An Investigation of In-service Teachers' Perceptions and Development of Computational Thinking Skills in a Graduate Emerging Technologies Course

Yi JIN[1]

Jason R. HARRON[1]

[1]Kennesaw State University, United States of America

DOI: https://doi.org/10.21585/ijcses.v6i2.165

Abstract

Computer science (CS) has become a critical part of K–12 education worldwide. Computational thinking (CT) skills are a key set of competencies in CS education that can solve problems and use computational design to create useful solutions. However, preservice and in-service teachers are not fully prepared to integrate CS and CT into their curricula. Furthermore, there are limited special topic courses and educational research on how to facilitate in-service teachers' professional learning of CS and CT, as well as their content-specific integration. Therefore, this study investigated in-service teachers' perceptions and development of CT skills in an online graduate emerging technologies course. Theoretically framed by the four cornerstones of CT (i.e., abstraction, algorithms, decomposition, and pattern recognition), participants perceived that they increased their CT problem-solving and creativity skills but decreased their collaborative learning and critical thinking skills. Additionally, teachers increased their CT test scores after taking the course. Most teachers used CT terminology correctly (i.e., algorithms and decomposition). However, only 59% correctly described abstraction and pattern recognition, while most teachers did not mention debugging. The authors call on teacher educators to address in-service teachers' CS knowledge gaps, increase their CT skills, and select appropriate strategies for CT professional learning.

Keywords: computational thinking, creative computing, online learning, perceptions, teacher education

1. Introduction

Computational Thinking (CT) skills are a key set of competencies that combine problem-solving and computational design to create useful solutions (Grover & Pea, 2018). Students and teachers with CT skills can collect and analyze data, decompose problems, recognize patterns, and filter out variables to find novel and elegant solutions. CT helps people to think like computer scientists and transform complex problems into ones that can be easily understood across a wide range of subjects. In combination, CT and coding have immense potential to transform K–12 education by integrating core computational concepts and principles across the curriculum.

In recent years, movements at the national and state levels in the U.S. have aimed to introduce students to computer science (CS) education by establishing frameworks, standards, and curricula with the goal of expanding CS opportunities to all. Nationally, this push includes the development of the *K–12 Computer Science Framework* (2016), which highlights CT as one of four significant themes that are interwoven throughout. This framework aligns with the *International Society for Technology in Education (ISTE) Standards for Educators* and *Students* by sharing the vision that CT is important for all teachers and students (ISTE, 2016a, 2016b). Based on these efforts, the Computer Science Teachers Association (CSTA) has proposed a comprehensive set of K–12 standards in collaboration with multiple national and international associations to guide how CS education is implemented in practice (CSTA, 2017). Similarly, many countries have incorporated CS education into their curriculum (Dufva & Dufva, 2016).

Due to these collective endeavors, CSforALL movements have been fruitful in the U.S. According to the *2022 State of Computer Science Education* report, 37 states have adopted at least five of nine recommended policies to make CS part of the education system while 27 states require all high schools to offer at least one CS course (Code.org, CSTA, & ECEP Alliance, 2022). Across the U.S., 53% of public high schools (13,865) offer fundamental CS, up from 35% in 2018. Moreover, 76% of students attend a high school that offers a foundational CS course. All 50 states and Washington D.C. allow CS courses to be counted toward the graduation requirement. Furthermore, Arkansas, Nebraska, Nevada, South Carolina, and Tennessee require high school students to take CS courses for graduation. Although there are great advances in offering CS courses at the high school level, only 3.9% of middle school and 7.3% of elementary school students from the 19 states who reported middle and elementary school data offered foundational CS in grades K-8, highlighting the need to integrate CS into all content areas at the K-8 level to broaden participation (Code.org, CSTA, & ECEP Alliance, 2022; Kennedy et al., 2021).

Despite the growth in CS offerings, there continue to be access issues in K–12. First, access disparities persist in rural schools, urban schools, and schools with high percentages of economically disadvantaged students. These disparities also exist across gender boundaries, with fewer female students enrolled in CS courses across the elementary (49%), middle (44%), and high school (32%) grade bands (Code.org, CSTA, & ECEP Alliance, 2022). Furthermore, students from underrepresented populations, such as African American, Hispanic/Latino/Latina/Latinx, and Native American/Alaskan, are less likely to have CS courses offered at their schools. Compared to their white and Asian peers, Hispanic/Latino/Latina/Latinx high school students are 1.4 times less likely to take a CS course. Similarly, English language learners, students with disabilities, and economically disadvantaged students are underrepresented in CS courses. These data emphasize that besides learning about CS and CT, preservice and in-service teachers also need to proactively seek strategies to teach these underrepresented students.

Although there are strong pleas to integrate CS and CT into all K–12 content areas (Grover & Pea, 2018; Kennedy et al., 2021), most teachers have not been able to achieve this goal in practice. One significant barrier causing the stagnant CT implementation includes a lack of preparation from teacher education programs and minimum professional development from schools and districts. For example, research shows that few teacher education programs provide CT training to preservice teachers (Yadav et al., 2017a). In addition, many K–12 in-service teachers had little knowledge about CT and did not know how to implement CT in their classrooms (Sands et al., 2018). In-service teachers also lack strategies for teaching CS and CT to underrepresented students (Gretter et al., 2019). Teachers even expressed that they were anxious about developing new learning resources and using novel technologies (Meerbaum-Salant et al., 2013), especially when teaching CT concepts and computing-related subjects (Grover & Pea, 2013). All these shortcomings underline the need for teacher educators to provide support and professional learning to both preservice and in-service teachers in integrating CS and CT into their subject areas and curricula (Voogt et al., 2015; Yadav et al., 2017b).

For in-service teachers, research has shown that targeted professional learning helps teachers improve their CT understanding and skills (Bower et al., 2017; Jaipal-Jamani & Angeli, 2017; Ketelhut et al., 2020). However, professional learning in literature occurred mostly in professional development programs, not courses in teacher education. Therefore, educational researchers need to design specific courses that facilitate teachers' professional learning in CS and CT, especially for elementary and middle school in-service teachers to design content-specific integration (Kennedy et al., 2021). In turn, this need warrants more studies examining the effectiveness of such courses. There is a limited number of this type of research in literature, especially those focusing on using the creative coding concept (Brennan, 2015; Yurkofsky et al., 2019). Thus, this study aims to investigate in-service teachers' perceptions and development of CT skills in a required emerging technologies course as part of an online instructional technology graduate program. The details of the design of this professional learning course and its effectiveness shed light on how to prepare in-service teachers to integrate CS and CT into their content areas. Moreover, the findings add to the literature on CT integration using the creative coding concept. Therefore, the current research intends to answer the following research questions:

> (1) What are in-service teachers' perceptions about their CT skills before and after taking the graduate emerging technologies course?

(2) Is there a difference in in-service teachers' CT test scores after taking the course?
(3) How frequently and accurately do in-service teachers apply CT terminology in their final reports?

2. Literature Review

To better understand what researchers currently know about how teachers develop their CT skills, a review of the literature is provided below. This review includes a brief overview of the skills, practices, and pedagogy associated with CT, and summarizes how CT has been studied in K–12 and teacher education programs.

2.1 Computational Thinking Skills, Practices, and Pedagogy

Computational thinking (CT) has its origins in the 1980s, stemming from research about using personal computers and computing environments to support the social processes of learning while aiding in the development of higher-order cognitive skills (Papert, 1980; Pea & Kurland, 1984; Solomon, 1988). Wing (2006) brought CT to the mainstream discussion with her seminal and influential *Communications of the ACM* article, where she argues that CT is not only for computer scientists but serves as a set of attitudes and skills that are universally applicable to everyone. In particular, CT provides its users with various mental tools to solve problems, design systems, and understand human behaviors using a broad range of CS concepts.

Since the publication of Wing's article over 15 years ago, there have been more than 31,000 publications about CT indexed by Google Scholar. Expanding upon Wing's foundational definition, Barr and Stephenson (2011) provided educators with an operational definition, which defined CT as a problem-solving process involving the following steps: (a) formulating a problem in such a way that the use of computer technology can help us solve it; (b) analyzing data and representing that data through models or simulations; (c) identifying possible solutions to the problem posed; (d) generalizing this process to a wide variety of situations and issues.

