purposes of this study, teachers' perceived pedagogical needs included the following themes:

3.1 Need for Learning Student Centered Strategies for CS Education

Most secondary CS teachers stated that they need to learn new strategies to teach CS content and enhance student learning in their courses. For example, one teacher wanted to know how to teach Linux with new instructional strategies: "I think it's really important for my students to learn Linux, but I have no idea how to teach it" (E-2). Pair programming was one of the student-centered learning strategies primarily discussed in the listserv. For example, one teacher shared her failure in using pair programming in a CS class and asked for advice from other CS teachers: "I have done pairing, but must not have done it correct, because it was not as productive as I'd liked. What ideas do you have" (E-4)? In the questionnaire, 17 teachers asked for help related to student-centered strategies in CS education. For example, one teacher stressed the need for facilitating students' learning: "I will appreciate further pedagogical help with teaching computer science and facilitating student knowledge, particularly helping students make better connections with the material, and more effectively debug without needing face time or one-on-one time from me" (Q-2). Another teacher emphasized his need for supporting students' learning via scaffolds in a computer-programming course: "I need better strategies for teaching computer programming to students who have never written computer programs before. What's best to teach first, second, etc. I want a scaffolded approach to teaching programming" (Q-3). When asked to explain this need in more detail and provide examples in the interviews, all eight participants described their current practices and explained why they want to learn new instructional strategies. For instance, I-2 described his current practice as "straight lecture" and explained it:

> *I do a lot of straight lectures in my classroom, I do lot of type; I'll pull the projector, I'll put code on the projector and have kids follow through on their own computers, basically they copy down what I'm doing. (I-2)*

Another teacher emphasized his need to learn new instructional strategies, problem-based and pair programming approaches for student-centered CS learning:

> *I want to learn new instructional strategies that I never got, because I didn't go to a teaching school, I never went to school to learn to be a teacher. I feel that it's personal that I needed to learn better how to do things like problem-based learning and pair programming. (I-7)*

3.2 Need for Strategies Guiding Students Transfer Skills Between Programming Platforms and Languages

In the listserv emails, the participants expressed the need for helping secondary students transfer their skills from visual programming platforms (e.g., block-coding) to text-based environments (e.g., Python), and between text-based environments (e.g., from Python to Java). For example, one teacher emphasized her students' difficulty transferring their CS learning from visual programming environments to text-based programming environments: "In my teaching, it seems that majority of students have difficulty migrating concepts they learned from visual environments into text-based environments. Starting with Scratch/turtle I found that I essentially had to reteach concepts in Python/Java" (E-11). Another teacher shared the same concern between text-based programming languages: "I have found the same thing with students going from python to Java or python to C" (E-12). In the questionnaire responses, the participants exemplified the transfer issue. For instance, a CS teacher who had started teaching an advanced placement CS (AP-CS) course for the first time and complained about her students' lack of transfer from her previous classes to the new AP CS class: "…we are working on strings again and it seems they do not remember what we were supposed to learn previously. It makes me think there must be something else I can do to help the process" (Q-8). Similarly, the interviewees listed their students' lack of transfer between the CS courses when there were connections. For instance, I-4 shared his failure to help students see the connections between Scratch and other programming languages when moved from one set of content to another:

> *At the end of each semester I ask them what did they think of it, did they feel using Scratch was valuable? Were they able to transfer what they had learned from one language to another in them? The responses I get, for the most part, are that while they found it interesting, they didn't see the parallels. I know the parallel is there. They just don't make the connections. (I-4)*

3.3 Need for Strategies Guiding Students' Errors while Coding

The email listserv members stressed the need for strategies addressing students' questions and problems in programming activities in classroom and explained it as the need for analyzing code quickly and guiding the students' CS learning process while coding. One teacher stressed the importance of analyzing code quickly: "When helping students with a project, the teacher needs to quickly analyze what is wrong with the frustrated students' program and then give some advice on how to fix it" (E-14). In another email, one teacher mentioned the need to provide one-to-one guidance to students "We don't provide one-on-one mentoring to students while programming" (E-3). In the questionnaire responses, the participants explained this need as assessing code for correctness, giving students appropriate feedback, guiding students through solving code errors, and creating scaffolding that does not need face-to-face guidance. For instance, one teacher explicitly stated correcting students' code as a need for better CS instruction: "I need help with fixing students code, etc. -sample debugging questions in C++, Java C - and pseudocode" (Q-11). Another teacher emphasized her need for strategies to assess students' code for correctness, structure, and efficiency: "I need to streamline the process of assessing my students' code for correctness, use of comments, ease of use, and grammar and spelling in output statements and comments" (Q10). When asked to explain this need in more detail and provide examples in the interviews, four teachers shared their own strategies and asked for more strategies that can lead their students to finding errors in their code. The interviewees asked for strategies that can lead students to find the errors themselves by exploration. One of these teachers used questioning as scaffolding and asked for more strategies that could guide his students:

> *The easy answer that is the wrong one is to point the student to the bug and say: Well, here's what you did wrong. The harder answer in mind, but the one that I prefer is to sort of ask the student: What do you mean by this chunk of code? And have them explain to me what they are trying to say and then Socratic method or using questioning to get them to see what they have said, where what they have said doesn't add up with what they intended to say. I would like to learn more strategies other than just questioning and flat out giving them the answer to help them with that. (I-1)*

3.4 Need for Strategies to Facilitate Student Interaction and Collaboration

The email listserv members stressed the need for strategies creating a collaborative classroom environment and guiding student discussions. For instance, when teachers discussed problems in their classroom, several teachers emphasized the importance of creating an environment for learning: “How do kids learn and how do we create an environment to allow that learning to best take place” (E-13). In the questionnaire responses, the participants explained this need as creating a collaborative environment where all the students help each other and attend learning activities: “How to facilitate student independence and helping each other and strategies for when they really are stuck and need individual help and I can't be there for everyone” (Q-12). Another teacher commented about a collaborative environment as a need: “Creating collaborative environment and excitement around problems and solutions” (Q-13). When asked to explain this need in more detail and provide examples in the interviews, four teachers described their own teaching practices, emphasizing the importance of student interaction and broadening the scope of the need to ensure all the students' active participation in collaborative work. One interviewee explained that collaboration is essential in a CS a class:

> *[Programming projects] forces collaboration in the classroom between me and the students but especially between the students. They quickly learn everybody's got questions and there's only one teacher. If you want your question answered, you've got to go to someone else in the class. The other people in the class don't have to be experts, they just have to be one step ahead of you, because they can help you with the thing you were trying to do.” (I-5)*

This collaboration requires a flexible environment where students help each other, become creative and learn from each other. Another teacher explained the need for a less structured CS classroom for effective collaboration and need for ensuring student learn in teamwork:

> *It's really trying to map out your lesson and your unit plans to allow that time, balance that time for them to be able to do some of that collaboration together, or working together around problem solving, or evaluating each other, like a game or an animation or something that they created. All of this requires a little bit less structured environment than what the students are used to, so it's about allowing them to be able to be creative, and have that time so they can just think and kick the tires, but also making sure that that time is effective, and they're learning (I-3)*

3.5 Need for Strategies Teaching Students with Low Interest in CS

Secondary CS teachers reported that they need more strategies to teach students with low interest in CS, which includes strategies to inform students about the challenges and benefits of CS. Increasing student interest especially became an issue for the participants when a CS class was required or students enrolled in a class when they did not have another option. They believe when a student does know what CS is and understands both the challenges and benefits of CS, their interest and motivation increase. Therefore, some teachers were willing to select students before they were allowed to register. The listserv discussions, questionnaire responses and interviews support these findings. The teachers in the email listserv stressed the need to increase their students' interest and motivation and defined those as preconditions, especially when learning programming. The following example demonstrates this need:

> *I have discovered as an in-service teacher that there are a small number of factors that have a disproportionate impact on learning. They are centered around student motivation and interest. The primary question here is how do you get the student engaged and actively seeking knowledge in CS. (E-12)*

While some teachers discussed that being an elective class reduced the enrollment rates in CS classes, some teachers in the listserv stressed this as a positive factor. One of the teachers described this as: "all it really needs is a desire to learn the stuff, and the fact that it is an elective really helps on that front" (E-12). In the questionnaire responses, 10 teachers commented about student interest in CS classes. In these responses, the teachers stated that increasing interest is a need in the following conditions:

- When the course is required, there were students in the class with no to little interest in learning CS (*N*=5).
- Students came with low interest from various backgrounds (*N*= 7).
- It was hard to sustain student interest and motivation in programming classes (*N*=7).

Therefore, those teachers solicited strategies for increasing student interest in CS, as in the following example with underrepresented student populations:

> *Students are now required to take a computer programming class before graduation - so I have many underrepresented populations and girls enrolled in my course. Students who have little interest in the class. I would be greatly beneficial to share strategies on how to motivate students who have little interest in the class. (Q-33)*

Sustaining interest in CS classes was also a need mentioned in the questionnaire responses. Most teachers stressed that many students come in with interest in CS, but as the class moves forward to difficult concepts in coding, they lose that interest. They are looking for strategies to sustain interest throughout the course. For instance, one teacher approached this issue and asked for help:

> *As a math teacher, I often encounter students not interested in a content. In that class, they have to accept it because it's a core requirement for graduation. In computer science, all students enter with an interest, but some lose that interest as the year progresses. How can I keep them motivated to finish the year even when interest wanes? (Q-35)*

The interviews validated the listserv and questionnaire responses. Seven out of eight interviewees expressed this as a need. When asked to explain this need in more detail and provide examples, the interviewees stated that there were students in their classes who lost interest in the subject and considered it both unrelated to their needs or too challenging. Therefore, most of the interviewees solicited strategies to motivate these students with low interest into learning CS. For instance, one of the teachers stated that there were not many electives for students to choose from in his school and students with low interest were forced to take his class. He expressed the need to be able to help those students to learn CS concepts and skills:

> *I do get a lot of kids in my classes because we're a small school, we don't have a ton of electives. Sometimes kids just get dropped into classes they don't necessarily want, but I still feel that there's value for those kids to understand. (I-2)*

