Journal of Computer and Education Research
March 2024 Volume 12 Issue 23 (ISSN:2148-2896)
http://dergipark.org.tr/jcer

Research Article

A Qualitative Case Study:Pre-service Teachers as Novice Programmers

Burcu ŞENER [1], Duygu UMUTLU [2*]

[1] *Boğaziçi University, İstanbul, Turkey, burcu.sener@bogazici.edu.tr*
[2] *Boğaziçi University, İstanbul, Turkey, duygu.umutlu@bogazici.edu.tr*
[*] *Corresponding Author: duygu.umutlu@bogazici.edu.tr*

Article Info

Received: *12 February 2024*
Accepted: *19 March 2024*

Keywords: Computer science education, programming skills, teacher education, computational thinking

10.18009/jcer.1435182

Publication Language: English

Abstract

To meet the needs of 21st-century learners in today's classrooms, it is needed that teachers be familiar with programming and computational thinking. Particularly, subject-area pre-service teachers should be exposed to programming instruction in their teacher education programs. This case study including three participants aims to explore the process of pre-service teachers' learning of programming while completing CT-oriented tasks through observations and interviews in a 14-week educational technology course at a public university in Turkey. The findings demonstrate that pre-service teachers, being novice programmers, prefer contextualized, structured and visually well-designed programming tasks. They use various strategies to face challenges, and the effort they put into dealing with these challenges enables them to produce higher-quality programs. Accordingly, implications for further research are also discussed in this study.

To cite this article: Şener, B., & Umutlu, D. (2024). A qualitative case study: Pre-service teachers as novice programmers. *Journal of Computer and Education Research, 12* (23), 292-318. https://doi.org/10.18009/jcer.1435182

Introduction

Computational Thinking (CT) is considered among the key skills required in the 21st century. CT is defined as is a problem-solving process that involves four fundamental phases: Decomposition, pattern recognition, abstraction, and algorithmic thinking (Wing, 2006). According to the standards set forth by the International Society for Technology in Education (ISTE), CT stands as a vital skill for students within 21st-century classrooms, as well. Therefore, educators must acquaint themselves with CT to effectively address the needs of their students (ISTE, 2016). However, the predominant focus of existing CT teacher education initiatives tends to center solely on computer science educators (Yadav et al., 2014). Consequently, it is advisable to revise teacher education curricula to empower educators across all disciplines with CT skills.

As articulated by Passey (2017), CT serves as a problem-solving framework, and programming can be a means to cultivate CT skills. In many countries, programming is

increasingly recognized as a fundamental skill and is being integrated into curricula as either a mandatory or optional course, particularly in K-12 settings (Hanbay-Tiryaki & Balaman, 2021). Hence, it could be argued that novel approaches to teacher education are necessary to prepare aspiring educators for teaching programming effectively (Instefjord & Munthe, 2017). In order to train pre-service teachers in this regard, it is also necessary to gain a deeper understanding of what kind of processes they, as novice programmers, go through in CT contexts through programming tasks.

There are many quantitative (Cutumisu et al, 2021; Çiftçi & Topçu, 2022; Jin & Cutumisu, 2023; Kaya et al., 2019; Yadav et al., 2014; Zha et al., 2020) and qualitative studies (Butler & Leahy, 2021; Dağ, 2019; Hunsaker & West, 2020; Piedade et al., 2020; Umutlu, 2022) having investigated the place of CT in teacher education in the literature. Most quantitative studies in the literature investigated effective ways to introduce CT concepts to pre-service teachers. Relevant qualitative studies explored pre-service teachers' understanding, perceptions about, and attitudes toward CT.

Almost all of the qualitative studies in the literature either rely mostly on document analysis of written accounts of pre-service teachers (i.e. Butler & Leahy, 2021), or their designed learning activities, lesson plans, or projects as data collection sources (Dağ, 2019). This case study aims to examine the process of pre-service teachers' learning of programming skills through CT-oriented coding tasks within the context of a 14-week required educational technology course. The central research question is "How does pre-service teachers' learning process of programming unfold in an educational technology course?". The sub-questions are: (a) What are their approaches toward CT-oriented programming tasks? (b) What kind of challenges do they experience during CT-oriented programming tasks? and (c) How do they deal with the challenges they experienced in CT-oriented programming tasks?

Literature Review

Computational Thinking

Although he did not provide an operational definition for it back then, Papert (1980) sowed the seeds of CT as a concept, by describing how children can benefit from programming and become "epistemologists", through "thinking about thinking" (p.19). Wing (2006) defines CT as "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (p.33). Brennan and

Resnick (2012) also created a CT framework that consists of three facets: 1) *Computational concepts*, 2) *Computational practices*, and 3) *Computational perspectives*. Computational concepts refer to concepts used by programmers, such as sequences, loops, parallelism, events, conditionals, operators, data. Computational practices involve testing and debugging, reusing and remixing, abstracting and modularizing. Finally, perspectives are about how a programmer's perspective toward the world changes through programming, and these are expressing, connecting, and questioning. It should be noted that programming and CT are not used synonymously in the framework; however, programming is one of the contexts where CT can be sufficiently exemplified (Brennan & Resnick, 2012). As this study aims to understand how pre-service teachers' learning of programming by completing CT-oriented tasks occurs, the next section presents the studies in the literature conducted with pre-service teachers.

CT Interventions and Programming Instruction in Teacher Education in Teacher Education

Forms of CT interventions for teacher education used in the previous studies can be categorized under two main headings: Fully CT-oriented courses or trainings, and CT modules integrated into educational psychology or educational technology courses. Regardless of their modes, the CT interventions in the research are generally composed of two main parts: Introduction of CT basics, concepts and practices, and designs of lesson plans or a set of learning activities. Different CT practice platforms and tools were used, including Scratch, LOGO, Basic, Alice 3, and robotics. Some interventions were directed at pre-service teachers studying at different educational departments (Umutlu, 2022), some were directed at Informatics majors (Piedade et al., 2020), and some to STEM teachers (Çiftçi & Topçu, 2020).

Several studies that incorporated a stand-alone CT training or a course oriented towards CT were identified (Albayrak & Özden, 2021; Bal et al., 2022; Butler & Leahy, 2021; Dağ, 2019; Kaya et al., 2019; Mouza et al., 2017; Umutlu, 2022). Their duration varied from 7 to 15 weeks. Most of the studies were conducted within the scope of the courses in the faculties of education. Some other studies (i.e., Ateşkan & Hart, 2021; Çiftçi & Topçu, 2022; Hunsaker & West, 2020; Mugayitoglu, 2016; Piedade et al., 2020; Yadav et al., 2014, Zha et al., 2020) were conducted in CT modules integrated into different educational courses. The duration of the intervention ranged from one week to three weeks.