However, despite the popularity of CT within the educational research community, there is still no consensus about how CT should be universally defined (Cansu & Cansu, 2019; Grover & Pea, 2018). The early definitions, which centered around the four cornerstones of abstraction, algorithms, decomposition, and pattern recognition, have been expanded upon to include a wide variety of CT skills/concepts and practices. For example, Mills et al. (2021) recently published a report that places CT at the intersection of computing, computer science, and programming. Their report proposes that CT consists of a set of skills and practices that can be applied to solve problems. CT skills include abstraction, algorithmic thinking, debugging, decomposition, pattern recognition, and selecting tools. CT practices combine these skills to solve problems through the creation of computer programs (i.e., automation), data visualizations, or computational models. Lastly, these CT skills and practices are centered around the use of inclusive pedagogies which includes strategies "for engaging all learners in computing, connecting applications to students' interests and experiences, and providing opportunities to acknowledge and combat biases and stereotypes within the computing field" (Mills et al., 2021, p. 10).

Similarly, Yaşar et al. (2015) considered computational pedagogy an inherent outcome of computing, math, science, and technology integration. They firmly believe that computational modeling and simulation technology (CMST) can be used to improve teachers' technological pedagogical content knowledge (TPACK) (Mishra & Koehler, 2006; Yaşar et al., 2015). Thus, Yaşar et al. (2015) extended TPACK into Computational Pedagogical Content Knowledge to highlight computational pedagogy.

For this particular study, the researchers decided to use the operational definitions from the BBC Bitesize courses, which were also used as instructional materials in the course. The website defines that "computational thinking allows us to take a complex problem, understand what the problem is and develop possible solutions. We can then present these solutions in a way that a computer, a human, or both, can understand" (BBC Bitesize, n.d., What is computational thinking section, para. 2). Furthermore, they define the four cornerstones of CT as

- Decomposition - Breaking down a complex problem or system into smaller, more manageable parts.

- Pattern recognition - Looking for similarities among and within problems.
- Abstraction - Focusing on the important information only, ignoring irrelevant detail.
- Algorithms - Developing a step-by-step solution to the problem, or the rules to follow to solve the problem (BBC Bitesize, n.d., What is computational thinking section, para. 3).

2.2 Computational Thinking in K–12 Education

Traditionally, CS has been introduced at the high school level, focusing on teaching the computer programming skills needed to pass the AP CS exam (Goode, 2008). CT breaks this mold by acknowledging that students in younger grades (K–3) have the cognitive capabilities to apply computational skills to relevant problems (Papdakis, 2021; 2022). These skills can be introduced through "unplugged" activities that do not require digital devices (Mills et al., 2021), such as having students give each other step-by-step instructions on how to brush their teeth (Hello Ruby, 2019). Other developmentally appropriate devices, such as Beebots or Codeapillar, allow students to manually program algorithms by giving step-by-step instructions at the push of a button (Papadakis et al., 2021). Besides these physical computing tools and activities, coding apps are used widely by younger learners, such as ScratchJr, Lightbot, Kodable, and Daisy the Dinosaur (Papdakis, 2021). In particular, Papadakis (2022) conducted a literature review on ScratchJr and found that it helped young learners understand CT concepts, practice coding skills, develop social-emotional skills, introduce students to STEM learning, especially numeracy concepts, and help them develop problem-solving strategies, planning methods, and thinking skills. Therefore, CT can be taught to young students and should be taught as early as possible (Kotsopoulos et al., 2017; Papadakis, 2021; 2022; Yadav et al., 2011).

In upper-grade levels (4–12), students can continue to develop their CT skills through the use of block-based programming languages, such as Scratch, or through the exploration of devices that utilize the Blockly programming library (Weintrop, 2021). Some of these devices include BBC micro:bit, Circuit Playground Express, Lego Mindstorms, Ozobots, Raspberry Pi, and Sphero. The user-friendly nature of these block-based programming languages allows for an entry point to computer science not only for students but also for teachers who are learning to code for the first time. Kalogiannakis et al. (2021) conducted a systematic review of the use of BBC micro:bit in elementary schools. They found that students and teachers show a positive attitude towards the tool. Moreover, students believe that micro:bit encourages creativity and facilitates their learning of the conceptual and procedural knowledge of CT and problem-solving. However, the findings also indicate teachers' lack of confidence in designing their own activities and instructions.

There is a trend to integrate CT into K–12 content areas. For example, CT has become a core scientific practice in STEM (NGSS, 2013; Weintrop et al., 2016). To facilitate empirical research, Weintrop et al. (2016) proposed a Computational Thinking in Mathematics and Science Taxonomy with four categories to ground CT in STEM. These categories include (a) data practices, (b) modeling and simulation practices, (c) computational problem-solving practices, and (d) systems thinking practices. Furthermore, CT integration into the science classrooms is well-researched on topics such as adding coding activities with little support for science learning (Grover et al., 2015), integrating CT into the science content knowledge of science textbooks (Wilkerson & Fenwick, 2017), and integrating computation as used by STEM professionals (Orton et al., 2016).

Empirical research about CT integration in math is expanding as well. In a scoping review, Hicknott et al. (2017) found that most CT integration in K–12 mathematics classrooms mainly concentrated on teaching programming skills and rarely focused on mathematical concepts in probability, statistics, and measurement of functions. Likewise, Barcelos et al. (2018) conducted a systematic review and found 42 studies. Fourteen programming languages were used in 22 studies, with Scratch being the most popular one. These studies also covered a wide range of math skills and contents, which were developed in conjunction with CT. The researchers suggested that interest in investigating the relationship between CT and math was growing.

Concerning CT instructions in K–12 settings, two main approaches are used, unplugged and programming activities. Huang and Looi (2021) conducted a critical review of the unplugged pedagogies used in K–12. They found that most unplugged activities were designed for younger students and non-specialist teachers and they were popular across age groups and learner characteristics. They summarized that unplugged pedagogy supports CT development,

complements programming to develop CT, integrates with other subjects to develop CT, and facilitates teacher learning about CT and CS.

For teaching coding in K–12, Hsu et al. (2018) found that teachers mostly used visual programming languages in their CT instruction. Teachers' top strategies for CT instruction are project-based learning, problem-based learning, cooperative learning, and game-based learning. In contrast, other activities involving aesthetic experience, design-based learning, and storytelling are rarely adopted. To determine the general effectiveness of using programming for developing K–12 students' CT skills, Sun et al. (2021) conducted a meta-analysis. They found 86 empirical studies with 114 effect sizes. According to their results, programming activities could improve K–12 students' CT skills. They also found some instructional design factors that were more conducive to the goal, which were interdisciplinary integration of programming, setting the duration to be within one week to one month, having a class size of fewer than 50 students, and a practical selection of programming instrument and CT assessment types. Because of the popularity of Scratch as a programming language in K–12 CT instruction, numerous scholars have conducted research to analyze the impact of Scratch on fostering CT. Montiel and Gomez-Zermeño (2021) conducted a systematic review and found 30 articles. They suggested that Scratch is suitable for teaching CT in K–12 education. Although research investigating CT skills in K–12 is prolific, studies investigating how preservice and in-service teachers are prepared for learning and teaching CT skills are relatively scarce, underscoring a need to conduct more empirical research on the teacher population.

2.3 Coding and Computational Thinking in Teacher Education

While the topic of teaching CS in K–12 schools has recently received widespread interest, issues related to teaching coding and CT as part of teacher education have existed for over 40 years (Bull et al., 2020; Schmidt-Crawford et al., 2019). Most notably, the debate in favor of introducing programming to children in K–12 environments stems from Seymour Papert and the publication of *Mindstorms* (Papert, 1980). In his book, Papert argues that by learning computer programming children teach the computer how to think, which can serve as a catalyst for children to embark on the epistemological journey of thinking about their own thinking. Designed as a tool for learning, Papert and a team of researchers at MIT developed the Logo Programming Language (Logo Foundation, 2014). Early versions of the Logo allowed people to control a robotic turtle, which Papert (1980) described as a "computational object-to-think-with" (p. 11). The turtle eventually migrated to the computer screen as a controllable graphic called a "sprite," which could be used to draw shapes, graphics, and patterns.

