Core Competencies of K-12 Computer Science Education from The Perspectives of College Faculties and K-12 Teachers

Meina Zhu[1]

Cheng Wang[1]

[1]Wayne State University, USA

DOI: https://doi.org/10.21585/ijcses.v6i2.161

Abstract

Given the increasing need for employees with computational skills, understanding the core competencies of K-12 computer science (CS) education is vital. This phenomenological research aims to identify critical factors of CS education in K-12 schools from the perspectives and visions of CS faculties in higher education and teachers in K-12 schools. This study adopted a phenomenological research design. The researchers conducted a semi-structured interview with 13 CS faculties and K-12 CS teachers in Michigan and analyzed the data using thematic analysis. The findings indicated that: (1) the core competencies for K-12 CS education include problem-solving through computational thinking, math background, and foundational programming skills, and (2) what is essential is not the programming languages taught in K-12 schools but computational thinking, which enables the learners to easily transfer from one language environment to another. The findings provide important implications for K-12 CS education regarding the core competencies and programming languages to be taught.

Keywords: K-12 computer science education, core competencies, computational thinking, problem-solving, math

1. Introduction

As computers become one of the essential social fabrics that construct the infrastructure of our world, the need for K-12 computer science (CS) education is increasing. The CS education community made K-12 CS education standards in 2017 which "delineate a core set of learning objectives designed to provide the foundation for a complete computer science curriculum and its implementation at the K-12 level" (CSTA, n.d.). For each state, defining CS and establishing rigorous K-12 CS standards is one of the nine policies to be developed according to the Code.org advocacy coalition. Michigan adopted the Computer Science Teachers Association (CSTA) K-12 CS standards in 2019 (Code.org, CSTA, & ECEP Alliance., 2020). However, only 37% of Michigan high schools offered CS courses during the 2019-2020 academic year (Michigan Department of Education, 2020). A majority of schools do not have a clear understanding of CS education and its needs, which may hinder their adoption and implementation of CS education. Given that CS faculty in higher education usually hold a doctoral degree in the field and have in-depth knowledge about CS education, their perceptions of core CS competencies and expectations from high school graduates can provide insights into K-12 CS education. At the same time, K-12 CS teachers are the practitioners in the field, and thus their experiences and feedbacks are as important as that of CS faculties in higher education. Therefore, this study aims to identify key factors in pre-college CS education from the perspectives and visions of CS college faculties and K-12 CS teachers so that CS researchers, educators, experts, policymakers, and other stakeholders in the field can provide better K-12 CS education to students.

2. Literature Review

2.1 K-12 CS Education

Given the importance of computing technology in modern society, the needs of employees with CS skills were increasing (Barr & Stephenson, 2011). CS has been widely adopted in diverse scientific and humanity areas. Nowadays, scientific and research innovations in social and humanity areas could not be accomplished without computers or computing skills (Gal-Ezer & Stephnson, 2014). Thus, CS knowledge and skills become essential in the 21st century.

CS was defined as the area that studies computers and algorithms, such as principles, hardware, and software design, applications, and evaluation by the Association for Computing Machinery (ACM) and the Computer Science Teachers Association (CSTA) K-12 standards task force (Seehorn et al., 2016). CS education in K-12 settings can develop students' higher-order thinking skills, reflective thinking skills, and critical thinking skills (Tran, 2019) for problem-solving (Ministry of Education, 2014).

K-12 CS education has been implemented in several countries. For example, Webb et al. (2017) investigated K-12 CS education curricula in five counties and found that these countries have agreed on the importance of CS and the advantages of having CS education as early as possible in K-12. However, there are still multiple concerns regarding K-12 CS education. The very first one is whether it is necessary to teach K-12 students CS since not all students will pursue CS majors or careers in the future (Grover & Pea, 2013). Next, if K-12 CS education is necessary, what are the core competencies to be developed among students? Lastly, given that curricula in K-12 is already packed and the time and space for CS education is limited, which kinds of programming languages and environments will be more appropriate and effective in implementation?