In almost all the reviewed studies, there is a reference to the effectiveness of the CT intervention applied with pre-service teachers, and most of them measured this effectiveness through more than one outcome. These are pre-service teachers' CT abilities, skills, pedagogical knowledge, and their self-efficacy in teaching in terms of CT. How pre-service teachers conceptualize CT and its practices, components, and aspects is another common investigation (i.e. Butler & Leahy, 2021; Çiftçi & Topçu, 2022; Mouza et al., 2017; Umutlu, 2022). Pre-service teachers' perceptions towards the implementation of CT itself or CT as a pedagogical tool are also explored (i.e. Kaya et al., 2019, Mugayitoglu, 2016; Piedade et al, 2020).

The findings of these studies demonstrate that CT interventions – either full courses or short trainings- can yield an improvement in pre-service teacher's CT and programming skills (i.e., Ateşkan & Hart, 2021; Butler & Leahy, 2021; Umutlu, 2022). However, integration of CT into pedagogical knowledge and skills (Bal et al., 2022), and forming accurate conceptions regarding CT (Mouza et al., 2017) were found to be areas of struggle although the attitudes of pre-service teachers toward CT interventions were mainly positive (Dağ, 2019; Kaya et al., 2019; Mouza et al., 2017).

Jin and Cutumisu (2023) used different machine-learning methods to predict the pre-service teachers' CT skills (n=93). The intervention was a CT training module on Scratch and they used CCTt, a perception survey and a demographic survey to collect data. According to the findings, the best-performing Decision Tree method yielded that three factors (time spent in training, prior CT abilities, and perceptions of difficulty) can predict their CT skills. These two model-building studies show us that pre-service teachers' attitudes, time spent in training, and perceptions of difficulty are among the factors predicting their CT skills. Therefore, it is important to understand the challenges pre-service teachers experience while learning programming in the context of CT and take their opinions about these into consideration to be able to design better learning environments.

Programming serves as a potent tool to bolster the CT skills of learners as they are expected to use computer science principles while coping with programming challenges (Kazımoğlu et al., 2012; Lye & Koh, 2014; Qualls & Sherrell, 2010). Programming encompasses a series of commands directed towards a computer to devise solutions for coding challenges using a programming language (Lye & Koh, 2014). Proper coding in a programming language necessitates problem comprehension, algorithm formulation, and

algorithm implementation (Kazımoğlu et al., 2012). Moreover, programming should be integral to K-12 education so that students can acquire 21st-century skills (Çoklar & Akçay, 2018).

Efforts have been made by global initiatives to integrate CT into K-12 curriculum in several countries (Buitrago Flórez et al., 2017). Furthermore, several teacher education programs include computer science courses to help future educators understand how CT aligns with their disciplines (Yadav et al., 2014). Nevertheless, teacher education curricula fall short in adequately teaching CT through programming (Yadav et al., 2014). Additionally, studies (e.g., Denning, 2017; Grover & Pea, 2013) highlight teachers' insufficient knowledge in effectively integrating CT into K-12 learning environments, attributable to the dearth of programming courses in teacher education programs (Burden, et al., 2016). Revising teacher education curricula and introducing novel approaches to programming education are imperative to address this deficiency (Instefjord & Munthe, 2017).

Relevant literature also shows that even though the existing CT modules or courses designed for pre-service teachers have accomplished a certain level of improvement in CT skills, it is vital to tailor such interventions according to the further needs of pre-service teachers so that they can benefit from them best. Therefore, there is a need to fully understand the perceptions, challenges and the strategies employed by pre-service teachers during programming CT-oriented tasks. Further exploration of pre-service teachers' learning of programming and CT by eliciting their contextualized experiences in their learning environments can contribute to research and help with designing better CT interventions that include programming instruction.

Use of Scratch as a Programming Tool in Teacher Education

Several programming tools have emerged specifically catering to beginner programmers. Designed and developed as a visual programming language by the Massachusetts Institute of Technology (MIT) Media Lab in 2008, Scratch aims to make programming accessible for children, adolescents, and novices. Visual programming languages, such as Scratch, facilitate the learning process for inexperienced programmers by reducing the complexity through visual elements, as noted by Maloney et al. (2004) and Wakil et al. (2019).

According to Maloney et al. (2004), Scratch empowers programmers to create multimedia projects incorporating graphics, animation, music, and sound. Moreover, as Ilic

(2021) highlights, Scratch enables the creation of video games, animations, and interactive stories. Scratch also supports the development of various programming activities that engage learners and contribute to the enhancement of their CT skills (Fagerlund, et al., 2021). As the current study focused on learning programming and CT of pre-service teachers, who are inexperienced programmers, Scratch was found to be appropriate to use in the context of the study.

Method

This qualitative study employed the case study design (Yin, 2014). A case study is suitable for understanding a phenomenon in a bounded system. As Flyvbjerg (2011) defines, "Case study is an intensive analysis of an individual unit (as a person or community) stressing developmental factors in relation to environment" (p. 301). Since the aim of this study is to explore a particular developmental phenomenon and process, which is pre-service teachers' development of programming and computational thinking skills, an educational technology course offered to pre-service teachers has been chosen as the case. By delving into the details of this learning environment using a case study approach, it is possible to draw a better, contextualized picture by setting boundaries within a single course. Also as Merriam (2009) states, case studies are particularistic. That is, that particular case is important for what it can reveal. Since the case is a course oriented at programming and CT skills, taken by novice coders, it provides necessary information in order to understand the phenomenon.

Three participants were selected as the sample of the study as they were regarded as the key informants while exploring pre-service teachers' development of CT and programming skills: Yvonne, Kyle, and Alice. All names are pseudonyms. The criteria for the sampling were pre-service teachers' having no programming experience and completion of all programming tasks.

Participants and Context

Upon obtaining the ethics committee approval from the university where the study was conducted, participants were informed about the study verbally and in written form. Out of 22 pre-service teachers in the course, eight agreed to participate in the study and granted their consent. Through maximum variation sampling (Merriam, 2009), which is a type of purposeful sampling, three participants were selected for this study. As maximum

variation method for sampling in this study was used to detect different perspectives, increase the variation in the findings and trustworthiness of the study (Creswell, 2013), the three participants were selected from different departments: Math and Science Education, Foreign Language Education, and Chemistry Education. Two criteria were used for purposeful sampling: a) pre-service teachers' having no experience in programming and b) their completion of all programming tasks. By using these criteria for purposeful sampling, it was ensured that the participants were the key informants for this case study. Profiles of the participants, prepared using the data coming from the observations and the interviews, can be found below. They were from different departments, and they had varying levels of motivation toward the course, as the course instructor and the researchers' observational notes indicated.

Yvonne.