In the early 1980s, Logo and other programming languages (e.g., BASIC and Pascal) were starting to find their way into the K–12 classrooms. For example, by January 1983, the state of California had established 15 Teacher Education and Computing Centers with the goal of providing training to teachers in mathematics and CS (Gray, 1983). A few months later, Apple announced their Kids Can't Wait program, which aimed to place 9,250 Apple IIe computers in California elementary and secondary schools (Uston, 1983). Each computer included a copy of the Apple Logo, and representatives from Apple dealers were trained to assist teachers in how to use the programming language.

While Logo had an initial uptake by enthusiastic progressive educators in the US and UK, by the mid-to-late 1980s the majority of teachers dreaded the Logo training sessions out of a fear of being embarrassed in front of their colleagues, or by being "shown up" by students in the classroom who had more expertise at debugging code (Agalianos et al., 2001). Although Logo was initially seen as a promising way to transform curriculum, cognitive and metacognitive studies from the mid-1980s found little to no difference between Logo and non-Logo users (Ames, 2018). Despite these failures in the K–12 setting, researchers at MIT continue to develop new platforms, such as LEGO/logo, which allowed people to build programmable machines with LEGO bricks (Resnick & Ocko, 1990). As part of the LEGO/logo project, a new version of the Logo was created called Logo Blocks (Logo Foundation, 2014). This innovation allowed users to create programs by snapping together jigsaw-like puzzle pieces instead of writing text-based lines of code. This block-based coding innovation was incorporated into a new Logo programming environment called Scratch, which was officially launched to the public in 2007 (Resnick et al., 2009).

While the timing of Wing's 2006 article on CT and the 2007 release of Scratch are not directly correlated, they both serve as a catalyst for the reintroduction of CS into teacher education programs. One of the challenges with introducing

these concepts into teacher education is addressing misconceptions about what delineates CS, CT, and coding. As Yadav et al. (2017a) point out, while CS unplugged activities and block-based programming languages like Scratch are an approachable way to introduce preservice and in-service teachers to CT, care must be taken in teacher education programs to ensure that CT is not mistakenly equated with programming or instructional technology. Their survey study, which examined 134 preservice teachers' conceptions of CT and classroom implementation, found that participants defined CT in terms of problem-solving and logical thinking, and often associated the concept with the use of a computer. They recommend that teacher educators should embed CT within educational technology and content-specific method courses. By doing so, preservice teachers will have more opportunities to think computationally and gain experience with CT as a generic set of skills that do not require a computer.

While CT does not require a computer, robotics and other physical computing tools have been used to introduce preservice and in-service teachers to CT. Jaipal-Jamani and Angeli (2017) studied how 21 preservice teachers learned about CT as part of an elementary science methods course. Their study found that throughout the semester-long course, preservice teachers' interest and self-efficacy toward robotics increased and that participants showed gains in CT skills such as learning how to write algorithms and debug programs. Additionally, Mason and Rich (2019) performed a literature review that synthesized 21 studies on elementary preservice and in-service teachers' attitudes, self-efficacy, or knowledge to teach computing, coding, or computational thinking. As part of their review, six of the studies focused on both CT and robotics. They found that although most interventions were relatively short in duration, training and professional development led to gains in preservice and in-service teachers' computing content knowledge and self-efficacy.

In addition, Bower et al. (2017) have also shown that in-service K–8 teachers can improve their CT pedagogical capabilities through a combination of "unplugged" and block-based coding activities. They conducted a series of CT workshops which found that teachers developed their CT understanding, pedagogical capacities, technological knowledge, and confidence through these targeted professional learning opportunities. While research has shown that teachers can be successful in learning how to code as part of their in-service training, these coding and CT skills do not automatically transfer to their teaching practices (Guven & Kozcu Cakir, 2020). Instead, teachers need to be introduced to CT within the context of the subject area in which they teach (Yadav et al., 2017c).

2.4 The Impact of the COVID-19 Pandemic on Teachers' Professional Learning of CS and CT

The COVID-19 pandemic has also been posing challenges in providing in-service teachers with needed professional learning opportunities on CS and CT. Virtual professional development (PD) programs have become a popular way to solve participation problems. For example, Jocius et al. (2021) transformed their summer PD workshops into a virtual conference format, including emerging technology tools, pre-PD training, synchronous and asynchronous sessions, Snap! Pair programming, live support, and live networking. They found that the digital tools, formats, and support for teacher engagement and collaboration were the most effective changes they made that increased participants' self-efficacy in teaching CT, supporting collaboration, enabling participants to design CT-infused content-area lessons, and learning about strategies for virtual, hybrid, and face-to-face classroom teaching. Based on the overall success, this group of researchers commented that they plan to continue to develop and use virtual PD.

Similarly, Mouza et al. (2022) decided to utilize a virtual PD institute for K–12 in-service teachers, which includes both synchronous and asynchronous sessions. Participants reported higher scores in knowledge and skills after the virtual PD program, as well as a higher level of confidence and preparation to teach CS in practice. Both Jocius et al. (2021) and Mouza et al. (2022) pointed out the importance of teachers' collaboration and sharing officially and unofficially during virtual PD programs. Jocius et al. (2021) cautioned the researchers to increase the number of facilitators, provide more extensive pre-workshop training, and carefully select virtual tools. Comparably, Mouza et al. (2022) especially recommend diversifying and broadening teacher participation, providing differentiated instruction, increasing hands-on activities, and prioritizing teachers' engagement.

To address the need for content-specific integration of CS and CT and broadened participation, the authors of this study introduced in-service teachers to CT and coding as part of a graduate-level online course. These teachers developed their own content-specific CT lessons and implemented those lessons in their K–12 classrooms,

makerspaces, or as part of after-school programs. In particular, this study aims to investigate in-service teachers' perceptions and development of CT skills in this required emerging technologies course as part of an online instructional technology graduate program.

3. Methods

In this section, the researchers describe the implementation of a case study methodology to study in-service teachers' perceptions and development of CT skills (Yin, 2017). Using a holistic single-case design, the unit of analysis is bounded to 29 participants who were enrolled in a graduate emerging technologies course during the Fall of 2021.

3.1 Research Context and Module Design

Creating with Emerging Technologies is an asynchronous online graduate-level course that is designed to introduce in-service teachers to trends and issues related to instructional technology and design. This course was launched in the Fall of 2021 with four class sections that averaged 20 students per section. The course consists of eight modules, including (1) Introduction to Constructionism, (2) Computational Thinking, (3) Algorithms in Education, (4) Machine Learning and Artificial Intelligence, (5) Learning Spaces (i.e., makerspaces, Fab Labs, and active learning spaces), (6) eXtended Reality (i.e., virtual, augmented, and mixed reality), (7) Open Educational Resources (OER), and (8) The Creative Classroom. As part of a 15-week course, the first seven modules are designed to take two weeks each, with the last module serving as a one-week final reflection. Each module consists of required reading, online videos, a written reflection, and either a coding, electronics, or 3D modeling project. During the first week of each module, students complete the readings, watch the videos, and post a 300-500 word summary as part of a Google Slide design journal. During the second week, students reply to at least two of their peers, and complete a weekly project (e.g., creating a digital story in Scratch). The required materials for the course include the *SparkFun Inventor's Kit for micro:bit*, which includes a micro:bit, breadboard, and various electrical components such as LEDs, resistors, wires, potentiometer, servomotor, and switches (see Figure 1). While the course is designed for the micro:bit V2 (which includes a built-in speaker, microphone, and capacitive touch), this research study used the micro:bit V1 due to supply chain shortages. Kits for the study were purchased with internal grant funds and two of the four class sections were picked via a random number generator to participate in the study.