3.6 Need for Strategies Teaching Students Who Lack Literacy and Math Skills

For all the participants of this research, math background was reported as an issue for the students in CS classes. Furthermore, reading comprehension tended to be a problem in some contexts where there were economically disadvantaged and minority students with limited content knowledge in general. Secondary CS teachers stated that they need more strategies to teach students with low mathematics and reading comprehension skills. The email listserv participants stated that some of their students lacked fundamental skills, which influenced their understanding and ability to apply the concepts and skills in CS classes. One teacher described the problem: “Somehow some of my students have core (base) knowledge missing or so confused that it makes it hard for them to progress” (E-37). Math, especially algebra, emerged as the most important skill that students needed to be successful in CS classes: “I have many instances in which I have to divert my curriculum to teach them Algebra concepts that should have known” (E-39). Reading comprehension was another factor identified in the listserv, especially for understanding instructions in CS classes. One teacher highlighted her concern: “Reading comprehension is a struggle. Some of them can't follow specific instructions and don't understand the importance of flawless execution, error free and clear thinking when writing programs” (E-39). In the questionnaire responses, 25 teachers shared opinions about fundamental skills they consider important for learning CS and their need for strategies in dealing with those students who lack them. For instance, one teacher emphasized this as an evolving need with CS becoming more available in schools:

> *Excellent question! As schools adopt CS for everyone, it will add the challenge of working with students with lower reading and math skills. This issue could be the game changer in CS education since up until now the students in the CS program have typically been quite high academic achievers. (Q-26)*

Seven out of eight final-phase interviewees expressed learning strategies to teach students who lack fundamental skills as a need. The interviewees provided examples of scenarios where their students had difficulty on simple calculations and reading directions. One of the interviewees explained lack of math background as a problem with an example in his CS classes:

> *Some students could think algebraically or abstractly without a transcript credit, but those students who have a limited background in math will struggle when solving complex problems. Even the very simple assignment like I'm going to prompt the user for the number of gallons of gasoline burned and the distance traveled and calculate*

> *the miles per gallon, I've seen students in 9th and 10th grade really struggle to figure out how to do that. (I-1)*

Reading comprehension was not explicitly mentioned in overall interview discussions; however, some teachers connected this issue to students' disadvantages and limited skills in core content areas in general. For instance, one of the teachers mentioned English as a second language students in his CS classes:

> *I do have a large number of students who are classified as English language learners. I would say the majority of my students, English is not their first language. The vast majority of their parents have very limited English and speak Spanish at home. (I-8)*

One of the interviewees explained this further, and argued that learning CS requires a background in different content areas:

> *I think it's a combination. I think the content knowledge is obviously extremely important because it is what's out there. I think it goes hand in hand with trying to figure out how to best present them to students. It involves all the related to other things they are learning about in school. It obviously ties in with a lot of math, science, English, a lot of other subjects want to make sure there's some development for students. (I-7)*

4 Discussion

The overall findings suggest that secondary CS teachers need community help from other teachers to meet their needs in teaching CS (Ni & Guzdial, 2012). This study focused on the pedagogical needs and found the participants' primary need as learning and using student-centered learning strategies in CS classes. Specifically, secondary CS teachers stressed the need for scaffolding strategies that can guide students in solving computing problems in various levels at the individual level and in teams. The sections below discuss the participants' pedagogical needs in detail.

4.1 Learning and Applying Scaffolding Strategies in CS Classes

The primary discussion was evolved from the need for student-centered learning strategies in solving CS problems in coding activities. In order to be successful in learning CS, students need guidance while solving problem based activities (Hmelo-Silver, 2004; Mayer, 2003). This guidance is conceptualized as scaffolding in the literature. Scaffolding helps students to complete complex learning tasks that are difficult to achieve without assistance (Belland, 2014; Wood, Bruner, & Ross, 1976). Scaffolding makes the learning process more manageable (Hmelo-Silver & Bromme, 2007) and is of primary importance in supporting students to reflect on their own learning and develop higher order thinking skills (Azevedo, Cromley, Winters, Moos, & Greene, 2005; Saye & Brush, 2002). Brush and Saye (2002) defined two types of scaffolding: hard and soft. Hard scaffolds are planned in advance such as multimedia resources (e.g. helpful websites). Soft scaffolds are more dynamic and happen simultaneously. Examples of soft scaffolding include teachers' using questioning strategy to guide their problem-solving processes.

Guzdial (2015) also stressed scaffolding as one of those primary conditions for effective CS education. In the present study, secondary CS teachers identified similar issues. In CS, when students are assigned a problem, they often use trial-and-error when they are not given adequate facilitation (Shute, Sun, & Asbell-Clarke, 2017). Trial-and-error is not an efficient problem-solving strategy (Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013), it takes time and often yields no results. The findings suggest a more purposeful design of computational problem solving (Shute et al., 2017), with teachers' scaffolding embedded in the process and facilitating students' coding practices. Secondary CS teachers want to learn more strategies to answer students' questions during coding practices, particularly when students are debugging their own codes, finding and fixing errors, and increasing the efficiency of their code. These are important components of computational thinking in programming activities (Grover & Pea, 2013). Students tend to expect teachers to point out their mistakes, but this is not an effective teaching method. Providing feedback in the form of guiding questions helps students to assess and reflect on their own learning (Nicol & Macfarlane-Dick, 2006).

4.2 Creating a Collaborative Teamwork Environment in CS Classes

The results suggest that teachers need tools and strategies to make sure all students actively participate and equally contribute during teamwork and recommend further research to identify successful team building and management strategies, especially in pair programming activities. Teamwork is a crucial part of CS learning and benefits students' learning experience through sharing information and receiving feedback within a social community of peers (Sancho-Thomas, Fuentes-Fernández, & Fernández-Manjón, 2009). In pair programming, students work collaboratively in teams. Pair programming involves strong pair collaboration within teams, with less teacher involvement (Nagappan et al., 2003). It is not surprising that teachers who requested feedback in the email listserv about pair programming were also looking for team building and management skills to create successful collaborative environments. The findings suggest that CS teachers want to make sure that all their students actively participate in teamwork. Poor teams often involve one expert student taking all the responsibility, while other members become passive participants who may not benefit from the learning opportunities (Shimazoe & Aldrich, 2010). Cooperative team work is the primary condition of successful pair programming practices in computer programming courses (Umapathy & Ritzhaupt, 2017). Teachers in the study also solicited strategies for creating a collaborative learning environment where students search for answers from their partners rather than the teacher. It is important to note that CS teachers have limited time during a class period to answer all the questions, and team work becomes an important opportunity to alleviate those problems (Sancho-Thomas et al., 2009). However, teachers need to be aware of the differences between team members in terms of knowledge/skills and personality (Ally, Darroch, & Toleman, 2005).

4.3 Transferring Knowledge and Skills between Programming Languages and Platforms

Even though limited examples were found, the findings suggest that secondary teachers need strategies to help students transfer learning and build on their previous knowledge in subsequent CS classes. Transfer of learning is important, not only to make connections between concepts and skills, but also when developing new learning (Driscoll, 1994). Transfer of learning can make CS experiences more connected and meaningful. This need was emphasized in previous CS education studies related to programming. For example, learning the concepts and logic of programming (e.g., variables, loops) from visual programming tools (e.g., Scratch) in middle school helped students learn complex programming concepts more easily in text based languages, and also increased enrollment numbers in CS classes (Armoni, Meerbaum-Salant, & Ben-Ari, 2015). In another study, Franklin et al. (2016) conducted a study with high-school students' and reported this as a challenge. Franklin et al. reported this as an important concern and recommended the teachers help students see the connection between visual programming and text-based programming activities in their instruction.

4.4 Increasing Students' Interest in Learning CS

Most secondary teachers in this study reported that students in their classes do not perceive a connection between CS and their personal lives or their professional futures; thus, they do not value learning CS. Therefore, if teachers want to increase students' positive beliefs and interest in learning CS, they need to find ways to make CS relevant to their students' lives (Tew, Fowler, & Guzdial, 2005). For example, Umbleja (2016) stressed that code.org activities increase students' understanding and interest in CS if they are provided the concepts and skills in primary education or earlier secondary levels. Umbleja emphasized that as they age into their teenager years, it may be difficult to change their biases about CS. This statement is validated in studies with middle school girls. Outlay, Platt, and Conroy (2017) stressed that young ages are critical to develop positive beliefs of CS competency and knowledge and increase interest. To assist with CS teachers' needs regarding students' beliefs, the researchers suggest the following:

1. Introducing CS to students early in their education through short-term programs and local campaigns,
2. Developing curriculum that represents diverse student interests,
3. Defining short-term benefits and learning outcomes

Short-term programs and national promotions (e.g., code.org) have been reported as helpful in gaining interest in CS (Wilson, 2014); however, sustaining that interest in the classroom may be challenging. The findings suggest that teachers need to constantly define goals and benefits for students to retain their interest. Therefore, a place-based education approach may be a successful strategy to define goals for students to serve their local communities through service-learning projects. Service learning projects in CS higher education provide long-term motivational benefits for learning (Sanderson, 2003) while creating an engaging and motivating learning environment in K-12 (Billig, 2000). However, K-12 CS education literature appears to be limited in this regard. Service-learning approaches may be promoted, and further research should explore effective strategies to employ service learning in K-12 CS education.

4.5 Teaching Students with Limited Math and Reading Background

The findings suggest that low abilities in Math and reading create challenging teaching environments for CS teachers trying to manage and teach students with widely varying needs and learning goals. For example, CS teachers reported that some students were unable to do simple calculations and read instructions to complete simple programming tasks. These findings align with previous research regarding the importance of math background in learning CS. In a study with 123 first-year introductory programming courses at a higher education institutions, Bergin and Reilly (2005) found that mathematics ability is one of the important predictors of students' success in programming classes. Grover, Pea, and Cooper (2016) reported correlation between middle school students' prior reading and Math abilities, and learning computer programming. Regarding students' math and reading comprehension, secondary CS teachers were looking for strategies to guide students with problems in those areas before they register for CS classes. However, this goes against the national goal known as "CS for All" (Smith), and would create bigger problems in the future. In fact, several teachers in the study mentioned that "CS for All" may create classes that are difficult to manage. Teachers need strategies to help them deal with wide range of skills in math and reading among their students. Further research may be helpful to develop student-centered practices for teaching students with different math and reading skills in secondary CS classes.