Yvonne was a 22-year-old female, studying Mathematics and Science Education at the time of the study. She had no previous coding experience. She had some experience with digital drawing software, but she said that this did not make her a programmer. She believed that by using her knowledge of Scratch, she would be able to design better instructional games on more complex platforms in the future. She was quite articulate with her specific experiences on Scratch. She was able to accurately describe the functioning of CT concepts she used such as loops, conditionals, and sequencing. When she was working on a project, she used CT practices such as abstraction and being incremental. Sometimes she struggled findings ideas during Scratch tasks requiring creativity, yet she usually finished the tasks quite fast. The programs she created throughout the course met the task requirements with adequate complexity. She was able to include the target code blocks of the week in her designs properly and thus demonstrated her CT skills to produce functional projects.

Kyle.

Kyle was a 22-year-old male, studying in the Department of Foreign Language Education when the study was conducted. He did not have any previous coding experience. He had a solid understanding of the role of technology in instruction. He acknowledged the fact that his future students not only would be more competent in technology but also would demand the use of it in classrooms. He was aware that he should be technologically

competent as a teacher. Like Yvonne, he was able to correctly describe how CT concepts work on Scratch. He mentioned that he completed the tasks quickly in the classroom since he did not experience creative difficulties. Although he was an early finisher, when the programs he produced were examined, it was found that sometimes they were too simplistic, although they functioned smoothly.

Alice.

Alice was a 24-year-old female, studying in Chemistry Education at the time of the study. Like Yvonne and Kyle, she had no previous coding experience. She was older than her classmates and had a more loaded course schedule. She was fairly good at describing CT concepts, but sometimes she had trouble remembering how exactly they work. She frequently referred to being incremental and trial and error when she felt stuck during programming tasks. Sometimes she could not finish them during class time, and she needed to complete them at home, which made her feel bored. She also missed some of the classes and this classroom attendance issue might have hindered her CT improvement. It was observed in the classes that certain misconceptions or gaps persisted in terms of her coding knowledge and skills, which also revealed itself through her appeal for the instructor's help about basic CT concepts, such as sequences or loops in lab sessions in the later weeks of the semester.

The context of the study was an educational technology course compulsory for 3rd and 4th-year pre-service teachers from different departments in a public university in Turkey. In a 14-week semester, two class hours (50 minutes each) of discussion sessions, and two class hours (50 minutes each) of laboratory sessions were held on separate days every week. The focus of discussion sessions was to develop an understanding of the theories and practices for the use of technology when designing instructional materials. The laboratory sessions focused on the practice of these ideas using Scratch, which is a block-based programming platform. And, the data of the study were collected during the laboratory sessions of the course.

The curriculum implemented in the lab sessions of the course was adapted from Getting Unstuck Curriculum, a Scratch curriculum developed by the Creative Computing Lab at the Harvard Graduate School of Education (2021). In a typical laboratory session of

the course, the instructor introduced the CT concept/Scratch code of the week. Pre-service teachers completed two "Explore", and two "Create" tasks (see Table 1).

Table 1. Task descriptions

Task	Length	Description
Explore 1	~20 mins	Students are provided with example Scratch projects including the target code(s) of the week, and they are expected to "explore" how they work.
Explore 2	~20 mins	Students are given image(s) of code blocks, and they are expected to "read" them and post their answers on the discussion forum on the classroom Moodle page.
Create 1	~30 mins	Students are provided with a Scratch project which does not work properly, and they are expected to detect the problems, and debug them.
Create 2	~30 mins	Students are expected to create their own mini projects including the target code(s) of the week.

In Explore 1, they examined one or two Scratch projects with target codes (When Clicked, Loops…) and how the codes were used. In Explore 2, pre-service teachers deciphered the functions of codes looking a screenshot of another project (see Figure 1). In Create 1, they were given an ill-structured problem on Scratch and they made it work by remixing it (see Figure 2). Finally, in Create 2, they created their own short projects by using the target code of the week. They were given time until the end of the day if they could not finish the last task within the class hour. When they finished, they posted the link to their Scratch projects, and their projects were assessed with a pass/fail system.