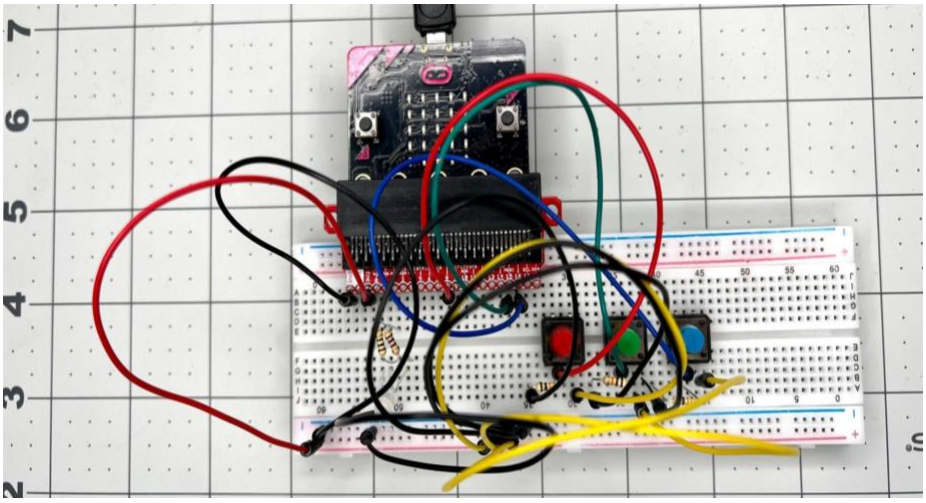

Figure 1. BBC micro:bit with a breadboard, wires, and electronic components.

As part of the course modules, participants are introduced to block-based coding using Scratch (Scratch, n.d.) and Microsoft Makecode for micro:bit (Microsoft Makecode, 2022). Activities with these platforms include creating a digital story in Scratch (Module 1), programming two inputs and outputs with the BBC micro:bit (Module 2), programming and wiring two inputs on outputs with the breadboard (Module 3), and creating an interactive robotic

pet (Module 4). These activities are part of the first four modules in the course and are supported by prerecorded video tutorials, plus two weekly synchronous "Hour of Code" sessions for live troubleshooting. Additionally, as part of the second module, students are introduced to CT through required readings (Grover & Pea, 2018; Wing, 2006) and complete an online quiz based on the BBC Bitesize CT learning modules (BBC Bitesize, n.d.). While CT is the focus of the second module, the concepts and terminology are reinforced throughout the entire course. As part of the fifth module, participants developed a lesson proposal for a Creative Computing Project, which involved teaching CT and a design process (e.g., creative play, design thinking, or engineering design process) in an alternative setting (e.g., a non-traditional classroom, makerspace, or after-school program.) Suggested Creative Computing Projects included hands-on CS Unplugged activities, digital storytelling in Scratch, or breadboarding with Makecode and the BBC micro:bit. After implementing their project, participants wrote a Creative Computing Project final report, which documented the design and implementation of their project and was due by the end of the seventh module. The final report includes a section on CT, where participants are encouraged to use CT terminology as part of their open-ended responses.

3.2 Participants

Overall, 29 in-service teachers voluntarily participated in this study. Among them, 24 teachers completed both the pre and post-surveys while one teacher only filled out the presurvey. Four teachers did not respond to the survey requests. Based on the 25 responses to the demographic questions, six teachers identified as men and 19 as women. Five participants were 23-26 years old, two were 27-32 years old, six were 32-40 years old, nine were 40-50 years old, and three were more than 50 years old. Fourteen teachers are white, seven are African Americans, three are Asians, and one is in the other category. Nine participants had Bachelor's degrees while 16 had Master's degrees. The years of teaching experience ranged from 2 to 28 years. These participants also taught in a variety of content areas and some of them taught in several categories: science (8), all subject areas (6), social studies (6), English Language Arts (4), math and science/STEM (3), health and physical education (2), food science and nutrition (1), video production (1), and one participant did not report their content area. Seven teachers worked in elementary schools, ten in middle schools, six in high schools, and two in the K–12 levels.

Twenty-four in-service teachers filled out the survey with questions about their competencies in programming languages. Three teachers said that they had some background in coding such as a Bachelor's degree in Computer Information Systems, coursework in computing languages, and teaching experiences with coding and robotics in their classrooms. However, 21 teachers reported that they did not have any coding background prior to the course. One teacher did not answer the questions. Teachers also reported their competencies with various coding languages (see Table 1). Overall, in-service teachers did not have extensive experience in programming languages. Furthermore, the majority of the teachers never programmed anything. Compared to other programming languages, teachers had relatively more experience in using educational coding languages, such as Scratch and OzoBlockly.

Table 1. In-Service teachers' self-reported competencies in programming languages (n = 24).

Programming languages	Never programmed in this language.	Minimal experience. Maybe compiled a test program.	Some experience. Wrote several small to medium-sized programs.	Substantial experience. Wrote several small to medium-sized programs.	Extensive experience. Wrote many programs.
C++	21	2	/	1	/
JAVA	18	4	2	/	/
Visual Basic	22	1	/	1	/
Python, Perl, or other scripting-	21	3	/	/	/

based languages					
JavaScript, HTML, ASP, or other web-based languages	17	6	1	/	/
Scratch, OzoBlockly, or another block-based coding	5	11	7	/	1

3.3 Data Collection and Analysis

The researchers used a validated survey instrument called the CTS scale to collect data on in-service teachers' perceptions of CT skills. The researcher who designed the survey instrument computed Cronbach's Alpha of the overall scale and reported an internal consistency coefficient of .969 (Yağci, 2019). The survey used in the current study has ten demographic questions and 42 Likert-scale questions on four variables: (a) problem solving (20 questions), (b) collaborative learning & critical thinking (8 questions), (c) creativity (9 questions), and (d) algorithmic thinking (5 questions). A pre and post-survey design was used. An informed consent form was sent to students in the course. Once the participants signed the consent form, a link to the presurvey was sent to them. It took students around 15 minutes to complete the survey. At the end of the coding instructions, a link to the post-survey was sent to the participants and it took them around 15 minutes to finish the post-survey. Cronbach's Alpha ranges from .45 to .89 (presurvey: .81, .74, .80, .53; post-survey: .89, .62, .79, .45). Cronbach's Alphas of the first three variables indicate they are very reliable, which demonstrates a high level of internal consistency for the scales with this specific sample. Cronbach's Alphas of the last scale, algorithm thinking, show it is a moderately reliable scale with the current sample (Hinton et al., 2004). Pair-sample *t*-tests were used to examine whether there were statistically significant differences in teachers' perceptions of CT.

A test of CT skills was also used in this study. This test has 12 multiple-choice questions and four open-ended questions. Participants took a pretest before learning the modules and afterward, they took the post-test. Paired-sample *t*-tests were conducted to investigate whether there were statistically significant differences in teachers' pre and post-test scores. These test scores are a way of measuring teachers' CT skills, which provides triangulation to the self-reported data on teachers' CT perceptions.

Qualitative data consisted of the participants' Creative Computing Project final report. This report included eight open-ended sections, one of which was devoted to CT. The prompt for the CT section stated, "Using language such as abstraction, decomposition, pattern recognition, and algorithms, describe the computational thinking that you observed as part of your Creative Computing Project. If you could redesign your lesson, what would you do to encourage more computational thinking?" Based on the themes of abstraction, decomposition, pattern recognition, algorithms, and debugging the researchers used deductive coding (Miles et al., 2019) to identify whether the CT terminology was used correctly, incorrectly, or was absent based on the definitions of the BBC Bitesize CT learning modules (BBC Bitesize, n.d.). The researchers calibrated their coding criteria by analyzing two of the participants' CT sections together and then coded the other 27 participants separately. Once coding was complete, the researchers initially agreed on the use of 93% of participants' use of terminology. Based on a Cohen Kappa, interrater reliability (IRR) was found to be 0.86, or a "near-perfect agreement" (Cohen, 1960; Ranganathan et al., 2017). The data was then reanalyzed to resolve any disagreements until 100% IRR was achieved.