2.2 Problem Solving and Computational Thinking in CS Education

One of the primary purposes of CS is to solve computational problems. The problem-solving approach is often related to computational thinking (CT) (Grover & Pea, 2013; Israel et al., 2015), which has long been considered as one of the key factors in CS education. CT refers to using an algorithmic approach to solve real-world problems, which is a necessary skill in different contexts and situations (Shute et al., 2017). The term, CT, was introduced by Seymour Papert's book (1980) regarding the programming language LOGO. Later, Wing (2006) defines CT as "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (p. 33). Wing (2006) considers CT just as one of the analytical abilities like reading, writing, and arithmetic. Since 2006, CT has become a popular term in the CS education field. Regarding the components of CT skills, Selby and Woollard (2013) define CT as five subcomponents models: abstraction, decomposition, algorithm, generalization, and evaluation. While the definitions of CT were inconsistent and vague (Korkmaz et al., 2017), there is a common understanding of CT education: with CT skills, students can think like CS professionals to solve problems through steps such as decomposition, pattern recognition, and algorithm (Barr & Stephenson, 2011).

Give its values in modern society, CT is considered not only as one of the skills that could change students' thinking in different fields (Papert, 1980) but a universal skill for every student to obtain (Barr & Stephenson, 2011; Voogt et al., 2015). The OECD and UNESCO state that CT is a necessary skill for digital citizens (Organisation for Economic Cooperation and Development, 2018; World Economic Forum, 2015). International Society for Technology in Education (ISTE) (2018) has included CT as one of the learning standards so that students can use computational methods to solve problems in the digital era. Moreover, CT is proposed to be included in compulsory education in the report from European Commission (Bocconi et al., 2016). Thus, some countries have included CT in their curricula, such as the U.K. and Australia (Bower et al., 2017).

2.3 Programming Languages in CS Education

Programming is one of the fundamental skills in CS and a vital tool to develop CT skills (Grover & Pea, 2013; Lye & Koh, 2014). Research indicated that introducing CT to students in their earlier years is important as it could equip students with critical thinking skills (Tran, 2019). The programming approach has been implemented for CT education in pre-school (e.g., Çiftci & Bildiren, 2020) and K-12 education (e.g., Schmidt, 2016). For example, Çiftci and Bildiren (2020) found that programming can help develop 4-5-year-old preschool students' problem solving and cognitive skills. Irish and Kang (2018) found that integrating programming into other learning activities can engage students in both programming and general subjects learning.

Consequently, programming languages and environments play an important role in K-12 CS education. The question of which language should be taught in K-12 has been a controversial topic. Currently, popular programming languages such as Python, Java, C, and C++ are widely used in industry and academia (TIOBE, 2021). These languages are also called textual programming languages as they are primarily written in text editors. Therefore, programmers should learn not only logical thinking but also the syntax of the language. Although textual programming languages may be difficult to approach for novice learners, research has indicated that students who learn textual programming language as the introductory programming language can transit to other textual languages easier as they move forward (Enbody

& Punch, 2010). Thus, they recommend that it is preferable to have textual programming language for novice learners, given that the textual programming languages are universally used in real life.

On the contrary, the non-textual programming languages and environments, which comprises diverse visual formats such as diagrams, flowcharts, and coding blocks (Dehouck, 2016), are expected to be easy enough for beginners to get started and extensive enough to meet the needs for advanced programming (Grover & Pea, 2013). Visual programming environments that are widely used include Scratch, Game Maker, Code.org, Alice, Kodu, etc. Some of the visual programming languages, such as Scratch and Alice, are block-based languages in the programming environments, of which students can drag and drop coding blocks to the workspace. Thus, novice CS learners can focus on the computational concepts and logic without being bothered by the syntax (Bau et al., 2017; Kelleher & Pausch, 2005). Some research argues that visual programming languages might be more appropriate for novice learners as they are easier to learn (Bau et al., 2017; Chen et al., 2020; Malan & Leitner, 2007). For example, Chen et al. (2020) analyzed data from 10,000 undergraduate students who enrolled in CS courses and found that students whose first programming language was visual performed better than did students whose first programming language was textual when the programming languages were first introduced in K-12 stages. Moreover, the visual languages and environments provide scaffolds and enable knowledge transfer. Research indicates that visual programming languages are used in K-12 CT education. For example, Hsu et al. (2018) and Lockwood and Mooney (2017) find that many schools have utilized visual programming languages to teach CT skills. Other studies indicate that using visual programming languages to teach students CT skills is effective in elementary education (The Horizon Report, 2017). Application of visual program languages in K-12 CS education is found to significantly improve students' understanding of computational concepts and computation practices (Saez-Lopez et al., 2016), logical thinking skills (Lindh and Holgersson 2007), and problem-solving skills in general (Chou 2018).