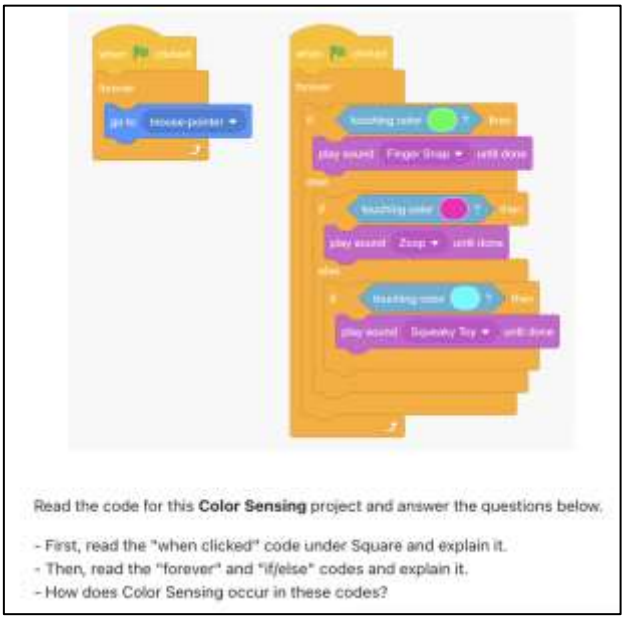

Figure 1. Explore 2 example task

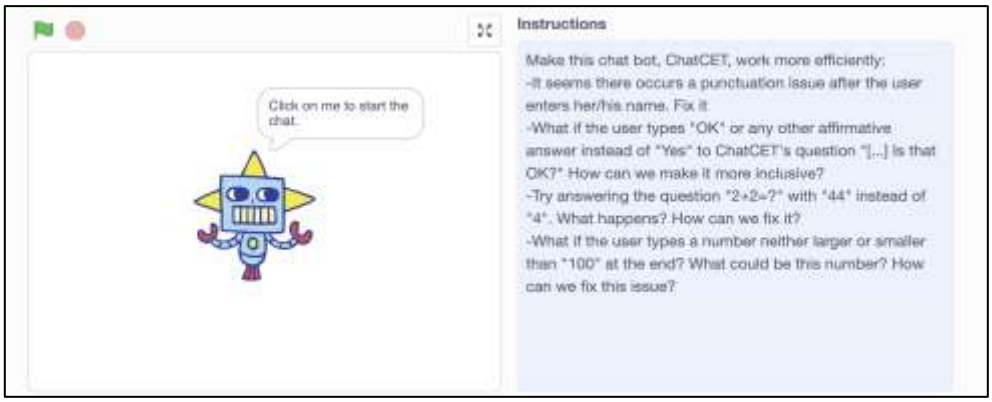

Figure 2. Create 1 example task

Data Collection Procedures

Data for this study were collected from two main sources over 14 weeks. The first author observed all the lab sessions where participants completed programming tasks on Scratch and took notes. Each of the 14 weekly lab sessions took approximately 100 minutes. The second author, who was also the course instructor, took notes of critical instances during the lab sessions. Based on the observation notes, the quality of participants' lab task completion, and the area of their teaching, three participants were invited for semi-structured qualitative interviews in the last week of the study. The first author conducted three semi-structured interviews each of which took about 40 minutes with the three participants in April 2023. Interview questions were prepared based on the research questions of the study, with the goal of examining participants' learning process of programming and CT as a whole. The semi-structured interviews included questions and prompts, such as "Describe a typical lab session of this course to me" or "What do you do when you're stuck in a task?" (see Appendix A for the full list of interview questions). Additionally, Create 1 and Create 2 tasks of the participants were collected. In total, the dataset of the study included observation notes of important situations from 1400-minute lab sessions, three interviews, and programming tasks.

Data Analysis

Thematic analysis was employed to analyze the dataset of the study. Audio recordings of the interviews were transcribed through an automatic speech recognition system called *Whisper* (OpenAI, 2022). First, the interviews analyzed through open coding

(Creswell, 2013) on MAXQDA (VERBI Software, 2023) to see emerging salient patterns from the data. Then, the whole dataset (interviews, observations, and programming tasks) was examined through focused coding (Saldana, 2013) based on the CT framework (Brennan & Resnick, 2012). In the first cycle of coding, the authors analyzed one interview independently and then discussed the codes they drafted. Upon reaching a consensus, the first author drafted a coding scheme. The second author provided peer debriefing on the codebook. Based on the agreed-upon codebook, the first author analyzed the other two interviews and observation notes. After three cycles of coding and revisions, 4 categories and 13 codes were extracted, which can be seen in Table 2. The second author provided a review on the analysis. From those categories, the authors together crafted three themes. External peer experts who were not involved in the process of data analysis were also consulted for review.

Trustworthiness

To ensure trustworthiness of the study, the observational data were collected of a rather long period of time, which is 14 weeks. Also, three key informants with a varying observed classroom engagement were selected for interviews, and their programming tasks were also analyzed for the purpose of data triangulation. During the data analysis, two researchers coded the data independently and then came together to reach a consensus on the codes and categories to ensure researcher triangulation (Lincoln & Guba, 1985). Afterwards, the authors formed the themes based on the data analysis. Finally, peer debriefing (Cooper, et al., 1998) over all the codes, categories, and themes was given by an external peer expert, who was well-versed in computer science education.

Findings

Analysis of the dataset yielded the following themes: (1) Pre-service teachers, as novice coders, prefer more structured and visually well-designed tasks that would encourage them to produce more codes, (2) How much they struggle through creative and cognitive challenges seems to determine the quality of pre-service teachers' programming, and (3) Being novice coders, pre-service teachers rely mostly on external resources, trial-and-error or remixing to overcome different types of challenges they encounter during programming.

Theme 1 – Ways of Being Productive during Programming

Findings of the current study indicate that when the tasks were found interesting and visually well-designed by the participants, they became willing to be involved in coding more. In the interviews, participants expressed their opinions regarding the laboratory sessions and the programming tasks they completed. For instance, Yvonne emphasized that when the initial tasks (Explore 1, 2, and Create 1) were not well-structured and interesting enough, she got lost in the last task (Create 2) and did not know what to do to complete it properly:

> As I said, sometimes *Create 1* tasks are not so good, sometimes they are not so interesting. In this case, I don't know how to complete *Create 2*. For example, sometimes I ask myself 'How should I use [this code]?' […] But that *Question-Bot* task was really nice, to be honest. It was quite interesting. It was well thought.

Yvonne also added that she found Create 2 tasks more fun as they were free to create any projects as long as they used the target code of the week. Yet, in the cases that she did not have any idea about what her Create 2 project could be like, she struggled: "Create 2 can be more fun because we are more independent [while coding the programming task]. However, if I do not have any idea to present, it can be a problem".

When her Create 2 tasks were examined for the "Question-Bot" week, it was seen that her enjoyment of the Create 1 task was partially reflected on her Create 2 task. In that week, participants were supposed to complete two Create 1 and Create 2 tasks (Random and Ask & Answer). Although her first Create 2 task (Random) was quite simplistic and uncomplex - perhaps due to time constrains, she produced a well-designed program with a good narrative in her second Create 2 task (Ask & Answer). This can be associated with her willingness to complete the Create 1 task (Question Bot) of that target code. That is, as the task was structured and visually well-designed one, it might have encouraged Yvonne to produce well-designed program that includes several codes working properly for the following task (Create 2).

In addition, as Kyle stated at the interview, when the tasks did not require them to create new codes, but just to examine given ones, they were not very useful for participants' learning of programming.

> *Create* tasks are fun. We solve problems in one of them, and create something new in the other. But, in *Explore* tasks, we write something. We try to explain the codes in

> Scratch projects using our own words. For example, "This code was used here because of this". Yet, to me, it seems like a waste of time. Because I see [the codes] right there, but we need to write [what we see]. But it goes fast. Still, I don't know, they don't seem beneficial.

Even though Kyle found Create tasks more fun, he stated that he did not deem it necessary to put a lot of effort in his choice of sprites on Scratch. That is, he emphasized that he focused on the quality of coding rather than visual design of the tasks.

> I figured out the platform more or less. I know some sprites and almost always use those in my projects. Some of my classmates, for instance, find pictures, background images, and characters online. To be honest, I don't do it. I don't care about that aspect of the task.

His comments in the interview aligned with his coding performance in many of the Create 2 tasks he produced. When his Create 2 tasks were examined, it was found that most of his projects were either too simple, incomplete or did not include the target code of the week. Moreover, he used the same sprites as his characters in multiple projects, as he also mentioned in his interview (see Figure 3). Therefore, it can be concluded that he may not have put enough effort in his Create 2 tasks. That is, he aimed to just program the target code of the week properly, and he did not make an effort for creating visually well-designed programs that may have a context.

Figure 3. Kyle's three different create 2 tasks

Participants also emphasized that more structured tasks and examples would be helpful scaffolds for them during programming. For instance, Alice stated "Sometimes we can't apply the things we learned. […] If we were given more examples, different examples,

it would be easier." Similar to Alice, Yvonne highlighted that being completely free while completing programming tasks may sometimes be very challenging for novice coders:

> Perhaps there could be something that will make it easier in *Create 2.* I got everything at my hand in *Explore 1* and *Explore 2*. Similarly, [I have everything I need] in *Create 1.* At least, there are some ideas. Then, we directly move onto *Create 2*. We have nothing. We're on our own. It would be better if there were some clues. Well, actually we can get some ideas from *Create 1*, but still, [I struggle a bit].