5 Conclusion

In the present study, the data in the email listserv demonstrated a wide range of teachers' self-reported needs in a natural community setting. Furthermore, rich discussions evolving from the listserv discussions allowed the researchers to enrich the findings with the questionnaire and interview data. Similar studies were also conducted to understand CS teachers’ needs in smaller samples. This study is important to validate the findings of previous studies and start new discussions in the area with rich data. Similar to this study, previous studies recommended using student-centered strategies for effective CS education (Hazzan et al., 2015). For example, current literature discussed problem-based learning (Mills & Treagust, 2003) and pair programming (McDowell et al., 2006). Furthermore, scaffolding and facilitating students' learning in student centered environments were discussed in the CS education literature (e.g. (Caspersen & Bennedsen, 2007). However, this study provided evidence that secondary CS teachers try to use pair programming, problem-based learning and scaffolding in their classes but they were not satisfied with the outcomes of their efforts. Secondary teachers need feedback to improve their practices.

Although this study represents a large population of secondary CS teachers, the participants in this study are U.S. members of one international organization and do not represent all the CS teachers in the U.S. Furthermore, the findings are not suggested to be generalized. The data represents teachers who are members of the CSTA, and were identified as secondary CS teachers in the email listserv based on available information. Although all the efforts have been made to exclude them, there is a small risk that some elementary teachers or higher education faculty might be included in the findings. This study suggests that future studies need to be conducted to explore the following topics in CS classrooms:

- Successful student-centered learning strategies (e.g. PBL and pair-programming) in secondary CS classes
- The development of strategies to increase student interest and motivation in learning CS.

Furthermore, it would be helpful to develop research studies that observe successful teachers' practices in their classrooms. Action research studies that allow teachers and researchers to work together (Lang et al., 2013) and aim to address secondary CS teachers' needs in practice would create helpful information for practicing teachers. Developing PD programs based on the suggestions of this study may promote and develop data-driven PD programs for teachers. With the goal of CS for All, this study may be replicated to explore the needs of elementary CS teachers.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Ally, M., Darroch, F., & Toleman, M. (2005). A framework for understanding the factors influencing pair programming success. In *Proceedings of the International Conference on Extreme Programming and Agile Processes in Software Engineering* (pp. 82-91). New York, NY. https://doi.org/10.1007/11499053_10

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Journal of Educational Technology & Society*, *19*(3), 47-57.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to “real” programming. *Transactions on Computing Education, 14(*4), 25. https://doi.org/10.1145/2677087

Azevedo, R., Cromley, J. G., Winters, F. I., Moos, D. C., & Greene, J. A. (2005). Adaptive human scaffolding facilitates adolescents’ self-regulated learning with hypermedia. *Instructional Science, 33*, 381-412. https://doi.org/10.1007/s11251-005-1273-8

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

Belland, B. R. (2014). Scaffolding: Definition, current debates, and future directions. In *Handbook of Research on Educational Communications and Technology* (pp. 505-518): Springer. https://doi.org/10.1007/978-1-4614-3185-5_39

Bergin, S., & Reilly, R. (2005). Programming: Factors that influence success. *ACM SIGCSE Bulletin, 37*(1), 411-415. https://doi.org/10.1145/1047124.1047480

Bers, M. U., Ponte, I., Juelich, C., Viera, A., & Schenker, J. (2002). Teachers as designers: Integrating robotics in early childhood education. *Information Technology in Childhood Education Annual, 2002*(1), 123-145. https://www.learntechlib.org/primary/p/8850/article_8850.pdf

Birt, L., Scott, S., Cavers, D., Campbell, C., & Walter, F. (2016). Member checking: A tool to enhance trustworthiness or merely a nod to validation?. *Qualitative Health Research, 26(*13), 1802-1811. https://doi.org/10.1177/1049732316654870

Billig, S. (2000). Research on K-12 school-based service-learning: The evidence builds. *Phi Delta Kappan, 81*(9), 658-664. https://digitalcommons.unomaha.edu/slcek12/3

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology, 3*(2), 77-101. https://doi.org/10.1191/1478088706qp063oa

Brown, N. C. C., & Kölling, M. (2013). A tale of three sites: Resource and knowledge sharing amongst computer science educators. In *Proceedings of the Ninth Annual Conference on International Computing Education Research* (pp. 27–34). La Jolla, CA: ACM. https://doi.org/10.1145/2493394.2493398

Brush, T. A., & Saye, J. W. (2002). A summary of research exploring hard and soft scaffolding for teachers and students using a multimedia supported learning environment. *The Journal of Interactive Online Learning, 1*(2), 1-12. http://www.ncolr.org/jiol/issues/pdf/1.2.3.pdf

Caspersen, M. E., & Bennedsen, J. (2007). Instructional design of a programming course: A learning theoretic approach. In *Proceedings of the 3rd International Workshop on Computing Education Research* (pp. 111-122). Atlanta, Georgia: ACM. https://doi.org/10.1145/1288580.1288595

Che, S. M., Kraemer, E. T., & Sitaraman, M. (2019). Prospective high school computer science teachers' perceptions of inquiry pedagogy and equity. In *Proceedings of the AERA Online Paper Repository*. Toronto, Canada: AERA. https://doi.org/10.302/1444296

Code.org. (2019). State Tracking 9 Policies. Retrieved from https://docs.google.com/spreadsheets/d/1YtTVcpQXoZz0IchihwGOihaCNeqCz2HyLwaXYpyb2SQ/pubhtml

Code.org Advocacy Coalitian. (2019). 2019 State of Computer Science Education. Retrieved from https://advocacy.code.org/

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

Cutts, Q., Robertson, J., Donaldson, P., & O'Donnell, L. (2017). An evaluation of a professional learning network for computer science teachers. *Computer Science Education*, *27*(1), 30-53.

Davenport, D. (2000). Experience using a project-based approach in an introductory programming course. *IEEE Transactions on Education, 43*(4), 443-448. https://doi.org/10.1109/13.883356.

DeLyser, L. A., & Wright, L. (2019). A Systems Change Approach to CS Education: Creating Rubrics for School System Implementation. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 492-498). Aberdeen, Scotland: ACM. https://doi.org/10.1145/3304221.3319733

DeLyser, L. A., & Preston, M. (2015). A public school model of cs education. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering* (pp. 233–238). Nevada, USA. https://pdfs.semanticscholar.org/90fc/817b131253e2092e712aa1c78fd0f7778d68.pdf

Driscoll, M. P. (1994). *Psychology of learning for instruction*. Washington, DC: Allyn & Bacon.

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2011). Improving teaching and learning of computer programming through the use of the Second Life virtual world. *British Journal of Educational Technology, 42*(4), 624-637. https://doi.org/10.1111/j.1467-8535.2010.01056.x

Fessakis, G., & Prantsoudi, S. (2019). Computer science teachers' perceptions, beliefs and attitudes on computational thinking in Greece. *Informatics in Education, 18*(2), 227. https://doi.org/10.15388/infedu.2019.11

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87-97. https://doi.org/10.1016/j.compedu.2012.11.016

Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (2011). *How to design and evaluate research in education*. New York: McGraw-Hill

Franklin, D., Hill, C., Dwyer, H. A., Hansen, A. K., Iveland, A., & Harlow, D. B. (2016). Initialization in Scratch: Seeking knowledge transfer. In *Proceedings of the 47th ACM Technical Symposium on Computer Science Education* (pp. 217–222). Memphis, Tennessee, USA: ACM. https://doi.org/10.1145/2839509.2844569

Giannakos, M. N., Doukakis, S., Pappas, I. O., Adamopoulos, N., & Giannopoulou, P. (2015). Investigating teachers' confidence on technological pedagogical and content knowledge: An initial validation of TPACK scales in K-12 computing education context. *Journal of Computers in Education, 2*(1), 43-59. https://doi.org/10.1007/s40692-014-0024-8

Glaser, B. G. (1965). The constant comparative method of qualitative analysis. *Social Problems, 12*(4), 436-445. https://dx.doi.org/10.4135/9781412950558.n101

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

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

Guba, E. G., & Lincoln, Y. S. (1985). *Naturalistic inquiry*. New York, NY: Sage.

Guzdial, M. (2003). A media computation course for non-majors. *ACM SIGCSE Bulletin, 35*(3), 104-108. https://doi.org/10.1145/961511.961542.

Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics, 8*(6), 1-165. https://doi.org/10.2200/S00684ED1V01Y201511HCI033

Hazzan, O., Lapidot, T., & Ragonis, N. (2015). *Guide to teaching computer science: An activity-based approach.* New York, NY: Springer. https://doi.org/10.1007/978-1-4471-6630-6.

Hmelo-Silver, C. E. (2003). Problem-based learning. In J. W. Guthrie (Ed.), *Encyclopedia of Education* (Second ed., Vol. 4, pp. 1173-1175). New York: MacMillan Reference.