Given the mixed opinions of introducing visual or textual programming languages in K-12 CS education, Xu et al. (2019) conduct a meta-analysis on the block-based versus textual programming on student learning outcomes by reviewing 13 publications. They find a small effect size in favor of block-based programming languages on cognitive learning outcomes and suggest more research on the effectiveness of using block-based programming languages for novice learners in the future.

To sum up, the epistemology of K-12 CS education, including its necessity, its core competencies as well as programming languages that should be taught in its implementation, still need to be clarified. Thus, this study aims to explore the K-12 CS students' core competencies and programming languages that should be learned in K-12 from CS professors' and K-12 teachers' perspectives.

The following research questions guide this study:

(1) What are the CS competencies expected from K-12 students from the perspectives of CS faculties in higher education and teachers in K-12 schools?

(2) What are the programming languages to be introduced to K-12 students from the perspectives of CS faculties in higher education and teachers in K-12 schools?

3. Method

To answer the two research questions, we use the qualitative interview data coming from the Computer Science Teachers in Michigan (CSTIM) project that led by the two authors of the present study. The CSTIM project adopts a mixed-method design (Creswell & Plano-Clark, 2017) to investigate the necessity of K-12 CS education, core competencies of CS learners, current trends and issues related to K-12 CS education, and teaching strategies as well as teachers' competencies to teach CS in K-12 schools. The project is comprised of three components. First, from the ideological perspective, the researchers aim to capture the fundamental values in CS education and the core competencies for K-12 CS students through semi-structured interviews of CS college faculties and K-12 CS teachers. Second, from the practical perspective, the researchers investigate the CS teaching strategies, K-12 teacher competencies, and professional development approaches through semi-structured interviews. Third, based on the analysis results of the first two phases, the researchers extract the keywords for mining data from Twitter to examine the current trends and issues related to K-12 CS education. This current study focuses on the first component of our entire CSTIM project.

In its qualitative part, the CSTIM project applies a phenomenological research design (Giorgi & Giorgi 2003). In particular, we conduct semi-structured interviews with eight CS faculties in higher education and five CS teachers in

K-12 schools to understand CS education phenomenon. We choose the qualitative approach was because it can benefit the discovery and interpretation of the investigated phenomena (Yu & Hai, 2005). Moreover, the semi-structured interviews provide rich information about respondents' experiences and perceptions of CS education.

3.1 Instruments

An interview protocol can provide a framework to guide the semi-structured interviews (Patton, 2015). The interview protocol is developed from the literature review regarding K-12 CS education (i.e., CSTA n.d.; K-12 Computer Science Framework Steering Committee, 2016; Wing, 2006). The interview protocol for higher education CS faculties includes 13 questions related to CS learners' competencies, challenges, strategies, and expectations in K-12 CS education (Zhu & Wang, 2023). Please see the detailed interview protocol in Appendix. The first question is about the interviewee's background information. Questions two to seven are related to interviewees' perceptions of CS learners' competencies, programming languages, effective strategies, and challenges while teaching CS students and typical successful CS learners. Questions eight to ten are related to interviewees' opinions of the necessity of K-12 CS education, curricular, and programming languages. Question 11 to 13 are about K-12 CS teachers' competencies to teach K-12 CS courses.