As the quotations above show, participants expressed their willingness to complete the tasks that were visually well-designed, structured, and complex in their perception. When the programming task had a meaningful and motivating context, and the sprites and background images were consistent with each other, the participants demonstrated higher engagement with programming. It can be concluded that learners who are inexperienced in programming may value tasks more if they can clearly decipher the task requirement and its objective. For example, if the tasks involve problem-solving and debugging or require programming a contextual project such as a game or a story, then novice coders' engagement and motivation get higher. If they do not see the rationale behind the tasks, such as *Explore 2* which involves code reading, they might not develop a positive attitude.

Theme 2 – Challenges

Based on the analysis of the study dataset, it can be inferred that participants experienced technical, creative, and cognitive challenges during programming tasks. During their programming, technical problems were usually caused by the web-based platform of Scratch. For instance, Yvonne mentioned that sometimes Scratch platform did not allow the user to add sounds. Participants usually could not figure out how to overcome these technical problems as they did not have a control on the coding platform itself.

This situation was also apparent in the analysis of participants' Create 2 tasks. In the second week when the participants worked on Parallelism, Kyle and Alice fixed their programs successfully. However, Yvonne did not share the link for her program. Yet, instead she posted a comment in which she told that the sound was not working properly, and she could not make one of the sprites work at all. Such technical problems might create frustration and lead novice coders, like Yvonne, to give up during their debugging processes.

When it comes to creative and cognitive challenges, findings of the present study demonstrate that how much they struggle through these challenges seems to determine the

quality of pre-service teachers' programming. For instance, both Yvonne and Alice touched upon the creative and cognitive challenges they had during *Create 2* tasks. They struggled to find novel project ideas. To illustrate, Yvonne described how difficult for her to come up with a story where she could use the target code block of the week. This situation seems to have involved both a creative struggle, such as finding a meaningful story context and a cognitive challenge as she was expected to integrate the target code block into that story: "Sometimes it takes time to decide on your story and decide how to use the codes. 2 or 3 weeks ago, there was one time that I didn't know how to present that code and it took long for me to complete it. I submitted the assignment very late at night." Similarly, Alice acknowledged that she had experienced challenges: "Sometimes, I say 'OK' but actually it's not OK. It either works wrong or it doesn't work like I want it to […] And this challenges me, sometimes."

In Yvonne's case, it can be clearly seen that some of her tasks were creatively uncomplex. She usually preferred to design projects, which worked properly in terms of the target codes but had simple scripts. This was consistent with the comments she made in her interview where she mentioned that she sometimes struggled creatively.

Examination of Alice's Create 2 tasks yielded a quite interesting picture. She also mentioned her creative struggles in the interview; however, some of her projects were quite creatively designed with complex and meaningful scripts with visually well-chosen images. For instance, in her Key Press and Loops tasks, she used high-quality background pictures she searched and found outside the Scratch platform. The sprites she used were also usually consistent with the backgrounds. She was able to design functioning games, with an authentic narrative. For instance, in the Key Press task, she designed a game where the user controls the crab and catches the fish (see Figure 4). Therefore, even though she mentioned creativity as one of her biggest challenges, she was occasionally quite good at designing creative tasks.

Figure 4. Alice's key press task

On the other hand, her struggle in terms of creativity was visible in some of her tasks. For instance, in her Random task (see Figure 5), there were problems with the design. The basketball was larger than the basket, and there was an unnecessary sprite (X button) near the basket, which she used to write a conditional block for "if the ball touches X". Also, the cartoonish style of the basketball player did not align with the style of the realistic background. Given that Create 1 tasks were also more compelling compared to previous weeks and they were supposed to work on two target codes (Ask and Answer & Random) in that week, this cognitive challenge may have led her to allocate less time on creativity.

Figure 5. Alice's random task

In contrast, Kyle stated that he did not experience any creative backlash because he had already started to think about potential project ideas during *Explore 1, 2,* and *Create 1* tasks. And he stated he struggled only in one of the tasks:

> I usually complete the tasks in 10 or 15 minutes, because I usually have an idea before. […] Let me give you an example. Most probably, the most challenging task for me was *Question-Bot*. No other task challenged me more. It was hard to understand why some codes were working and others not. […] What I mean by struggle is that it took me 5 or 6 minutes. Normally it takes much shorter. But this task was a bit more complex.

On the other hand, as the analyses of his interview and Create 1 and Create 2 tasks showed, he may have not experienced any creative challenges because he did not aspire to be creative as his classmates did. While Yvonne and Alice spent longer time in the classroom, Kyle was quick to finish the tasks. However, when the quality of the programs they produced were examined, it was seen that even though Yvonne and Alice experienced more challenges, their *Create 2* tasks were more complex and had higher quality than the ones Kyle completed.

When it comes to the cognitive challenges that they experienced in Create 1 tasks, it can be concluded that although they were mostly able to address and fix the bugs in the programs, they had some issues from time to time. For instance, in the first When Clicked debugging task, Yvonne and Kyle did not set the original position for one of the sprites when green flag was clicked; therefore, the end program did not work properly. It might be due to the fact that this was the first week of Scratch tasks, and they might not have been capable of the basic working principles of the platform.

In the debugging tasks directed at the code categories of Loops, Broadcast, Color Sensing, Random and Ask & Answer, participants demonstrated partial success. For instance, in Loops task, Kyle did not use the Repeat block at all, which he was supposed to do in order to get rid of the repetitive code blocks (see Figure 6). When all the projects she completed were examined, it was clear that her conceptualization of Loops was limited to the Forever block (see Figure 7). That is, she never used the Repeat block in her projects.