Hmelo-Silver, C. E. (2004). Problem-Based Learning: What and how do students learn? *Educational Psychology Review, 16*, 235-266. https://doi.org/10.1023/B:EDPR.0000034022.16470.f3

Hmelo-Silver, C. E., & Bromme, R. (2007). Coding discussions and discussing coding: Research on collaborative learning in computer-supported environments. *Learning and Instruction, 17*(4), 460-464. https://doi.org/doi:10.1016/j.learninstruc.2007.04.004

International Society for Technology in Education. (2011). NETS for Teachers. Retrieved from https://www.iste.org/standards/for-educators

Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J. H., & Crawford, K. (2000). Problem-based learning for foundation computer science courses. *Computer Science Education, 10*(2), 109-128. https://doi.org/10.1076/0899-3408(200008)10:2;1-C;FT109

Kelleher, C., Pausch, R., Pausch, R., & Kiesler, S. (2007). *Storytelling alice motivates middle school girls to learn computer programming.* In the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 1455–1464). California, USA: ACM. https://doi.org/10.1145/1240624.1240844

Lang, K., Galanos, R., Goode, J., Seehorn, D., Trees, F., Phillips, P., & Stephenson, C. (2013). *Bugs in the system: Computer science teacher certification in the US*. Retrieved from https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CSTA_BugsInTheSystem.pdf

Mayer, R. E. (2003). *Learning and instruction*. New Jersey, NY: Pearson.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM, 49*(8), 90-95. https://doi.org/10.1145/1145287.1145293

Menekse, M. (2015). Computer science teacher professional development in the United States: A review of studies published between 2004 and 2014. *Computer Science Education, 25*(4), 325-350. https://doi.org/10.1080/08993408.2015.1111645

Mills, J. E., & Treagust, D. F. (2003). Engineering education—Is problem-based or project-based learning the answer. *Australasian Journal of Engineering Education, 3*(2), 2-16.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. *ACM SIGCSE Bulletin, 35*(1), 359-362. https://doi.org/10.1145/792548.612006

Ni, L., & Guzdial, M. (2012). *Who am I? Understanding high school computer science teachers' professional identity.* In P*roceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 499–504). North Carolina, USA: ACM. https://doi.org/10.1145/2157136.2157283

Nicol, D. J., & Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in Higher Education, 31*(2), 199-218. https://doi.org/10.1080/03075070600572090

Outlay, C. N., Platt, A. J., & Conroy, K. (2017). Getting IT together: A longitudinal look at linking girls' interest in IT careers to lessons taught in middle school camps. *ACM Transactions on Computing Education, 17*(4), 20. https://doi.org/10.1145/3068838

Papastergiou, M. (2009). Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Computers & Education, 52*(1), 1-12. https://doi.org/10.1016/j.compedu.2008.06.004

Qian, Y., Hambrusch, S., Yadav, A., & Gretter, S. (2018). Who needs what: Recommendations for designing effective online professional development for computer science teachers. *Journal of Research on Technology in Education, 50*(2), 164-181. doi:10.1080/15391523.2018.1433565

Sadik, O. (2017). *What do secondary computer science teachers need? Examining curriculum, pedagogy, and contextual support*[Doctoral dissertation, Indiana University]. ProQuest Dissertations Publishing.

Sancho-Thomas, P., Fuentes-Fernández, R., & Fernández-Manjón, B. (2009). Learning teamwork skills in university programming courses. *Computers & Education, 53*(2), 517-531. https://doi.org/10.1016/j.compedu.2009.03.010.

Sanderson, P. (2003). Where's (the) computer science in service-learning? *Journal of Computing Sciences in Colleges, 19*(1), 83-89

Saye, J. W., & Brush, T. (2002). Scaffolding critical reasoning about history and social issues in multimedia-supported learning environments. *Educational Technology Research and Development, 50*(3), 77-96. https://doi.org/10.1007/BF02505026

Schulte, C., & Knobelsdorf, M. (2007). Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. In *Proceedings of the Third International Workshop on Computing Education Research* (pp. 27–38). New York, NY: ACM. https://doi.org/10.1145/1288580.1288585

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*(2), 351-380. https://doi.org/10.1007/s10639-012-9240-x

Sentance, S., & Humphreys, S. (2018). Understanding professional learning for computing teachers from the perspective of situated learning. *Computer Science Education, 28*(4), 345-370. https://doi.org/10.1080/08993408.2018.1525233

Sentance, S., & Csizmadia, A. (2016). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies*, 1–27. https://doi:10.1007/s10639-016-9482-0

Shimazoe, J., & Aldrich, H. (2010). Group work can be gratifying: Understanding & overcoming resistance to cooperative learning. *College Teaching, 58*(2), 52-57. https://doi.org/10.1080/87567550903418594

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.

Smith, M. (2016). Computer science for all. Retrieved from https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all

Tenenberg, J., & Fincher, S. (2007). Opening the door of the computer science classroom: The disciplinary commons. *ACM SIGCSE Bulletin, 39*(1), 514-518. https://doi.org/10.1145/1227504.1227484

Tew, A. E., Fowler, C., & Guzdial, M. (2005). Tracking an innovation in introductory CS education from a research university to a two-year college. *ACM SIGCSE Bulletin, 37*(1), 416-420. https://doi.org/10.1145/1047124.1047481

Tucker, M. S. (1996). Skills, Standards, Qualification systems, and the american workforce. In L. B. Resnick & J. G. Wirt (Eds.), *Linking school and work: Role for standards and assessment* (pp. 23-51). San Francisco CA: Jossey-Bass.

Umapathy, K., & Ritzhaupt, A. D. (2017). A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education, 17*(4), 16. https://doi.org/10.1145/2996201

Umbleja, K. (2016). *Can K-12 students learn how to program with just two hours?* In *Proceedings of the International Workshop on Learning Technology for Education Challenges* (pp. 250-264). New York, NY: Springer. https://doi.org/10.1007/978-3-319-42147-6_21

Vivian, R., Franklin, D., Frye, D., Peterfreund, A., Ravitz, J., Sullivan, F., ... & McGill, M. M. (2020). Evaluation and assessment needs of computing education in primary grades. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 124-130). Trondheim, Norway: ACM. https://doi.org/10.1145/3341525.3387371

Weber, R. P. (1990). *Basic content analysis*. Los Angeles, CA: Sage Publications.

Wilson, C. (2014). Hour of code: We can solve the diversity problem in computer science. *Inroads, 5*(4), 22. https://doi.org/10.1145/2684721.268472

Wilson, C., & Guzdial, M. (2010). How to make progress in computing education. *Communications of the ACM, 53*(5), https://doi.org/35-37.10.1145/1735223.1735235

Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, *17*(2), 89-100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: Understanding teacher experiences and challenges. *Computer Science Education*, *26*(4), 235-254. https://doi.org/10.1080/08993408.2016.1257418

Yadav, A., Subedi, D., Lundeberg, M. A., & Bunting, C. F. (2011). Problem-based learning: Influence on students' learning in an electrical engineering course. *Journal of Engineering Education*, *100*(2), 253-280. https://doi.org/10.1002/j.2168-9830.2011.tb00013.x

Perceived Acceptance and Use of Scratch Software for Teaching Programming: A Scale Development Study

Serife Nur Yildiz[1]
Alev Ates-Cobanoglu[2]
Tarik Kisla[3]

[1] Izmir Institute of Technology
[2] Ege University
[3] Ege University

DOI: 10.21585/ijcses.v4i1.59

Abstract
This paper reports the development process of a scale for Information and Communication Technology (ICT) teachers' acceptance and use of Scratch for teaching programming. For early beginners of programming, Scratch is the most popular block-based software for facilitating programming teaching (Zhang and Nouri, 2019) worldwide. Also, in Turkey, Scratch is widely used and has been chosen as a block-based coding tool for developing problem solving ability, self-efficacy, motivation, interest (Kasalak and Altun, 2018). Despite that wide use, there is a lack of scale for understanding ICT teachers' acceptance and use of Scratch for teaching programming. Theoretically, this scale development study is based on Unified Theory of Acceptance and Use of Technology (UTAUT) which is also explained in the paper. The sample of the study includes 265 ICT teachers from Turkish Ministry of National Education (MoNE) secondary schools who used Scratch software in their courses. According to exploratory and confirmatory factor analyses results, the final version for the scale includes 28 items. The Cronbach Alpha coefficient is 0.97. "Perceived Acceptance and Use of Scratch Software for Teaching Programming Scale" (PAUSS-TPS) can help the practitioners who aim at understanding the ICT teachers' level of acceptance and use of Scratch; the researchers who wish to investigate perceived contribution according to various variables and the decision-makers for deciding to use Scratch or shift to another block-based tool for teaching programming at early ages.

Keywords: scratch; teaching programming; coding instruction; computer science instruction; block-based programming

1. Introduction: Defining Computational Thinking

In today's developing and changing world, individuals need to acquire 21st century skills such as critical thinking, creative thinking, problem solving, and cooperative working in line with the diversifying social requirements. To teach 21st century skills to learners and to become a technology-producing society, programming or coding education has been on the agenda of Turkey particularly as of the 2000s. As Gülbahar and Kalelioğlu (2018) reported that particularly since 2012, computer science topics such as problem solving, and programming have been the focus of the Turkish curriculum. The literature includes findings revealing that an effective teaching programming manages to teach these skills. For example, when learners are taught programming and designing tools, their product development, problem solving and analytical thinking skills improve (Akpınar and Altun, 2014; Çakıroğlu, Sarı and Akkan, 2011). Besides using the products provided by technology, another quality expected from the learners of the 21st century is developing projects and products. It is recommended that programming should be taught at early ages to be able to raise individuals who can develop projects and products (Yükseltürk and Altınok, 2015). Great attention is attached to teaching programming in many countries like Korea, China, India, Spain, and Canada. In Korea, for instance, amendments were made in the curricula of the information and communication course at the secondary school level in 2010 and at the high school level in 2011. Programming is included in the computer sciences course curriculum at high school level in India and Israel (Gülmez, 2009); while in Canada it is offered within the scope of the courses delivered in connection with computer engineering and computer sciences at secondary education institutions (Stephenson, 2001).

On the other hand, teaching programming involves several challenges in terms of the programming language in use and the characteristics of the target population. One of these challenges is the structure of traditional programming languages especially for new beginners (Çatlak, Tekdal and Baz, 2015; Genç and Karakuş, 2011; Gomes and Mendes, 2007). Another difficulty is that some procedures and concepts concerning programming are abstract for learners (Ersoy, Madran and Gülbahar, 2011). In this regard, teaching programming at early ages, it becomes important to use software focusing on such aims as defining the problem, doing analysis, evaluation and creativity by taking the learner away from the challenges of code-writing (Kert and Uğraş, 2009). To this end, visual programming tools like Scratch, Code.org, Alice, Small basic and Toontalk were developed to provide new beginners with a more interest-provoking and enjoyable environment.

In Turkey, some additions were made into the ICT and software course as of 2012 regarding such topics as Microsoft Office applications, desktop settings, control panel, of which many learners today have some knowledge. In 2012, MoNE included the learning domains of problem solving, programming, and developing original products into the "ICT and software course" content, which was also mentioned by Gülbahar and Kalelioğlu (2018). Revised curriculum of the Information technology and Software Course also covered the "Problem solving and Programming" unit in 6th grade program (MoNE, 2018). Concerning these learning domains, specifically Scratch program was recommended as a block-based visual programming tool for introducing programming in the formal annual lesson plans of 6th graders of the mentioned course.