4. Results

The researchers analyzed both quantitative and qualitative data to answer the three research questions, focusing on in-service teachers' perceptions and development of CT skills. Findings were triangulated using three types of data from the self-reported survey, CT pre and post-test, and the CT section of participants' final written report on their CT implementation. In the following section, results are written to answer each research question.

4.1. RQ 1: What were in-service teachers' perceptions about their CT skills before and after taking the graduate emerging technologies course?

In-service teachers' CT perceptions changed after taking the modules on coding and creative computing (see Table 2). There was a statistically significant improvement in their perceptions of problem-solving, $t(24) = -3.99$, $p < .001$, from 80.16 ± 6.81 to 86.44 ± 7.43, an improvement of 6.28 ± 7.88. A statistically significant decrease was found in teachers' perceptions of collaborative learning and critical thinking, $t(24) = 1.99$, $p = .03$, from 19.16 ± 5.23 to 17.36 ± 4.12, a decrease of 1.80 ± 4.52. Last, the researchers discovered a statistically significant increase in teachers' perceptions of creativity, $t(24) = -2.21$, $p = .02$, from 35.28 ± 4.69 to 36.92 ± 3.82, an increase of 1.64 ± 3.71. Changes in problem-solving had a large effect size of .88, while differences in collaborative learning & critical thinking and creativity had small effect sizes of .38. Algorithmic thinking had no statistically significant change.

Table 2. Results from the paired sample *t*-tests on in-service teachers' CT perceptions (*n*=25).

CT perceptions	Pre		Post		Paired sample *t*-tests		
	M	*SD*	*M*	*SD*	*t*	*p*	Cohen's d
Problem solving	80.16	6.81	86.44	7.43	-3.99	<.001***	.88
Collaborative learning & critical thinking	19.16	5.23	17.36	4.12	1.99	.03*	.38
Creativity	35.28	4.69	36.92	3.82	-2.21	.02*	.38
Algorithmic thinking	19.28	2.19	18.72	2.11	1.22	.12	.26

Note. * $p < .05$; ** $p < .01$; *** $p < .001$.

4.2 RQ 2: Was there a difference in in-service teachers' CT test scores after taking the course?

In-service teachers took the same test focusing on CT skills before and after the coding and creative computing modules. The test has a total of 100 points. Their pre and post-test scores of CT skills had a wide range, with pre-scores ranging from 28 to 100 and post-scores ranging from 25 to 100. Their pre and post-test scores changed after taking the coding and creative computing modules. There was a statistically significant improvement in their CT scores, $t(23) = -1.74$, $p < .05$, from 65.17 ± 19.04 to 73.04 ± 18.52, an improvement of 7.88 ± 22.18. The effect size is .42, a medium effect size.

The researchers conducted another paired sample *t*-test to further examine the difference in the test scores of the 12 multiple-choice questions. There was a statistically significant improvement in their scores on the multiple-choice questions, $t(23) = -3.57$, $p < .001$, from 36.88 ± 11.96 to 45.63 ± 10.35, an improvement of 8.75 ± 12.00. The effect size is .78, a large effect size. Overall, according to the CT test scores, in-service teachers developed their CT skills after studying the modules.

4.3 RQ 3: How frequently and accurately did in-service teachers apply CT terminology in their final reports?

As described in the Data Analysis section, two researchers coded the qualitative data focusing on the frequency and accuracy of the CT concepts, which were collected from participants' final reports after implementing their course projects. Table 3 illustrates a few examples of how in-service teachers wrote about the terminology of CT skills.

Table 3. Examples of teachers' writing on the terminology of CT skills.

CT terminology	**Examples from qualitative data**	
	Used correctly	**Used incorrectly**
Abstraction	An example of pattern recognition used by the students is knowing that an animal classified as a mammal has to give live birth, have warm blood, have fur or hair, and breathe with lungs. Students used the process of **abstraction** to be able to filter out any unnecessary information that is not needed in order to introduce their newly discovered animal.	***Abstraction:*** The students reread the ending and we decided to ignore the entirety of Chapter 23 which is the last chapter of the novel. The students had lots of debate about whether or not the project should start from the moment Jonas leaves versus the last chapter. To help the students, we watched the last ten minutes of "The Giver" movie which really appealed to all the students. Due to some PG-13 thematic elements, I could not show the entire movie.
Algorithms	To develop solutions to solving this problem, the students will use **algorithmic thinking**. To gain an understanding of this process, I will ask the students to make a sandwich. In doing this, we will discuss the sequence and order of making a sandwich using **algorithmic thinking**. In using the Scratch program, code blocks are called scripts. A script is an ordered list of instructions that can also be called an **algorithm**. The character in the program is called a sprite. The stage refers to the background of the story or the game.	***Algorithms:*** Students used the tutorials for adding saved images as sprites and backdrops in Scratch.
Decomposition	This was followed by having students give verbal directions in pairs to accomplish a simple task such as writing "hello" with a pen. This introduces students to some of the concepts of computational thinking by asking students to engage in **decomposition** and breaking the task down into smaller parts.	When coding using cups as a hands-on manipulative, scholars were able to recognize patterns to create the codes and **decomposition** to solve premade codes.
Pattern recognition	Teacher reviewed **patterns** in strings of shapes to remind students of the concept of **patterns**. The teacher explained to students that **pattern recognition** can make coding easier. The teacher asked students to open their Scratch codes to look for **patterns**. The teacher explained to students how to use code to make their Sprites repeat actions. Students demonstrated using Scratch code the concept of repeating an action in their digital storyboard.	The students will use **pattern recognition** to help with coding the movements and speech for each background to help make the coding more organized and appropriate for each scene.
Debugging	To test their thinking, students had opportunities to try out the command language created by other groups – they worked collaboratively to **debug** any steps and provided feedback to their peers for ways to make the process more efficient for other users.	/

Table 4 shows the numbers and percentages of the terms that were used correctly, incorrectly, or not mentioned at all. It is noticeable that most teachers used two CT terms correctly, algorithms and decomposition. However, only 59% of teachers used the terms abstraction and pattern recognition correctly. Furthermore, most teachers did not mention debugging at all, possibly due to the term being absent from the final report's question prompt. The finding highlights the need to emphasize certain CT terms, specifically abstraction, pattern recognition, and debugging, in future iterations.

Table 4. Usage of the CT terminology in teachers' final reports (n = 29).

CT terminology	Used correctly		Used incorrectly		Absent	
	n	%	n	%	n	%
Abstraction	17	59%	4	14%	8	28%
Algorithms	25	86%	1	3%	3	10%
Decomposition	23	79%	3	10%	3	10%
Pattern recognition	17	59%	3	10%	9	31%
Debugging	4	14%	/	0%	25	86%

Additionally, the researchers ran multiple Pearson's correlation tests using the demographic variables and the data from the survey, test, and final reports. However, no statistically significant correlation was found. This finding revealed that no relationships were found between the demographic variables, survey results, tests, and usage scores. Moreover, it means that the self-reported data from the CT perceptions survey did not correlate with the performance-based data from the CT test and terminology usage scores.

5. Discussion

5.1 Impact on In-service Teachers' CT Perceptions

The purpose of this study was to investigate in-service teachers' perceptions and development of CT skills in an online graduate emerging technologies course. Data analysis indicated that participants reported that they developed some aspects of their CT skills, such as problem-solving and creativity. Moreover, the change in their perceptions of problem-solving had a large effect size. These findings demonstrated that the online course had a positive impact on teachers' perceptions of CT skills, especially problem-solving and creativity. These results were also motivating since the course modules were designed to focus on creative computing with ample opportunities for problem-solving. Similar findings were found in other virtual PD programs (Jocius et al., 2021; Mouza et al., 2022).