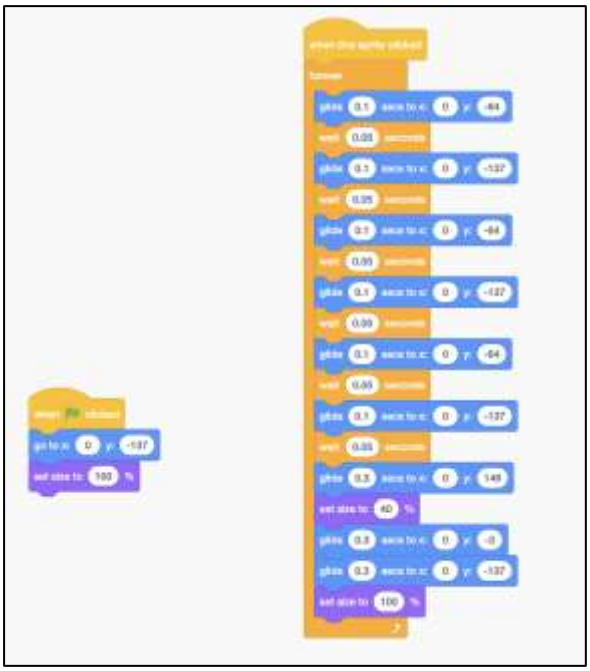

Figure 6. Kyle's loops task

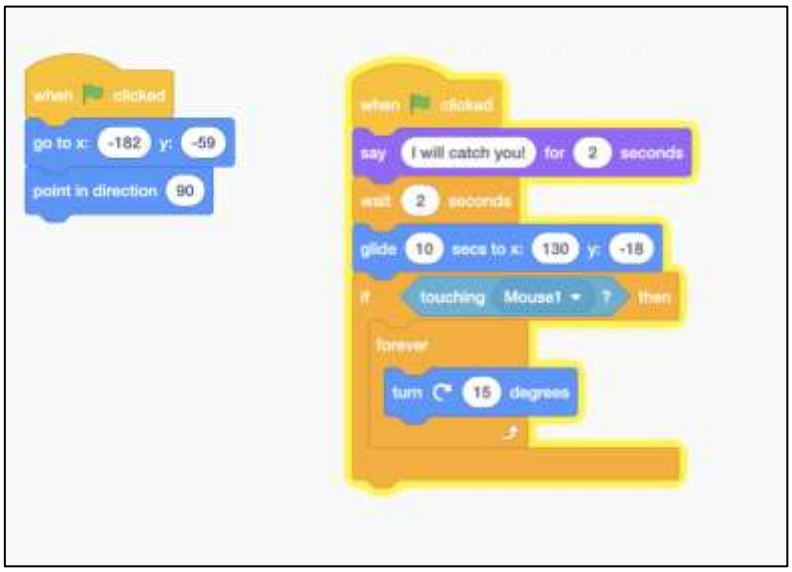

Figure 7. Alice's loops task

To give another example, Yvonne did not use any Broadcast blocks at all in the Broadcast task but relied on Wait blocks, which showed that she might have had misconceptions about how Broadcast block worked. However, in the following week's task (Color Sensing), she was able to produce properly-working Broadcast chains.

In Random and Ask & Answer tasks, participants fixed most of the codes, but none of their debugged programs were fully functioning. They either forgot to fix one of the bugs or simply did not know how to do, so they left it as it was. This situation might have been due to the fact that they were assigned to complete two complex tasks in one laboratory session

in that week, and they may have been overwhelmed by the cognitive challenges while programming of the tasks. Key Press, Variables, and Lists tasks were successfully completed by the participants. This might have stemmed from the fact that these three tasks required them to solve only one or two issues, unlike the previous tasks.

While dealing with programming tasks, the novice coders were expected to experience such creative and cognitive challenges. Findings of the current study show that while handling those challenges, participants usually tended to produce more codes, which resulted in more complex programs. In other words, cognitive and creative challenges seem to not only have a role in the time spent in task completion, but also in some cases, in the end product designed by novice programmers.

Theme 3 – Overcoming Challenges

As described above, participants experienced several challenges during programming tasks in a 14-week semester. To cope with these challenges, they employed different strategies, such as using external resources, trial-and-error while programming, or remixing of codes. For instance, they referred to in and out-of-class resources, and one of the coping strategies was appealing for the instructor's help as Yvonne mentioned: "Teacher, teacher! Can you help me? […] There were probably many times I pinned down the teacher." From time to time, especially for Create 2 tasks, Kyle sought for instructor's help and approval when he was not sure about whether the program he had created met the task requirements or not: "I asked questions to the lab assistant a few questions during Create 2: "Can you check my progress, does it look correct?". I ask questions when I doubt it, because we are graded on that."

When participants did not have time to ask for help from the instructor during the laboratory session, they preferred to resort to external resources, such as tutorial cards on Scratch website and other websites on the Internet. That is, using in or out of class materials during the challenges was another strategy for participants. As the interview data indicate, Kyle used Scratch tutorial cards while he was creating his own program for the Question-Bot task, whereas Alice preferred to find some external resources on the Internet and Yvonne watched tutorial videos about how to make codes work properly on Scratch.

Trial-and-error was also used frequently as a cognitive strategy during challenges, and this helped them figure out where the problem was in codes when they were stuck. Particularly, Yvonne used trial-and-error as a kind of debugging strategy: "Sometimes, I

watch somebody else's video, or read the codes step by step, again and again, until it works. I have to try until it works. I don't have any other choice anyway." Alice, on the other hand, used trial-and-error just to complete the task at hand: "I try out the codes. I complete the project by trying it out."

When participants had any cognitive and creative difficulty especially during a *Create 2* task, they also sometimes used remixing as a strategy to overcome the challenge. That is, they examined the given codes in *Create 1* and remixed them by adding some other codes to complete the *Create 2* task. From the observational notes, it can be seen that this is what Kyle mostly did to complete his *Create 2* tasks. He almost never produced a new program for *Create 2* tasks, but added a few more codes into the tasks he completed in *Create 1*.

As seen, referring to external resources, trial-and-error, and remixing were the common strategies participants utilized during the programming challenges. When the participants were stuck in the tasks, they asked for teacher's help, used Scratch tutorial cards, and took initiative and sought out out-of-class audiovisual contents. Also, when the programs they created did not work properly, they changed one part of the program to find out the piece that caused the disfunction and kept doing so until the program worked flawlessly. Remixing from previous programs was another strategy they employed particularly when they faced cognitive and creative challenges.

Discussion and Conclusion

This study explored the process of pre-service teachers' learning programming through CT-oriented tasks in a 14 week- educational technology course. Throughout the course, they examined computational concepts such as parallelism, loops and conditionals and used them in their own coding projects (Brennan & Resnick, 2012). They also applied computational practices such as testing, debugging, and remixing while programming the given tasks (Brennan & Resnick, 2012).

As the data obtained from interviews and observations show, sometimes the attitudes of novice coders toward programming tasks and in accordance with it, the time they spend on these tasks can influence the quality of their end products. For instance, the programs Yvonne spent more time on the tasks and created throughout the course were much more complex compared to Kyle, who mostly had a lower engagement with the tasks as stated above. It should be noted, however, that the time spent on tasks is not the sole predictor of programming skills. In Alice's case, even though sometimes she spent quite a lot of time

during the tasks, the end product was not always of high quality in terms of the codes she used. However, she was sometimes able to produce creatively complex tasks. This might have been due to the misconceptions she developed as a result of her lower motivation during the laboratory sessions where programming tasks were completed. These findings are partly in alignment with the studies which found that the time pre-service teachers spent on tasks and their attitudes interplay with the development of their programming skills (i.e. Jin & Cutumisu, 2023; Cutumisu et al., 2021).