Weintrop (2019) claims that block-based programming tools like Scratch provide visual cues, drag-and-drop options, and syntax error prevention to the beginners of programming and computer science. Block-based programming has two main purposes: (i) Simplifying programming syntax, (ii) Getting attention of target learners for programming to increase the number of learners who are interested in programming (Durak, Karaoğlan Yılmaz, Yılmaz and Seferoğlu, 2017; Yükseltürk and Altıok, 2016). It is considered that these facilitating characteristics of block-based programming tools have the potential to encourage children to produce their own tiny programs easier than traditional text-based programming. In another study by Oluk and Oluk (2019), block-based programming tools such as Scratch, Mblock, Alisblock, code.org, thinkerCad circuits, blockly, alice, code game were compared with features such as "Turkish language support", "platform", "pricing", "age level", "robotic coding compatibility". It was stated that the selection between programming tools should be made according to variables such as the target audience, content, educational environment considering the specified criteria. In addition, it was reported that Scratch, code.org and Blockly are used more than others. Zhang and Nouri (2019) reported that Scratch is the most popular visual block programming language as of 2018 rankings. Scratch, which is an educational and social software tool, enables children particularly over eight to develop video games and interactive stories and to share these over the Internet easily. Revealing the worldwide popularity of Scratch, Scratch website for statistics (https://scratch.mit.edu/statistics/) reported that there are almost 50 million registered users, as of January 2, 2020. Also, in Turkey, ICT teachers in Turkish secondary schools are responsible for teaching programming at introductory level and they make use of block-based programming tools like Scratch, Mblock, Code.org etc. Among these tools for early stages of programming, Scratch is widely used in Turkey as a block-based coding tool for developing problem solving ability, self-efficacy, motivation, and interest (Kasalak and Altun, 2018). Therefore, among other tools, the authors chose Scratch software in the study. Besides the popularity of Scratch, it would also be impractical to measure acceptance and use of all block-based programming tools. Consequently, the scope of present study is narrowed down to acceptance and use of Scratch software in teaching programming.
Scratch helps children to program a prototype of a video game or a story, to understand how its different parts run and to visualize an idea (Gonzalez, 2013). Developed by the Lifelong Kindergarten Group within the Massachusetts Institute of Technology (MIT), Scratch consists of three parts as Block Palette, Script Area and Stage. Blocks include block palettes divided in groups within themselves; script area is where programming is implemented, and stage displays sprites (visual material). With block palettes, users bring codes together using the drag-and-drop method and they can test them with stage quickly. The projects designed can be shared online with open-resource codes. Scratch has three versions as 1.4, 2 and 3. Version 2 of Scratch can be used both online and offline.

Studies concerning the use of Scratch in teaching programming have varying results in the literature. Some positive results, for instance, reveal that Scratch helps delivering computational thinking skills according to a systematic review study (Zhang and Nouri, 2019); Scratch increases programming and computer achievement (Ferrer-Mico et al., 2012; Malan and Leitner, 2007; Meerbaum-Salant, Armoni and Ben-Ari, 2013); it affects thinking skills positively (Kert and Uğraş, 2009); Scratch learning is found easy (Genç and Karakuş, 2011; Tanrıkulu and Schaefer, 2011); students enjoy using Scratch (Genç and Karakuş, 2011); students come to class with greater motivation (Wilson and Moffat, 2010; Yükseltürk and Altıok, 2016), and it affects students' creativity positively (Kobsiripat, 2015). On the other hand, some studies report that Scratch has no significant effect on certain variables. For instance, it is stated that Scratch has no significant effect on programming knowledge by Wilson and Moffat (2010), on algorithm instruction by Tekerek, Altan and Akdağ (2012) and on problem solving skills by Kukul and Gökçearslan (2014). As a scale development study, Kasalak and Altun (2018) developed a perceived self-efficacy scale related to block-based programming, namely related to the Scratch tool. While there are a variety of study findings concerning different effects of Scratch on students' problem solving, programming skills and creative thinking and its effects on teaching and learning programming; it is noticeable that most of the reported results are positive and there is a gap for scale development studies in this era. In addition, the perceptions concerning its contribution to teaching programming are mostly determined through blog posts and interviews (Genç and Karakuş, 2011; Kalelioğlu and Gülbahar, 2014). Apparently, there is a lack of comprehensive studies related to Scratch and ICT teachers which points out the importance of current scale development study to measure the level of acceptance and use of Scratch software by the ICT teachers dealing with teaching introductory programming.

Ursavaş (2014) mentioned that teachers' technology acceptance has been one of the most critical factors in the integration of ICT in education. Šumak and Šorgo (2016) claim that the adoption of any technology or teaching practice of a teacher is affected by many factors if the adoption is voluntary. The case in Turkey is that including the use of block-based programming tools, more commonly Scratch, in the curriculum of "ICT and Software" course required teachers to use such tools in their lessons. Clearly, teachers have a key role as change agents in education (Van Der Heijden, Geldens, Beijaard and Popeijus, 2015), and they are a key factor in technology use as well (Mueller, Wood, Willoughby, Ross and Specht, 2008). Effective and efficient use of Scratch by the ICT teachers affects the quality of programming lessons. To do that, we need to know the level of ICT teachers' acceptance and use of Scratch. The results of this scale can lead researchers and decision makers to make judgements about different aspects and effectiveness of Scratch. It is remarkable that there are an inadequate number of studies dealing with the acceptance of Scratch within the scope of the TAM or UTAUT model. Therefore, teachers' acceptance and use of any new technology - in this case, the Scratch software- needs further attention for investigating the factors of the effectiveness of introducing programming via Scratch.

Davis (1989) suggests that user acceptance affects effective implementation of an information technology or information system. As a unifying theory of technology acceptance related models, UTAUT can explain the factors that have the highest effect on individuals' technology acceptance behaviours. UTAUT model is helpful for predicting system usage and making technology-adoption- and technology-usage-related decisions (Chao, 2019). It has four essential determining components as performance expectancy, effort expectancy, social influence, facilitating conditions, behavioural intention to use the system, and usage behaviour and four moderators as gender, age, experience, and willingness to use technology (Venkatesh, Morris, Davis, and Davis, 2003). And this study utilized the UTAUT model as a theoretical framework for measuring the acceptance of ICT teachers for the scale development process.

1.1 Research Questions

In this respect, the present study aims to develop an instrument that can measure acceptance and use of Scratch by ICT teachers' who practice Scratch in "ICT and Software" course. As a result, it is targeted to provide an instrument with which practitioners, researchers and decision makers working on ICT teaching curriculum development and beginner level coding instruction. In this scale development study for measuring ICT teachers' perceived acceptance and use of Scratch for teaching programming based on UTAUT, the research questions are as follows:
RQ1: Statistically, is the PAUSS-TPS a valid scale according to the results of exploratory factor analysis?
RQ2: Statistically, is the PAUSS-TPS a reliable scale according to the results of internal reliability test?

2. The study

2.1.Method

This section explains the method, the sample, and the development process of the scale to measure the perceived acceptance and use of Scratch for teaching programming. In this scale development process, the authors adopted a mixed-method approach, specifically a sequential mixed model design. As Tashakkori and Teddlie (2003) suggests, the first strand of a sequential mixed model study is exploratory, and the second strand is confirmatory. The first phase of this study entails collecting qualitative data from literature records, and quantitative data afterwards. The steps of scale development are given in-detailed below section.

2.2.Participants

The participants of the study include the ICT teachers who work at secondary schools affiliated to MoNE in Turkey and use the Scratch program at 5th grade which is a part of the secondary school system in the current Turkish educational system. The participants were selected from the population using criterion sampling among purposeful sampling strategies. The participants are 256 ICT teachers who work at secondary schools affiliated to MoNE and use the Scratch program in their classes. It was assumed in the study that the participants were willing to take the survey and provide honest self-reflection. The demographics of the participants are shown in Table 1.

Table 1. Demographics of the participants

Variable	**Category**	**f**	**%**
Gender	Male	112	42.3
	Female	153	57.7
Professional Experience	1-5 years	134	50.6
	5-10 years	89	33.6
	10-15 years	42	15.8
School type	Public School	199	75.1
	Private School	66	24.9

2.3.Scale Development Steps

In scientific research, constructing a scale with specific measurement characteristics for the construct measured is the purpose of scaling and the Likert type, multiple choice, or forced-choice items are commonly used response formats (Kyriazos and Stalikas, 2018). In present scale development study, the construct to be measured is the acceptance and use of Scratch software in teaching programming. Mainly, the researchers followed the steps of scale development which Streiner, Norman and Cairney (2015) suggested. For making these steps clear, the authors drew a visual (Figure 1) and explained each step below Figure 1. Table 1 presents the questions that were developed by the researcher to give the participants the opportunity to demonstrate a range of computational thinking practices. These questions were similar in nature to questions commonly used by Computer Science teachers to stimulate students' thinking in their lessons. Many such examples are available to teachers making use of shared resources such as the Computing At School website (Computing At School, 2020). In addition, questions were developed in response to assessable themes identified in the literature around CT.