Nevertheless, at the same time, teachers' perceptions of collaborative learning and critical thinking skills decreased after taking the course. One plausible reason might be the lack of peer coding opportunities. The authors recognized the benefits of peer coding as evidenced by findings in the field (Campe et al., 2020; Hanks et al., 2011). Even so, since this course was an online course, it was challenging to design peer coding activities that allowed multiple in-service teachers to program the same project due to various reasons such as lack of time and lack of proper Web 2.0 tools for peer coding. Jocius et al. (2021) used Snap! Pair programming and live support methods in their virtual PD program, which might be promising strategies to use. The authors also plan to explore live peer coding tools like Glitch.com and Twitch.tv for future iterations. Furthermore, this finding warrants more research on peer coding in online courses and the effectiveness of various tools and approaches for peer coding activities in various learning modalities.

While the effect size is small, there is evidence that these creative computing activities have the potential for fostering more creativity in the classroom. All computational projects in the course were designed to be open-ended with inclusive pedagogies in mind, to ensure that all participants could be creative in how they express their ideas and identities. Creative computing is an emerging branch of computer science that is gaining recognition through the integration of coding, interactive art, and making (Blikstein, 2018). This approach is less used in research and practice, but deserves more attention for it involves aesthetic experience, design-based learning, and storytelling (Hsu et al., 2018). The computational tools and devices used in this study are just one feasible way of enabling teachers to engage in creative computing while also making connections between CT and their subject areas. The authors recognize that there are other creative computing curricula that are publicly available (Creative Computing Lab, n.d.) and encourage teachers and teacher educators to explore how CT can be used to foster creativity in the classroom.

5.2 Impact on In-service Teachers' Development of CT Skills

Besides examining in-service teachers' perceptions of CT skills, the authors also analyzed the pre and post-test scores on CT skills. Findings revealed that overall in-service teachers improved their test scores after the modules, which demonstrated the development of CT skills. These results infer that the modules are effective in developing in-service teachers' CT skills. Several design factors might contribute to the modules' effectiveness. First, the course content was chunked to build on knowledge from previous modules. In-service teachers used Scratch, a block coding programming language, to create their digital storytelling projects first. Once they developed foundational CT and coding skills using block-based coding, they wrote codes on the Microsoft Makecode platform to program their BBC micro:bit. Last, they transitioned to breadboarding and creating their robotic pet, which was more challenging due to the need to troubleshoot both the digital code and the physical electrical components. To summarize, the projects were purposefully designed to follow an easy-to-difficult progression in order to achieve maximal improvement (Wisniewski et al., 2019).

Another design feature is the synchronous "Hour of Code" office hours, which were offered twice a week for in-service teachers to create, discuss code, and hang out with the course instructor. Although these sessions were optional, in-service teachers joined the sessions from time to time. Moreover, these sessions were recorded for in-service teachers to watch anytime anywhere. This method offered in-service teachers more instructional time and opportunities to ask questions, create, and troubleshoot in a synchronous group setting. Providing live support and prioritizing teachers' engagement have been justified as useful strategies for virtual professional learning in the literature (Jocius et al., 2021; Mouza et al., 2022).

A third design feature is the open-ended course projects, which utilized a "low threshold, high ceiling" approach. This strategy allows in-service teachers to engage in a variety of projects and provides room for them to consider their contexts and subject areas. To facilitate this method, the course instructor curated and created ample course materials that matched teachers' different abilities and learning preferences. Future research should examine the design features of such a course, propose instructional models, and design criteria to help teacher educators better design such courses.

Nonetheless, results from the descriptive data revealed that there was a big gap in the testing scores of these in-service teachers. Some teachers earned full marks on the pre and/or post-tests while other teachers scored relatively low for both tests. This result is somewhat alarming because it shows that some in-service teachers are not well-equipped with enough CT skills and it will be challenging for them to design CT-related curricula. It also indicates that more preparation on the knowledge and application of CT is needed.

Pedagogical approaches that might be helpful to facilitate further preparation or professional development efforts are adaptive learning (Hooshyar et al., 2021), personalized learning (Moon et al., 2020), and instructional technology coaching (Garvin et al., 2019; Israel et al., 2015). The authors recommend teacher educators pay attention to the gap in teachers' prior knowledge of CT and coding and design preparation and professional development accordingly.

5.3 Correlations between Self-reported and Performance-based Data

Pearson's Correlation tests revealed no statistically significant correlations between the demographic variables, self-reported data, and performance-based data. In other words, in-service teachers' perceptions of their CT skills did not correlate with their actual CT skills demonstrated in the performance-based data. Furthermore, there was no correlation between the two types of performance-based data, the CT test scores and the CT terminology scores. These findings have direct implications for future research, which could explore the correlation between other self-reported data, such as CT attitudes and self-efficacy, and various types of performance-based data measuring CT skills and CT implementation. In addition, more validated and standardized instruments are needed to measure teachers' CT implementation.

5.4 Beyond the Four Cornerstones of Computational Thinking

As demonstrated by the findings of the qualitative data, terminology related to the four cornerstones of computational thinking (i.e., abstraction, decomposition, pattern recognition, and algorithms) were used by the majority of participants. While these cornerstones were established early in the development of CT frameworks, the concepts related to CT skills and practices have expanded to include numerous other concepts such as debugging, selecting tools, automation, computational modeling, and data practices (Mills et al., 2021). As teacher educators expand the learning of CT in teacher preparation and professional development programs, it is crucial to look beyond the four cornerstones to ensure teachers and students receive a solid foundation in the concepts and practices that will prepare them for later engagement in CS. For example, professionals in CS engage in an iterative process of testing, debugging, and evaluating to ensure their programs function as designed. Similar to learning how to play a musical instrument, both CT and CS require practice and repetition in order to improve skills, develop fluency, and accomplish larger goals.

The authors recommend that those developing professional development and courses related to CT should investigate frameworks that move beyond the four cornerstones and include a broader range of CT skills and practices (e.g., Grover & Pea, 2018; Mills et al., 2021). While the four cornerstones initially serve as a good introduction to short-term professional development, the concepts associated with CT have widely expanded over the past 15 years. Additionally, more emphasis should be placed on developing a conceptual understanding of abstraction, which Jeanette Wing (2010) considers to be the most high-level thought process in CT. Teacher educators should provide ongoing professional development that seeks to cultivate a deeper understanding of CT and CS concepts with the goal of achieving a higher degree of K–12 integration.

6. Limitations

Limitations of this study include a relatively small sample size of 29 participants, of which 24 completed both surveys. Despite the small sample, researchers were able to produce meaningful results from the data across various statistical tests. Another limiting factor includes the use of a self-reported survey instrument to measure in-service teachers' CT perceptions before and after taking the course. All participants were enrolled in an emerging technology course as part of an Instructional Technology graduate program. As a result, participants likely identified as advocates for technology in the classroom and may have more experience with CT than teachers enrolled in other graduate programs. While CT was included as the focus of the second module, the concepts and terminology are reinforced throughout the entire course. This includes a CT section in the final written Creative Computing Project report. This study design focused on the change in CT perceptions and skills before and after the course, further studies are needed to measure the impact of individual modules or topics. Furthermore, this study took place as part of an asynchronous online course, thus findings may not be generalizable to synchronous, in-person, or hybrid settings.

7. Conclusion

This study found that in-service teachers enrolled in an online asynchronous graduate emerging technologies course were able to improve their CT problem-solving and creativity skills through a series of learning modules and activities with large effect sizes, which indicates the effectiveness of a virtual course. Despite these gains, participants reported

a decrease in their collaborative learning and critical thinking skills, however, with a small effect size. Most teachers were able to correctly apply the terms algorithms and decomposition in their final reports. However, only 59% of teachers correctly used the term abstraction and pattern recognition, and most teachers did not mention debugging at all.

In general, more needs to be done to help in-service teachers develop their CT skills. As this study has demonstrated, it is possible for in-service teachers to develop these skills asynchronously and online with a certain degree of success. However, more research is needed to better understand how to facilitate the development of CT collaborative learning and critical thinking skills in different teaching and learning formats, such as face-to-face, hybrid, and especially virtual. Those teaching CT skills should model and practice the correct use of terminologies, such as abstraction and pattern recognition, which were the most frequently misused terms in this study. In addition, greater emphasis should be placed on testing and debugging in order to move beyond the four cornerstones of CT. More empirical research is needed that addresses how in-service teachers develop and implement their CT skills. In addition, course developers should engage in design-based research to help the academic community better understand how teachers can develop a deeper understanding of CT, implement CT skills in their subject areas, and cultivate a sustained interest in CS.