Findings of the current study indicate that the more challenged novice coders become through structured tasks that involve problem solving, debugging, and remixing, the more productive they become during programming. Rather than only reading a ready-made code, debugging, testing a program, and remixing engage novice learners of programming. As Brennan and Resnick (2012) argue, such productive tasks that ask novice coders to use computational practices enhance their learning process. Also, such structured and meaningful tasks within a context help novice programmers produce more codes that work properly (Kafai & Burke, 2013).

In the current study, pre-service teachers, as novice coders, sometimes struggle cognitively and creatively during programming. As in the case of Alice, misconceptions or a surface understanding of CT concepts during programming instructions may have resulted in cognitive challenges as they are not experienced in programming (Mouza et al., 2017). This finding aligns with previous research (i.e., Ozden, 2021; Yadav et al., 2016). The findings also demonstrate that despite being challenged, pre-service teachers who put more effort to overcome struggles during programming produce higher-quality codes. It can be argued that as long as they allocate sufficient effort to fixing their programs, and as long as it yields to resolving of the errors, even novice coders can successfully learn programming (Bers, 2019; Lee et al., 2011).

In addition, findings of the present study suggest that pre-service would benefit from an incremental design of programming tasks. In other words, task interdependence might be a remedy for the creative and cognitive challenges experienced by novice coders (Umutlu, 2022). Task interdependence can be defined as when sub-tasks are designed in a way to facilitate achievement in each sub-task as dependent on one another (Miyake & Kirschner, 2014). In this way, novice coders can transition from lower-level tasks to productive tasks more easily.

Pre-service teachers resort to instructor's support, external resources, remixing, and trial-and-error to handle programming challenges they experience. Being inexperienced in coding, if they cannot solve an issue in their programs, they either seek help from the instructor or external resources or get involved in trial-and-error and fix the code issue. While the former way can be explained through the need for a more knowledgeable other (Vygotsky, 1978) in programming and direct instruction (Kapur, 2015; Klahr & Nigam, 2004), the latter strategy (trial-and-error) can be classified as tinkering during programming (Dong et al., 2019). Taking the fact that pre-service teachers in the current study had no knowledge of coding into consideration, it can be argued trial-and-error and tinkering can be used by coders regardless of their prior knowledge in programming (Resnick & Rosenbaum, 2013). As Brennan and Resnick (2012) stated, remixing is another effective strategy especially for novice learners of programming when they get stuck during coding.

To sum up, the study findings highlight important phases of pre-service teachers' learning process of programming through CT-oriented tasks. It can be concluded that there are several elements that interplay with how pre-service teachers, as novice coders, engage in CT and how they handle the challenges encountered during programming. Within the scope of this study, task requirements, effort put to solve code issues, and support types to overcome challenges can be counted as a few of these factors.

Conclusions

This case study explored pre-service teachers' learning process of programming by completing CT-oriented tasks. It put forward how they perceived their programming sessions, what kind of challenges they went through during their programming tasks, and how they dealt with these challenges.

Understanding the learning processes the novice coders go through in detail is an important step to develop more effective learning environments for programming. Participants of this study were pre-service teachers, who could use their own programming knowledge in their own future teaching for lesson material design or as part of their course subject implicitly or explicitly. Therefore, on another note, the study has implications for pre-service teacher education, and indirectly for introduction of programming by subject area teachers in K-12 education (Yadav, et al., 2016). Examining the learning progress of pre-service teachers is eventually a step to empower them with the tools, such as coding, which would produce more effective results in their teaching.

This study aimed at bringing an understanding into pre-service teachers' development of programming and CT skills through obtaining data from observations of critical instances in a 14-week educational technology course, programming tasks, and interviews using a single case study design. This study may have limitations. First, the data was collected through interviews with participants, observations and via the end product of participants. However, no data collection was carried out regarding the participants' actual coding process, which would require screen recording. Another drawback of the study was participants were allowed to finish their Create 2 tasks outside of the class time, which limited the controllability and thus may have led to fluctuations in their coding process that could not be captured. Although it is a single case study, the number of participants who were considered as key informants was low. To mitigate this limitation, we aimed to enlarge the dataset by using multiple data sources. It should also be noted that participants of this study worked only on a block-based programming platform. Therefore, experiences of novice coders in text-based programming platforms might be different. Further research studies with microgenetic data collection (i.e. screen recording, think-aloud protocols, written reflections) and analysis methods (i.e. log analysis, process analysis, network analysis) within different contexts (i.e. text-based programming, hands-on programming with robotics) can be conducted to deeply explore how pre-service teachers experience learning programming and CT.

Ethical Committee Permission Information

Name of the board that carries out ethical assessment: Boğaziçi University Social and Humanities Scientific Research and Publication Ethics Board

The date and number of the ethical assessment decision: 24.01.2023-109746

Author Contribution Statement

Burcu ŞENER: *Conceptualization, methodology, data analysis, and writing.*

Duygu UMUTLU: *Conceptualization, literature review, methodology, data analysis.*

References

Albayrak, E., & Ozden, Ş. Y. (2021). Improvement of pre-service teachers' computational thinking skills through an educational technology course. *Journal of Individual Differences in Education*, *3*(2), 97-112.

Bal, I. A., Alvarado–Albertorio, F., Marcelle, P., & Oaks–Garcia, C. T. (2022). Pre–service teachers computational thinking (CT) and pedagogical growth in a micro–credential: A mixed methods study. *TechTrends*, *66*(3), 468-482.

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

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* (Vol. 1, p. 25).

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of Educational Research, 87*(4), 834-860.

Butler, D., & Leahy, M. (2021). Developing preservice teachers' understanding of computational thinking: A constructionist approach. *British Journal of Educational Technology*, *52*(3), 1060-1077.

Cooper, J.E., Brandon, P.R., & Lindberg, M.A. (1998). Evaluators' use of peer debriefing: Three impressionist tales. *Qualitative Inquiry*, *4*(2), 265-279.

Creative Computing Lab (2021). *Getting unstuck*. Harvard Graduate School of Education. Retrieved from https://gettingunstuck.gse.harvard.edu/

Creswell, J. W. (2013). *Qualitative inquiry and research design: Choosing among five approaches* (3rd ed.). SAGE Publications.

Cutumisu, M., Adams, C., Glanfield, F., Yuen, C., & Lu, C. (2021). Using Structural Equation Modeling to Examine the Relationship Between Preservice Teachers' Computational Thinking Attitudes and Skills. *IEEE Transactions on Education*, *65*(2), 177-183.