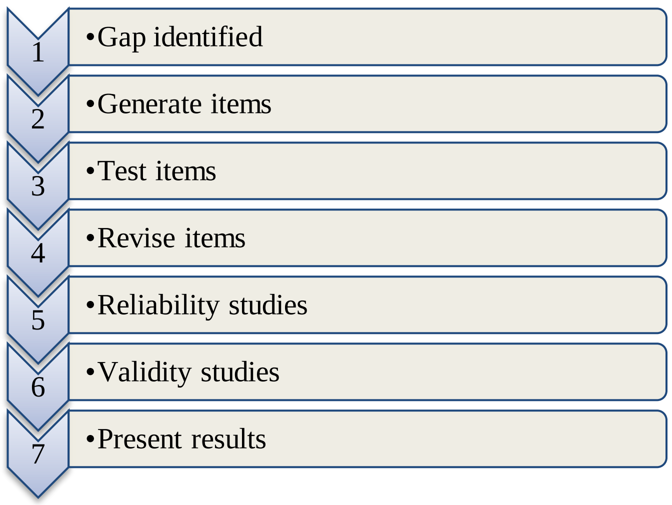


Figure 1. The steps of scale development process in the study (Streiner et al., 2015)

Step 1: Gap identified

For the first step of scale development, literature related to block-based teaching programming was reviewed, available scales were examined. About acceptance or effect of Scratch as a teaching tool, there are some studies with preservice ICT teachers (Choi, 2013; Fesakis and Serafeim, 2009; Saltan and Kara, 2016); and few scale development studies with students (Kasalak and Altun, 2018). Items of scale development studies based on technology acceptance in the literature were also examined (Çam, 2012; Dönmez and Akbulut, 2019; Esen and Büyük, 2014; Özer, Eriş and Özmen, 2012; Serçemeli and Kurnaz, 2016; Teo, 2015; Turan and Haşit, 2014; Ursavaş, 2014). For instance, Çam (2012) examined the factors which affect cloud-computing applications of IT experts and found out that behavioral intentions for applying this technology is affected by perceived usefulness at 99.8% level. In another study, Esen and Buyuk (2014) examined an electronic system, namely e-BYS, acceptance of the employee of the Higher Education Council of Turkey (YÖK). And they reported that self-efficacy, facilitating conditions and social norms positively affect perceived usefulness, perceived ease-of-use, and behavioral intention while anxiety negatively affects. Dönmez and Akbulut (2019) investigated teachers' acceptance of children's internet use. One of their findings is that perceived benefits, risks, and availability of online risks predicted acceptance. Regarding teachers' acceptance, Ursavaş (2014) examined teachers ICT acceptance based on TAM in his dissertation. He extended this framework with some extended variables such as anxiety, compatibility, self-efficacy, perceived enjoyment, facilitation conditions, technological complexity, and subjective norms. Neither of the reviewed studies of scale development were about teachers' acceptance of any of the block-based programming tool.

Step 2: Generate items

To generate the items of the pilot scale form, interviews were held with 15 ICT teachers working at secondary schools affiliated to the MoNE, located within Izmir province. The interviews were conducted using an 11-item semi-structured interview form developed by the first researcher taking the four subscales of the basic TAM into account. A 54-item pilot scale was developed considering the data obtained from the interviews, literature support and the subscales of the UTAUT. Regarding the scale items, opinions were collected from a panel of six experts in the field of Computer Education and Instructional Technology (CEIT). In line with the suggestions of these experts, a technology acceptance measure for teachers developed by Ursavaş (2014) was decided to be used. Of the items, those on which the experts agreed at a degree of 70- 80% were included in the scale (Büyüköztürk, 2017). As a result, a 39-item pilot scale was formed. Prior to the pre-implementation, 10 ICT teachers' feedback was collected about the clarity of the items and the whole scale, also it was observed that the scale was completed in 10 minutes on average.

Step 3: Test items

The pilot scale was applied to the convenience sample of volunteer ICT teachers working at secondary schools and using Scratch in their classes in Turkey. The data were collected via an electronic questionnaire for 2016-2018 years.

Step 4-5-6: Revise items; reliability and validity analysis

After revising items, validity and reliability studies were conducted. The results were reported for the pilot phase of the study. It was considered enough that the 39-item pilot form was filled out by 265 individuals for the analyses to be performed (Tabachnick and Fidell, 2001). The following analyses were carried out on the data collected; (i) exploratory factor analysis (EFA) on SPSS 23.0 using principal component analysis with the factor loadings accepted as minimum 0.30 through varimax rotation for the construct validity, and (ii) confirmatory factor analysis (CFA) on LISREL 8.72 package program to test construct validity. Additionally, Cronbach's Alpha internal consistency coefficient was calculated for the reliability of the scale.

Step 7: Present results

The results of analysis are given in the Results section below.

3. Results

3.1 Results of validity study

In the present scale development study, exploratory factor analysis was performed to reveal the construct validity and determine factor loadings. Before starting the analysis, Kaiser-Meyer-Olkin (KMO) coefficient was calculated to determine the suitability of data and Bartlett Sphericity test was performed. KMO was found as 0.97 and the result of the Bartlett Sphericity test (x^2 =14329.9, p=0.000) was significant according to George and Mallery (2011). As a result of the exploratory factor analysis, eigenvalue of the scale is seen to be gathered under 3 factors larger than 1. The variance explained by these factors is 75.47%. The items and their factor loadings are presented in Table 2.

Table 1. Item factor loadings obtained from the first factor analysis

Item	Factor 1	Factor 2	Factor 3	Item	Factor 1	Factor 2	Factor 3
m1	.902	-.040	.004	m21	.836	-.011	.061
m2	.920	-.094	.012	m22	.850	-.036	.064
m3	.900	-.142	.072	m23	.812	.014	-.071
m4	.914	-.125	.059	m24	.727	.719	.011
m5	.892	-.055	.065	m25	.804	.080	.055
m6	.901	-.109	.014	m26	.903	-.013	-.012
m7	.909	-.114	.003	m27	.636	-.616	-.014
m8	.853	-.076	.000	m28	-.020	.520	.454
m9	.589	.498	-.209	m29	.515	.456	-.222
m10	.889	-.053	.028	m30	.624	.586	.178
m11	.933	-.072	.022	m31	.475	.378	.308
m12	.927	-.083	-.019	m32	.551	.217	.456
m13	.928	-.108	-.016	m33	.356	.073	.368

m14	.934	-.079	-.037	m34	.883	-.116	.012
m15	.888	.032	-.097	m35	.932	-.066	.002
m16	.910	-.025	-.061	m36	.944	-.089	-.010
m17	.516	.542	-.507	m37	.919	-.062	-.010
m18	.574	.556	-.468	m38	.915	-.044	-.017
m19	.891	.015	-.048	m39	.878	-.076	-.049
m20	.835	-.027	-.094				

When Table 2 was examined, it was seen that items 9, 17, 18, 24, 27, 28 29, 30, 31, 32 and 33 are lower than 0.30 or overlapped items. After removing these items EFA was reapplied. The results of the analysis performed with 28 items show that the items gather under a single factor which explains 80.3% of the variance. Table 3 shows the item factor loadings. Table 3 shows the factor loadings obtained from EFA and CFA analyses and the t values estimated with CFA. Evaluating the t values in accordance with Table 2, the factor loadings are statistically significant. As a result of the EFA, the items included in the scale form consist of a total of 28 items gathering under a single factor with factor loadings ranging between 0.80 and 0.94. This factor explains 80.3% of the total variance. Eigenvalue distribution graph is shown in Figure 2.

Table 3. Item factor loadings obtained from the first factor analysis

	Factor 1			
Item	**EFA***	**CFA****	**t*****	**R2**
m1	.90	.85	11.14	.72
m2	.93	.87	11.09	.75
m3	.91	.85	11.14	.73
m4	.92	.84	11.18	.71
m5	.89	.82	11.22	.68
m6	.91	.83	11.21	.68
m7	.92	.85	11.15	.72
m8	.88	.79	11.28	.62
m10	.89	.84	11.18	.70
m11	.94	.91	10.85	.83
m12	.93	.90	10.89	.82
m13	.94	.92	10.78	.84
m14	.94	.90	10.90	.81
m15	.89	.85	11.14	.73
m16	.91	.87	11.07	.76

m19	.89	.85	11.16	.72
m20	.84	.78	11.29	.61
m21	.83	.79	11.27	.63
m22	.85	.80	11.26	.64
m23	.81	.76	11.32	.58
m25	.80	.76	11.30	.60
m26	.90	.86	11.12	.74
m34	.89	.86	11.13	.73
m35	.93	.90	10.85	.92
m36	.95	.91	10.75	.85
m37	.92	.89	10.99	.79
m38	.92	.89	10.98	.79
m39	.87	.85	11.15	.72

* EFA factor loadings

** DFA factor loadings

*** Value of significance (t) of the factor loadings estimated with CFA

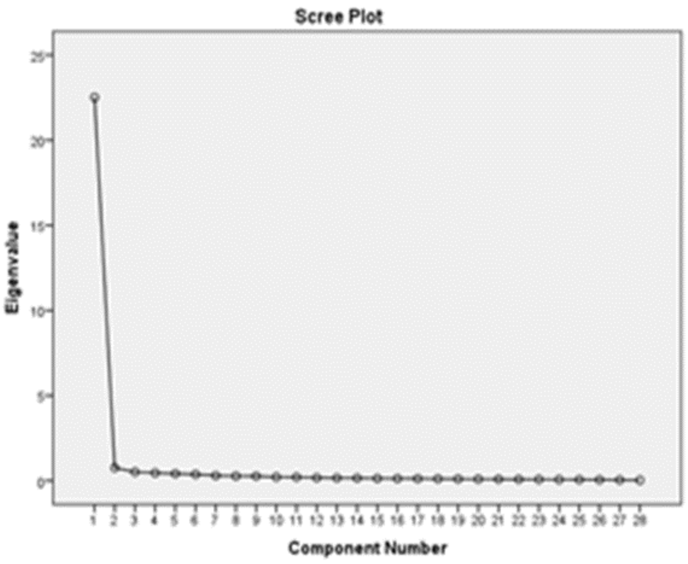


Figure 2. Eigenvalue distribution graph

As the next step of scale development, the items were renumbered, and CFA was performed to see whether the items predicted the factor or not. The model obtained from the analysis is presented in Figure 3. CFA reveals that the construct created is confirmed.