References

Agalianos, A., Noss, R., & Whitty, G. (2001). Logo in mainstream schools: The struggle over the soul of an educational innovation. *British Journal of Sociology of Education, 22*(4), 479–500. https://doi.org/10.1080/01425690120094449

Ames, M. G. (2018). Hackers, computers, and cooperation: A critical history of Logo and constructionist learning. In *Proceedings of the ACM on Human-Computer Interaction, 2*(18), 1–19. https://doi.org/10.1145/3274287

Barcelos, T. S., Muñoz-Soto, R., Villarroel, R., Merino, E., & Silveira, I. F. (2018). Mathematics learning through computational thinking activities: A systematic literature review. *Journal of Universal Computer Science*, 24(7), 815–845.

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

BBC Bitesize. (n.d.). Introduction to computational thinking. https://www.bbc.co.uk/bitesize/guides/zp92mp3/revision/1

Blikstein, P. (2018). *Pre-college computer science education: A survey of the field* [Report]. Google LLC. https://goo.gl/gmS1Vm

Bower, M., Wood, L. N., Lai, J. W., Highfield, K., Veal, J., Howe, C., Lister, R., & Mason, R. (2017). Improving the computational thinking pedagogical capabilities of school teachers. *Australian Journal of Teacher Education, 42*(3), 53–72. https://doi.org/10.14221/ajte.2017v42n3.4

Bull, G., Garofalo, J., & Hguyen, N. R. (2020). Thinking about computational thinking: Origins of computational thinking in educational computing. *Journal of Digital Learning in Teacher Education, 36*(1), 6–18. https://doi.org/10.1080/21532974.2019.1694381

Brennan, K. (2015). Beyond technocentrism. *Constructivist Foundations, 10*(3), 289–296. https://constructivist.info/10/3/289.brennan

Campe, S., Denner, J., Green, E., & Torres, D. (2020). Pair programming in middle school: variations in interactions and behaviors. *Computer Science Education, 30*(1), 22–46. https://doi.org/10.1080/08993408.2019.1648119

Cansu, S. K., & Cansu, F. K. (2019). An overview of computational thinking. *International Journal of Computer Science Education in Schools, 3*(1), 1–11. https://doi.org/10.21585/ijcses.v3i1.53

Code.org, CSTA, & ECEP Alliance. (2022). *2022 State of computer science education: Accelerating action through advocacy*. https://advocacy.code.org/stateofcs

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*, 37–46. https://doi.org/10.1177/001316446002000104

Creative Computing Lab. (n.d.). *Creative computing curriculum*. Harvard Graduate School of Education. https://creativecomputing.gse.harvard.edu/guide/

CSTA (2017). *K–12 Computer science standards*. Retrieved from https://drive.google.com/file/d/1-dPTAI1yk2HYPKUWZ6DqaM6aVUDa9iby/view

Dufva, T., & Dufva, M. (2016). Metaphors of code—structuring and broadening the discussion on teaching children to code. *Thinking Skills and Creativity, 22*, 97–110. https://doi.org/10.1016/j.tsc.2016.09.004

Garvin, M., Killen, H., Plane, J., & Weintrop, D. (2019, February). Primary school teachers' conceptions of computational thinking. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 899–905). https://doi.org/10.1145/3287324.3287376

Goode, J. (2008, March). Increasing diversity in K–12 computer science: Strategies from the field. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 362–366). https://doi.org/10.1145/1352322.1352259

Gretter, S., Yadav, A., Sands, P., & Hambrusch, S. (2019). Equitable learning environments in K–12 computing: Teachers' views on barriers to diversity. *ACM Transactions on Computing Education (TOCE), 19*(3), 1–16. https://doi.org/10.1145/3282939

Gray, L. E. (1983). TECC/8: A Teacher Education and Computing Center. *Teacher Education Quarterly, 10*(4). 8–21.

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

Grover, S., & Pea, R. (2018). Computational thinking: A competency whose time has come. In S. Sentance, E. Barendsen, & C. Schulte (Eds.) *Computer science education: Perspectives on teaching and learning in school* (pp. 19–38). Bloomsbury.

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

Guven, G., & Kozcu Cakir, N. (2020). Investigation of the opinions of teachers who received in-service training for Arduino-assisted robotic coding applications. *Educational Policy Analysis and Strategic Research, 15*(1), 253–274. https://doi.org/10.29329/epasr.2020.236.14

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education, 21*(2), 135–173. https://doi.org/10.1080/08993408.2011.579808

Hello Ruby. (2019, September 2). *Episode 02: computational thinking* [Video]. YouTube. https://www.youtube.com/watch?v=K3vwRQCfTHc

Hickmott, D., Prieto-Rodriguez, E., & Holmes, K. (2018). A scoping review of studies on computational thinking in K–12 mathematics classrooms. *Digital Experiences in Mathematics Education, 4*(1), 48–69. https://doi.org/10.1007/s40751-017-0038-8

Hinton, P. R., Brownlow, C., McMurray, I., & Cozens, B. (2004). *SPSS Explained*. Routledge Inc. East Sussex, England.

Hooshyar, D., Pedaste, M., Yang, Y., Malva, L., Hwang, G. J., Wang, M., Lim, H., & Delev, D. (2021). From gaming to computational thinking: An adaptive educational computer game-based learning approach. *Journal of Educational Computing Research, 59*(3), 383–409. https://doi.org/10.1177/0735633120965919

Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education, 126,* 296–310. https://doi.org/10.1016/j.compedu.2018.07.004

Huang, W., & Looi, C. K. (2021). A critical review of literature on "unplugged" pedagogies in K–12 computer science and computational thinking education. *Computer Science Education, 31*(1), 83–111. https://doi.org/10.1080/08993408.2020.1789411

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

ISTE (2016a). *ISTE standards for educators*. https://www.iste.org/standards/for-educators

ISTE (2016b). *ISTE standards for students*. https://www.iste.org/standards/iste-standards-for-students

Jaipal-Jamani, K., & Angeli, C. (2017). Effect of robotics on elementary preservice teachers' self-efficacy, science learning, and computational thinking. *Journal of Science Education and Technology, 26*(2), 175–192. https://doi.org/10.1007/s10956-016-9663-z

Jocius, R., Joshi, D., Albert, J., Barnes, T., Robinson, R., Cateté, V., Dong, Y., Blanton, M., O'Byrne, I., & Andrews, A. (2021, March). The virtual pivot: Transitioning computational thinking PD for middle and high school content area teachers. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (pp. 1198–1204). https://doi.org/10.1145/3408877.3432558

K–12 Computer Science Framework. (2016). https://k12cs.org

Kalogiannakis, M., Tzagkaraki, E., & Papadakis, St. (2021, March 18-19). A systematic review of the use of BBC micro:bit in primary school. In *Proceedings of the 10th Virtual Edition of the International Conference New Perspectives in Science Education*, 379–384, Florence, Italy. https://doi.org/10.26352/F318_2384-9509

Kennedy, C., Kraemer, E. T., & Benson, L. C. (2021). Active learning techniques for computing education. In C. Mouza, A. Yadav, & A. Ottenbreit-Leftwich (Eds.) *Preparing pre-service teachers to teach computer science: Models, practices, and policies* (pp. 3–28). Information Age Publishing, Inc.