Çiftçi, A., & Topçu, M. S. (2022). Improving early childhood pre-service teachers' computational thinking teaching self-efficacy beliefs in a STEM course. *Research in Science & Technological Education*, *41*(4), 1215-1241.

Çoklar, A., N. & Akçay, A. (2018). Evaluating programming self-efficacy in the context of inquiry skills and problem-solving skills: A perspective from teacher education. *World Journal on Educational Technology: Current Issues. 10*(3), 153-164.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39.

Dong, Y., Marwan, S., Catete, V., Price, T., & Barnes, T. (2019, February). Defining tinkering behavior in open-ended block-based programming assignments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 1204-1210).

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with Scratch in primary schools: A systematic review. *Computer Applications in Engineering Education*, *29*(1), 12-28.

Flyvbjerg, B. (2011). Case study. In N. K. Denzin & Y. S. Lincoln (Eds.), *The Sage handbook of qualitative research* (1st ed., p. 301-316). Sage Publications.

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

Hanbay-Tiryaki, S., & Balaman, F. (2021). Açık kaynak kodlu yazılımlardan scratch, arduino ve python kullanımı hakkında öğrenci görüşleri. *Journal of Computer and Education Research*, *9*(18), 831-852. https://doi.org/10.18009/jcer.938706

Ilic, U. (2021). The impact of scratch-assisted instruction on computational thinking (ct) skills of pre-service teachers. *International Journal of Research in Education and Science, 7*(2), 426-444.

Instefjord, E. J., & Munthe, E. (2017). Educating digitally competent teachers: A study of integration of professional digital competence in teacher education. *Teaching and Teacher Education*, *67*, 37-45.

ISTE (2016). ISTE standards for students. Retrieved from https://www.iste.org/standards/standards/for-students-2016

Jin, H. Y., & Cutumisu, M. (2023). Predicting pre-service teachers' computational thinking skills using machine learning classifiers. *Education and Information Technologies*, *28*, 11447–11467.

Kafai, Y., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan*, *95*(1), 61–65. https://doi.org/10.1177/003172171309500111

Kapur, M. (2015). Learning from productive failure. *Learning: Research and Practice*, *1*(1), 51–65.

Kaya, E., Yesilyurt, E., Newley, A., & Deniz, H. (2019). Examining the impact of a computational thinking intervention on pre-service elementary science teachers' computational thinking teaching efficacy beliefs, interest and confidence. *Journal of Computers in Mathematics and Science Teaching*, *38*(4), 385-392.

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science*, *9*, 522-531.

Klahr, D., & Nigam, M. (2004). The equivalence of learning paths in early science instruction: Effects of direct instruction and discovery learning. *Psychological Science, 15*(10), 661–667.

Lee, D. M. C., Rodrigo, M. M. T., Baker, R. S. D., Sugay, J. O., & Coronel, A. (2011). Exploring the relationship between novice programmer confusion and achievement. In *Affective Computing and Intelligent Interaction: 4th International Conference, ACII 2011, Memphis, TN, USA, October 9–12, Proceedings, Part I 4* (pp. 175-184). Springer Berlin Heidelberg.

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry*. Sage Publications.

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

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004). Scratch: A sneak preview [Conference proceeding]. In *Proceedings of Second International Conference on Creating, Connecting and Collaborating through Computing*. (pp. 104-109). IEEE.

Merriam, S. B. (2009). *Qualitative research: A guide to design and implementation*. Jossey-Bass.

Miyake, N., & Kirschner, P. A. (2014). The social and interactive dimensions of collaborative learning. In K. Sawyer (Ed.), *The Cambridge handbook of the learning sciences* (pp. 418-438). Cambridge.

Mouza, C., Yang, H., Pan, Y. C., Ozden, S. Y., & Pollock, L. (2017). Resetting educational technology coursework for pre-service teachers: A computational thinking approach to the development of technological pedagogical content knowledge (TPACK). *Australasian Journal of Educational Technology*, *33*(3), 61-76.

Mugayitoglu, B. (2016). Attitudes of pre-service teachers toward computational thinking in education. [Doctoral Dissertation, Duquesne University].

OpenAI. (2022). *Whisper* [Automatic speech recognition system]. https://openai.com/research/whisper

Piedade, J., Dorotea, N., Pedro, A., & Matos, J. F. (2020). On teaching programming fundamentals and computational thinking with educational robotics: A didactic experience with pre-service teachers. *Education Sciences*, *10*(9), 214.

Qualls, J. A., & Sherrell, L. B. (2010). Why computational thinking should be integrated into the curriculum. *Journal of Computing Sciences in Colleges*, *25*(5), 66-71.

Resnick, M., & Rosenbaum, E. (2013). Designing for tinkerability. In M. Honey (Ed.), *Design, make, play: Growing the next generation of STEM innovators* (pp. 163-181). Routledge.

Saldana, J. (2013). *The coding manual for qualitative researchers* (2nd ed.). Sage.

Umutlu, D. (2022). An exploratory study of pre-service teachers' computational thinking and programming skills. *Journal of Research on Technology in Education*, *54*(5), 754-768.

VERBI Software. (2022). *MAXQDA* [Computer software]. https://www.maxqda.com/

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Harvard University Press.

Wakil, K., Khdir, S., Sabir, L., & Nawzad, L. (2019). Student ability for learning computer programming languages in primary schools. *International e-Journal of Educational Studies*, *3*(6), 109-115.

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

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

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)*, *14*(1), 1-16.

Yin, R. K. (2009). *Case study research: Design and methods* (5th ed.). SAGE Publications.

Appendix A – Interview questions

Can you state your name, age, and your department?
Did you have any coding experience before taking this course?
Let's say I'm a friend who will take will course next year. How would you describe it to me?
- What are you learning in this course?
- Can you give me some examples of the concepts you are learning?
- Can you explain some of them to me?
- How much of your time and energy do you give to this course in a week?
- What do you think the purpose of this course is?
- Do you use what you have learned in this class in your daily life?

Describe to me a typical lab session of this course.
- What do you do at the beginning/middle/end?
- Can you tell me your feelings and thoughts during the lab tasks? You may focus on one of the previous weeks.
- What do you do during tasks?
- What kind of phases do you go through?
- What kind of strategies do you use when finishing the tasks? For instance, what do you do when you're stuck?
- How do you feel during tasks?
- Are there any parts you feel challenged about? If yes, can you explain through examples?
- Do you think this course has improved your Computational Thinking abilities?
- As a pre-service teacher, what would you change about these lab tasks?

As a student who is taking such a course, how do you imagine yourself as a teacher in the future?
- What kind of contributions do you think this course can make to your future teaching?
- How would you use what you have learned in the labs when you become a teacher? Can you give examples?

Is there anything you would like to add?