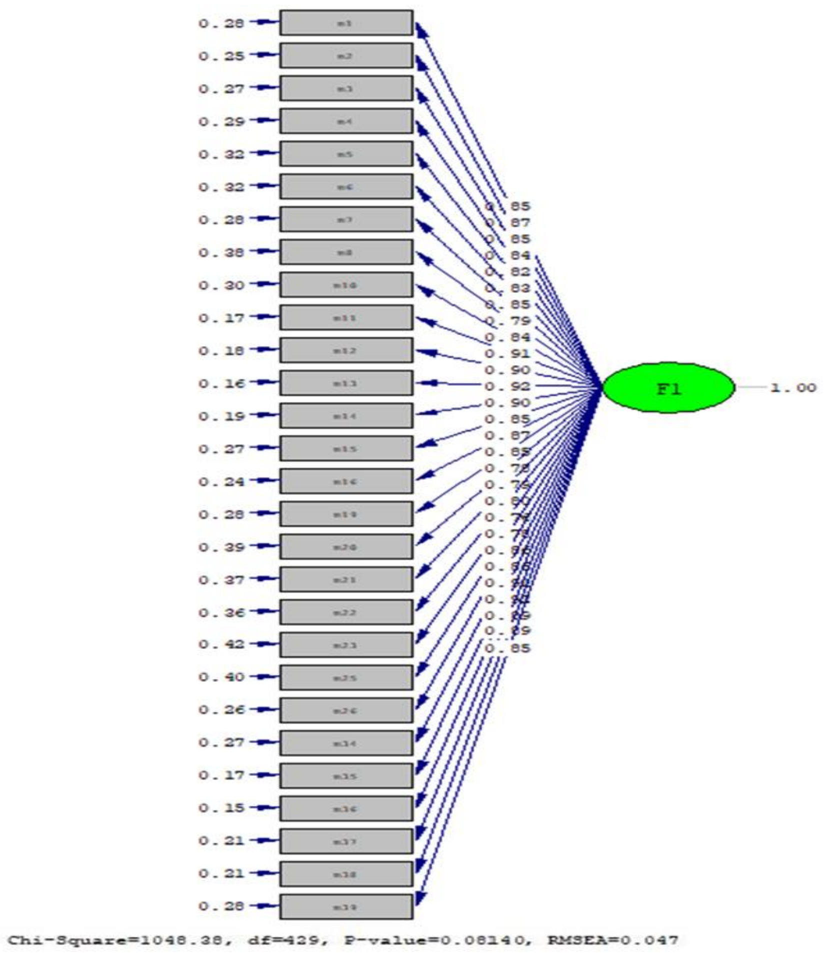


Figure 3. Path diagram pertaining to CFA

Statistical values obtained from CFA are shown in Table 4 together with their acceptable and goodness of fit values according to Schermelleh-Engel and Moosbrugger (2003).

Table 4. Statistical values pertaining to confirmatory factor analysis

	X2	X2/df	RMSEA	S-RMR	GFI	AGFI	CFI
Single factor construct	1048.38	2.44	0.047	0.039	0.93	0.87	0.97
Acceptable fit			.05<RMSA<.10	.05<SRMR<.1	.90<GFI<.95	.85<AGFI<.90	.90<CFI<.95
Goodness of fit value		<3	<.05	<.05	>.95	>.90	>0.95

RMSEA : Root Mean Square Error of Approximation

GFI : Goodness of Fit Index

AGFI : Adjusted Goodness of Fit Index

S- RMR : Standardized RMR

CFI : Comparative Fit Index

CFA shows that GFI and AGFI values are in the acceptable fit range, and RMSEA, S-RMR, CFI and ***X2/df*** values are in the good fit range. These values indicate that the model generated is an acceptable one (Schermelleh-Engel et al., 2003; Kline, 2005; Hooper et al, 2008). In conclusion, the results of the confirmatory factor analysis support the construct validity of the scale.

3.2 Results of reliability study

Cronbach's Alpha internal consistency coefficient was calculated to determine the internal reliability of the PAUSS-TPS. Accordingly, the Cronbach's Alpha internal consistency coefficient was calculated as 0.99. A Cronbach's alpha value of .70 and over is enough for the reliability of scale scores (Büyüköztürk, 2017). This indicates that the internal reliability of the scale is quite good. The highest possible score to be obtained from the five-point type 28-item PAUSS-TPS scale is 140 while the lowest possible score is 28. The whole scale consists of positive items. The final form of the scale is presented in Appendix A.

4. Discussion and Conclusion

In the field of information technologies education, teaching programming / coding has gained importance both in Turkey and around the world particularly since the 2000s. Children's interaction with programming at early ages helps them to improve the 21st century skill of problem solving. In addition, product development, problem solving, and analytical thinking skills are also improved with the instruction of programming and designing tools (Akpınar and Altun, 2014; Çakıroğlu, Sarı and Akkan, 2011). In this respect, Scratch, which is a visual programming environment that facilitates learning programming with a visual interface that can attract children's attention, is used in ICT classes. The number of studies conducted on Scratch has increased since 2012. For instance, Çatlak, Tekdal and Baz (2015) reviewed studies carried out in connection with Scratch. Accordingly, Scratch is most commonly used at secondary school level; and the most frequently focused topics studied in relation with Scratch include its effect on algorithm and teaching programming, its effect on problem solving skills, student opinions, its effects on affective characteristics and its use in different courses.

Related literature includes studies examining the effects of Scratch on different variables (Kert and Uğraş, 2009; Kobsiripat, 2015; Kukul and Gökçearslan, 2014; Wilson and Moffat, 2010; Zhang and Nouri, 2019); and the perceptions about contribution of Scratch to teaching programming are mostly determined through blog posts and interviews (Genç and Karakuş, 2011; Kalelioğlu and Gülbahar, 2014). Moreover, some studies about using Scratch in teaching programming and perspectives of preservice teachers (Choi, 2013; Fesakis and Serafeim, 2009; Saltan and Kara, 2016) and students (Kasalak and Altun, 2018) exist. However, the acceptance and use of Scratch by teachers play a key role for effective teaching programming. Introduction to programming via Scratch is undertaken by ICT teachers within the scope of "ICT and software" course at secondary school level in Turkey. Therefore, it is considered that the scale, which is developed in present study, namely "Perceived Acceptance and Use of Scratch Software for Teaching Programming Scale" (PAUSS-TPS), can provide insight about one of the most commonly used block-based programming tools (Zhang and Nouri, 2019) in teaching programming. Practically, the results of the scale can reveal the teachers' perspective for Scratch in teaching programming at early ages which is beneficial for understanding positive and negative aspects of the tool according to experiences of teachers. When implemented, the results of the scale may lead curriculum designers to suggest or avoid using Scratch which needs further investigations.

The scale consists of self-report questions which is a limitation of the study. Also because of practical reasons, the geographical location which the study took place is Turkey and that can be a delimitation. It is suggested researchers study at other geographical locations and compare acceptance and use of Scratch at other countries as well. It is recommended that researchers should use the PAUSS-TPS to examine ICT teachers' acceptance and use of Scratch for teaching programming in terms of several variables. These variables may include type of school, grade, ICT teacher's experience with Scratch, attitudes towards teaching programming. Additionally, it is also recommended that instruments for measuring perceived acceptance and use of other visual tools (Code.org, Alice etc.) that facilitate teaching programming should be developed, implemented and the results should be discussed.

As for decision-makers, it is suggested that the PAUSS-TPS should be applied to all ICT teachers teaching at secondary school level in Turkey and the contribution of Scratch software should be interpreted from the ICT teachers' perspective since they are given roles in formal curriculum of ICT and software course for using Scratch software and for introducing programming to the students. It is believed to help making decisions concerning adding or omitting activities about this software to/from the programming curriculum at secondary school level. It is suggested researchers study acceptance and use of Scratch and other block-based programming tools to describe the teachers' acceptance levels and make suggestions for better tools and increased effectiveness in teaching programming. The results of such studies help practitioners notice their colleagues' behaviors for acceptance and use of Scratch and can enable them to compare visual programming tools for teaching programming at early ages. The final form of the PAUSS-TPS consists of 28 items and a single factor, as seen in Appendix 1, is thought to be a useful measurement instrument for practitioners, researchers, and decision makers.

Acknowledgements

The present study was produced from the first author's master thesis and was supported by Ege University Scientific Research Projects Coordination Unit. Project Number: 15-EĞF-006.

References

Akpınar, Y. & Altun, Y. (2014). The need for programming education at schools of information society. *Elementary Education Online, 13*(1), dy:1-4.

Büyüköztürk, Ş. (2017). *Sosyal Bilimler için Veri Analizi El Kitabı: İstatistik, Araştırma Deseni, SPSS Uygulamaları ve Yorum [Handbook for Data Analysis in Social Sciences: Statistics, Research Design, SPSS Practices and Interpretation]*. Ankara: Pegem Academy.

Code.org (2018). About us. Retrieved 01.05.2019 from https://code.org/international/about

Choi, H. (2013). Pre-service Teachers' Conceptions and Reflections of Computer Programming using Scratch: Technological and Pedagogical Perspectives. *International Journal for Educational Media and Technology*, *7* (1), 15-25.

Chao, C. M. (2019). Factors Determining the Behavioral Intention to Use Mobile Learning: An Application and Extension of the UTAUT Model. *Frontiers in Psychology, 10.* http://dx.doi.org/10.3389/fpsyg.2019.01652

Çakıroğlu, Ü., Sarı, E. & Akkan, Y. (2011). The View of the Teachers About the Contribution of Teaching Programming To The Gifted Students In The Problem Solving, *5th International Computer & Instructional Technologies Symposium*, (September 22-24), Elazığ: Fırat University.

Çam, H. (2012). Determining the applicability of cloud computing technology in Turkish universities with the Technology Acceptance Model approach. (Unpublished doctoral thesis). Ataturk University, Erzurum.

Çatlak, Ş., Tekdal, M.& Baz, F. Ç. (2015). Scratch Yazılımı ile Programlama Öğretiminin Durumu: Bir Doküman İnceleme Çalışması (The Status of Teaching Programming with Scratch: A Document Review Work). *Journal of Instructional Technologies & Teacher Education, 4*(3), 13-25.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.* 13:319. http://dx.doi.org/ 10.2307/249008

Dönmez, O. & Akbulut, Y. (2019). Modelling teachers' acceptance of children's internet use: A risk-focused inquiry. *The Social Science Journal, 56*(4), 518-529, http://dx.doi.org/ 10.1016/j.soscij.2018.08.014

Durak, H., Karaoğlan Yılmaz, F. G., Yılmaz, R. & Seferoğlu, S. S. (2017). Erken Yaşta Programlama Eğitimi: Araştırmalardaki Güncel Eğilimlerle İlgili Bir İnceleme [Early Programming Education: An Investigation About Recent Trends in Studies]. B. Akkoyunlu, A. İşman and H. F. Odabaşı (Eds.) In *Eğitim Teknolojileri Okumaları 2017 [Educational Technology Readings 2017]*, 205-236.