Ketelhut, D. J., Mills, K., Hestness, E., Cabrera, L., Plane, J., & McGinnis, J. R. (2020). Teacher change following a professional development experience in integrating computational thinking into elementary science. *Journal of Science Education and Technology, 29*(1), 174–188. https://doi.org/10.1007/s10956-019-09798-4

Kotsopoulos, D., Floyd, L., Khan, S., Namukasa, I. K., Somanath, S., Weber, J., & Yiu, C. (2017). A pedagogical framework for computational thinking. *Digital Experiences in Mathematics Education, 3*(2), 154–171. https://doi.org/10.1007/s40751-017-0031-2

Logo Foundation. (2014). *Logo history*. https://el.media.mit.edu/logo-foundation/what_is_logo/history.html

Mason, S. L., & Rich, P. J. (2019). Preparing elementary school teachers to teach computing, coding, and computational thinking. *Contemporary Issues in Technology and Teacher Education, 19*(4), 790–824. https://citejournal.org/volume-19/issue-4-19/general/preparing-elementary-school-teachers-to-teach-computing-coding-and-computational-thinking

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

Microsoft Makecode. (2022) *Microsoft Makecode for micro:bit* (Version 4.0.18) [Computer software]. Microsoft. https://makecode.microbit.org/

Miles, M. B., Humberman, A. M., & Saldaña, J. (2019). *Qualitative data analysis: A methods sourcebook* (4th ed.). Sage Publishing.

Mills, K., Coenraad, M., Ruiz, P., Burke, Q., & Weisgrau, J. (2021, December). *Computational thinking for an inclusive world: A resource for educators to learn and lead*. Digital Promise. https://doi.org/20.500.12265/138

Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record, 108*(6), 1017–1054. https://doi.org/10.1111/j.1467-9620.2006.00684.x

Montiel, H., & Gomez-Zermeño, M. G. (2021). Educational challenges for computational thinking in K–12 education: A systematic literature review of "Scratch" as an innovative programming tool. *Computers, 10*(6), 69. https://doi.org/10.3390/computers10060069

Moon, J., Do, J., Lee, D., & Choi, G. W. (2020). A conceptual framework for teaching computational thinking in personalized OERs. *Smart Learning Environments, 7*(1), 1–19. https://doi.org/10.1186/s40561-019-0108-z

Mouza, C., Mead, H., Alkhateeb, B., & Pollock, L. (2022). A Virtual Professional Development Program for Computer Science Education During COVID-19. *TechTrends, 66*(3), 436–449. https://doi.org/10.1007/s11528-022-00731-y

NGSS Lead States (2013). Next generation science standards: For states, by states. The National Academies Press, Washington, DC. https://nap.nationalacademies.org/catalog/18290/next-generation-science-standards-for-states-by-states

Orton, K., Weintrop, D., Beheshti, E., Horn, M., Jona, K., & Wilensky, U. (2016, July). Bringing computational thinking into high school mathematics and science classrooms. In C. K. Looi, J. L. Polman, U. Cress & P. Reimann (Eds.), *Transforming Learning, Empowering Learners: The International Conference of the Learning Sciences (ICLS) 2016* (pp. 705–712). Singapore: International Society of the Learning Sciences. https://repository.isls.org/handle/1/183

Papadakis, S. (2021). The impact of coding apps on young children Computational Thinking and coding skills. A literature review. *Frontiers in Education, 6*, 657895. https://doi.org/10.3389/feduc.2021.657895

Papadakis, S. (2022). Can preschoolers learn computational thinking and coding skills with ScratchJr? A systematic literature review. *International Journal of Educational Reform*, 1–34. https://doi.org/10.1177/10567879221076077

Papadakis, S., Vaiopoulou, J., Sifaki, E., Kalogiannakis, M., & Stamovlasis, D. (2021). Attitudes towards the use of educational robotics: Exploring pre-service and in-service early childhood teacher profiles. *Education Sciences, 11*(5), 204. https://doi.org/10.3390/educsci11050204

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*(2), 137–168. https://doi.org/10.1016/0732-118X(84)90018-7

Ranganathan, P., Pramesh, C. S., & Aggarwal, R. (2017). Common pitfalls in statistical analysis: Measures of agreement. *Perspectives in Clinical Research, 8*(4), 187–191. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5654219/

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Resnick, M., & Ocko, S. (1990). *LEGO/logo--learning through and about design*. Cambridge: Epistemology and Learning Group, MIT Media Laboratory.

Sands, P., Yadav, A., & Good, J. (2018). Computational thinking in K–12: In-service teacher perceptions of computational thinking. In M. S. Khine (Ed.), *Computational thinking in the STEM disciplines* (pp. 151–164). Springer, Cham. https://doi.org/10.1007/978-3-319-93566-9_8

Schmidt-Crawford, D. A., Lindstrom, D. & Thompson, A. D. (2018). Coding for teacher education: A recurring theme that requires our attention. *Journal of Digital Learning in Teacher Education, 34*(4), 198–200. https://doi.org/10.1080/21532974.2018.1499992

Scratch. (n.d.). *Scratch* (Version 3.0) [Computer software]. Scratch Foundation. https://scratch.mit.edu/

Solomon, C. (1988). *Computer environments for children: A reflection on theories of learning and education*. MIT Press.

Sun, L., Hu, L., & Zhou, D. (2021). Which way of design programming activities is more effective to promote K-12 students' computational thinking skills? A meta-analysis. *Journal of Computer Assisted Learning, 37*(4), 1048–1062. https://doi.org/10.1111/jcal.12545

Uston, K. (1983, October). 9,250 Apples for the teacher. *Creative Computing, 9*(10), 178–183. https://www.atarimagazines.com/creative/v9n10/178_9250_Apples_for_the_teac.php

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: towards an agenda for research and practice. *Education and Information Technologies, 20*(4), 715–728. https://doi.org/10.1007/s10639-015-9412-6

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

Weintrop, D. (2021). The role of block-based programming in computer science education. In *Understanding computing education (Vol 1)*. Proceedings of the Raspberry Pi Foundation Research Seminar series. https://rpf.io/seminar-proceedings-2020

Wilkerson, M. H., & Fenwick, M. (2017). Using mathematics and computational thinking. In C. V. Schwarz, C. Passmore, & B. J. Reiser (Eds.), *Helping students make sense of the world using next generation science and engineering practices* (pp. 181–204). Arlington, VA: National Science Teachers' Association Press.

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

Wing, J. M. (2010). *Computational thinking: What and why?* [Unpublished manuscript]. Computer Science Department, Carnegie Mellon University. https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

Wisniewski, M. G., Church, B. A., Mercado, E., Radell, M. L., & Zakrzewski, A. C. (2019). Easy-to-hard effects in perceptual learning depend upon the degree to which initial trials are "easy." *Psychonomic Bulletin & Review, 26*(6), 1889–1895. https://doi.org/10.3758/s13423-019-01627-4

Yadav, A., Gretter, S., Good, J., & McLean, T. (2017a). Computational thinking in teacher education. In P. Rich & C. B. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 205–220). Springer, Cham. https://doi.org/10.1007/978-3-319-52691-1_13

Yadav, A., Good, J., Voogt, J., & Fisser, P. (2017b). Computational thinking as an emerging competence domain. In M. Mulder (Ed.), *Competence-based vocational and professional education* (pp. 1051–1067). Cham: Springer.

Yadav, A., Stephenson, C., & Hong, H. (2017c). Computational thinking for teacher education. *Communications of the ACM, 60*(4), 55–62. https://doi.org/10.1145/2994591

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011, March). Introducing computational thinking in education courses. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 465–470). https://doi.org/10.1145/1953163.1953297

Yağcı, M. (2019). A valid and reliable tool for examining computational thinking skills. *Education and Information Technologies, 24*(1), 929–951. https://doi.org/10.1007/s10639-018-9801-8

Yaşar, O., Maliekal, J., Veronesi, P., Little, L., & Vattana, S. (2015, March). Computational pedagogical content knowledge (CPACK): integrating modeling and simulation technology into STEM teacher education. In *Society for Information Technology & Teacher Education International Conference* (pp. 3514–3521). Association for the Advancement of Computing in Education (AACE). https://www.learntechlib.org/primary/p/150489/

Yin, R. K. (2017). *Case study research and application: Design and methods* (6th ed.). Sage Publishing.

Yurkofsky, M. M., Blum-Smith, S., & Brennan, K. (2019). Expanding outcomes: Exploring varied conceptions of teacher learning in an online professional development experience. *Teaching and Teacher Education, 82*, 1–13. https://doi.org/10.1016/j.tate.2019.03.002