Ersoy, H., Madran, R. O., & Gülbahar, Y. (2011). A Model Proposed for Teaching Programming Languages: Robotic Programming. XIII. *Academic Computing Conference*, 731–736, Malatya.

Esen, M. & Büyük, K. (2014). An investigation of electronic document management system in the context of
technology acceptance model: Case of the Council of Higher Education. *Dumlupınar University Journal of Social Sciences, 42,* 313-325. Retrieved from https://dergipark.org.tr/tr/pub/dpusbe/issue/4784/66004

Ferrer-Mico, T. and Prats Fernandez, M. A. & Redo-Sanchez, A. (2012). Impact of Scratch Programming on Students' Understanding of Their Own Learning Process. *Procedia - Social and Behavioral Sciences,* 46, 1219-1223.

Fesakis, G. & Serafeim, K. (2009). Influence of the Familiarization with "Scratch" on Future Teachers' Opinions and Attitudes about Programming and ICT in Education. *ITiCSE'09*, July 6–9, 2009, Paris, France.

Genç, Z. & Karakuş, S. (2011). Learning by design: Using Scratch for developing educational computer games. *5th International Computer & Instructional Technologies Symposium*, Elazığ, Turkey.

Gomes, A. & Mendes, A. (2007). Learning to program - difficulties and solutions, *International Conference on Engineering Education – ICEE 2007*, Coimbra, Portugal.

Gonzalez, C. (2013). *Student Usability in Educational Software and Games: Improving Experiences*. USA: IGI Global.

Gülmez, I. (2009). The effect of using visual tools on learner achievement and motivation in programming instruction. (Unpublished master thesis). Marmara University, İstanbul.

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

Kasalak, İ. & Altun, A. (2018). Blok Temelli Programlamaya İlişkin Öz Yeterlik Algısı Ölçeği Geliştirme Çalışması: Scratch Örneği. *Eğitim Teknolojisi Kuram Ve Uygulama, 8*(1), 209-225.

Kert, S.B. & Uğraş, T. (2009). Simplicity and fun in programming education. *1st International Educational Studies Congress,* Çanakkale, Turkey.

Kobsiripat, W. (2015). Effects of the Media to Promote the Scratch Programming Capabilities Creativity of Elementary School Students. *Procedia-Social and Behavioral Sciences,* 174, 227-232.

Kyriazos, T. A., & Stalikas, A. (2018). Applied Psychometrics: The Steps of Scale Development and Standardization Process. *Psychology, 9,* 2531-2560. https://doi.org/10.4236/psych.2018.911145

Kukul, V. & Gökçearslan, Ş. (2014). Investigating The Problem Solving Skills Of Students Attended Scratch Programming Course. *8th International Computer & Instructional Technologies Symposium*, Trakya.

Malan, D. J. & Leitner, H. H. (2007), Scratch for Budding Computer Scientists, *SIGSCE'07*, Covington, KY, 223-227.

Meerbaum-Salant, S., Armoni, M. & Benari, M. (2013). Learning computer science concepts with Scratch. *Computer Science Education*, *23*(3), 239-264. https://doi.org/10.1080/08993408.2013.832022

Ministry of National Education (MoNE). (2018). Bilişim Teknolojileri ve Yazılım (Information Technology and
Software). Retrieved 01.02.2020 from
http://mufredat.meb.gov.tr/ProgramDetay.aspx?PID=374

Mueller, J., Wood, E., Willoughby, T., Ross, C., & Specht, J. (2008). Identifying discriminating variables between teachers who fully integrate computers and teachers with limited integration. *Computers & Education, 51*(4), 1523-1537. http://dx.doi.org/ 10.1016/j.compedu.2008.02.003

Oluk A., Oluk, A.H. (2019). Blok Tabanlı Programlama, Korkmaz Ö. (Eds) In *Programlama Öğretimi Yaklaşımları*, 978-605-7846-90-7. Nobel, pp.69- 89.

Ortiz-Colon, A., & Maroto Romo, J. (2016). Teaching with Scratch in Compulsory Secondary Education. *International Journal Of Emerging Technologies In Learning (IJET), 11*(02), 67-70. doi:http://dx.doi.org/10.3991/ijet.v11i02.5094

Özer, P. S., Eriş, E. D. & Timurcanday Özmen, Ö. N. (2012). An Integrative Model Proposition On Behavioral Factors Affecting Intention of Use In Information Technologies Implications. *Dokuz Eylul University Faculty of Economics and Administrative Sciences Journal*, 27, 94-114.

Saltan, F. & Kara, M. (2016). ICT Teachers' Acceptance of "Scratch" as Algorithm Visualization Software. Higher Education Studies, 6 (4), 146-155. doi:10.5539/hes.v6n4p146

Serçemeli, M. & Kurnaz, E. (2016). Investigation of Using Information Technology Products with Technology Acceptance Model (TAM) in Auditing. *Istanbul University Journal of the School of Business, 45*(1), 43-52. Retrieved from https://dergipark.org.tr/tr/pub/iuisletme/issue/30530/330269

Streiner, D. L., Norman, G. R., & Cairney, J. (2015). *Health Measurement Scales: A Practical Guide to Their Development and Use* (5th ed.). Oxford, UK: Oxford University Press. https://doi.org/10.1093/med/9780199685219.001.0001

Šumak, B., & Šorgo, S. (2016). The Acceptance and Use of Interactive Whiteboards Among Teachers: Differences in UTAUT Determinants Between pre-and Post-Adopters. *Computers in Human Behavior, 64,* 602–620. http://dx.doi.org/ 10.1016/j.chb.2016.07.037

Tanrıkulu, E., & Schaefer, B. C. (2011). The users who touched the ceiling of scratch. *Procedia – Social and Behavioral Sciences, 28,* 764 – 769.

Tashakkori, A. & Teddlie, C. (2003). *Handbook of Mixed Methods in Social & Behavioral Research*. Thousand Oaks: Sage.

Teo, T. (2015). Comparing pre-service and in-service teachers' acceptance of technology: Assessment of measurement invariance and latent mean differences. *Computers & Education*, *83* (2015), 22-31. http://dx.doi.org/10.1016/j.compedu.2014.11.015

Turan, B. & Haşit, G. (2014). Technology Acceptance Model and An Implementation on Elementary School Teachers. *International Journal of Alanya Administration Faculty*, *6*, 109-119.

Ursavaş, Ö. F. (2014). Modeling and examining teachers' ICT acceptance (Unpublished doctoral thesis). Gazi University, Ankara.

Van Der Heijden, H.R.M.A., Geldens, J.J.M., Beijaard, D. & Popeijus, H.L. (2015). Characteristics of teachers
as change agents, *Teachers and Teaching, 21*(6), 681-699. http://dx.doi.org/10.1080/13540602.2015.1044328

Venkatesh, V., Morris, M. G., Davis, G. B., and Davis, F. D. (2003). User acceptance of information
technology: toward a unified view. MIS Quarterly, *27*(3), 425–478.

Yükseltürk, E. & Altıok, S. (2015). Bilişim Teknolojileri Öğretmen Adaylarının Bilgisayar Programlama Öğretimine Yönelik Görüşleri (Pre-Service Information Technologies Teachers' Views on Computer Programming Teaching). *Amasya Education Journal, 4*(1), 50-65.

Yükseltürk, E. & Altıok, S. (2016). Pre-Service Information Technology Teachers` Perceptions about Using
Scratch Tool in Teaching Programming. *Mersin University Journal of the Faculty of Education*, *12*(1), 39-52. http://dx.doi.org/10.17860/efd.94270

Wilson, A. & Moffat, D. C. (2010). *Evaluating Scratch to introduce younger schoolchildren to programming,* 1–12. Retreived 10.02.2018 from http://scratched.media.mit.edu/sites/default/files/wilson-moffat-ppig2010-final.pdf

Zhang, L. & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education, 141* (2019), 1-25. https://doi.org/10.1016/j.compedu.2019.103607

Appendix A. Perceived Acceptance and Use of Scratch Software for Teaching Programming Scale

Dear teacher, you are kindly invited to the study titled “Perceived Acceptance and Use of Scratch Software for Teaching Programming”. Before making your decision whether to participate in this study or not, you need to know why and how the study will be carried out. In this respect, it is highly important that this form is well-read and understood. Please feel free to ask us if you happen to fail to understand anything and if there are any unclear points, or you wish to get further information. Participation in this study is completely **voluntary**. You hold the right to **not participate** in the study or to **leave** the study at any point after participation. **Your response to the study** will be evaluated as **your consent for participation in the study**. Please do not get under anybody’s pressure or suggestion while responding to the questions on the **forms** given to you. The personal information to be obtained from these forms will be kept completely confidential and will only be used for study purposes. Please read the statements given below. Mark your degree of agreement to the statements with an X. Thank you.

	Totally	gree	ndecided	isagree	Totally
1. Drag-and-drop method facilitates the use of Scratch.					
2. Turkish language support facilitates the use of Scratch.					
3. Grouping of commands in menus facilitates the use of Scratch.					
4. Testing code blocks quickly facilitates the use of Scratch					
5. Scratch increases course efficiency.					
6. Scratch facilitates coding instruction.					
7. Scratch enriches the course.					
8. Scratch improves students’ problem-solving skills.					
9. Scratch improves students’ algorithm skills.					
10. I enjoy teaching Scratch.					
11. Using Scratch is interesting.					
12. I enjoy taking interest in Scratch.					
13. I will use Scratch in my future courses too.					
14. I will suggest Scratch to others.					
15. I think I will use Scratch frequently.					
16. Students support my use of Scratch.					
17. I can develop different applications using Scratch.					
18. I am self-confident in using Scratch.					
19. If I have a problem using Scratch, I can solve it myself.					

20. I use Scratch because it is a free software tool.					
21. It is easy for students to understand Scratch.					
22. Students are willing to learn Scratch.					
23. It is fun to use Scratch.					
24. I enjoy having classes Scratch.					
25. I like using Scratch.					
26. Scratch is an appropriate program for course objectives.					
27. Scratch is a suitable program for students' level.					
28. Scratch is important to my profession.					

www.ijcses.org