The interview protocol for K-12 teachers includes 11 questions regarding their understanding of K-12 CS standards and competencies, experiences, and feedbacks in K-12 CS education, contents, and programming languages they used in classrooms (Zhu & Wang, 2023). The first question is about the interviewee's background information. Questions two to three are about their understanding of CS standards and CS education. Questions four to seven are related to K-12 CS learners' competencies, curricular, programming language, assessment approach. Questions eight to question 11 are related to K-12 CS teachers' teaching challenges, resources and support, and professional development. Given that this study adopts a semi-structured interview method, follow-up questions are asked based on each individual interviewee's response.

3.2 Participants

The participants of the CSTIM project include both faculties in higher education and K-12 teachers. The criteria for selecting the faculties in higher education include: (1) having at least three years' CS teaching experience, (2) have taught undergraduate freshman or sophomore courses, and (3) their universities are located in Michigan state. The criteria for choosing K-12 teachers are: (1) having experience of teaching CS courses in the past three years and (2) their schools are located in Michigan state. The researchers gather CS college faculties' emails from their university websites and send an email invitation to participate in our study. Eight CS college faculties accept the invitation and participate in the study. They come from six universities in Michigan, including the University of Michigan, Wayne State University, Oakland University, Central Michigan University, Western Michigan University, and Eastern Michigan University. Seven out of eight CS instructors held a Ph.D. degree in CS, and one was working on his Ph.D. degree. To recruit K-12 CS teachers, the researchers use a snowball sampling method, and five K-12 CS teachers accept the invitation and participate in our study. The five interviewees include three high school teachers and two middle school teachers. Among the five teachers, only one had a bachelor's degree in CS. The rest of them did not have CS related degrees. Detailed information about the interviewees is shown in Table 1.

Table 1. Participant information

Pseudonyms	Occupations	Institutions	Educational background	Gender
Arthur	Teacher	High school	Ph.D. in physics	Male
Diego	Teacher	High school	Bachelor in CS & Master's degree in arts and teaching	Male
Eli	Teacher	Middle school	N/A	Male
Kate	Teacher	High school	CS workshops	Female
Lucy	Teacher	Middle and high school	Bachelor with a math major and CS minor; master's degree in teaching	Female
Aiden	Associate Professor	Higher education	Ph.D. in CS	Male
Daxton	Instructor	Higher education	Working on a Ph.D. degree in CS	Male

David	Associate Professor	Higher education	Ph.D. in CS	Male
James	Professor	Higher education	Ph.D. in CS	Male
Kash	Associate Professor	Higher education	Ph.D. in CS	Male
Lawrence	Associate Professor	Higher education	Ph.D. in CS	Male
Luke	Assistant Professor	Higher education	Ph.D. in CS	Male
Tong	Assistant Professor	Higher education	Ph.D. in CS	Male

3.3 Data Collection Procedures

The interview protocol is shared with the interviewees at least one day before the interview for them to prepare for the answers. Each interview lasts approximately 30 minutes. Since the CSTIM project is conducted during an ongoing pandemic of COVID-19, the face-to-face interview is infeasible. The interviews are primarily audio-recorded via Zoom, an online conference tool, along with the Smart Recorder app installed on the researchers' smartphone as a secondary means to secure the data collection. The recordings are transcribed verbatim. To appreciate their participation, the researchers provide a $25 Amazon gift card after each participant validates his or her interview data.

3.4 Data Analysis

The researchers use thematic analysis (Braun & Clarke, 2006) to analyze the interview data. The thematic analysis enables researchers to identify patterns across datasets in order to describe the invested phenomenon (Guest, 2012). It includes six phases for researchers to form themes from the qualitative data (Bernard & Ryan, 2009). The first phase includes familiarizing with the data. Researchers read the data repeatedly to identify the patterns in the data. In the second phase, codes are generated by labeling words, phrases, sentences, and paragraphs. In the third phase, closely related codes are combined into themes. Fourth, the themes are reviewed and revised. Some themes might be grouped together, while others might be split. In the fifth phase, themes are defined and named. Finally, the results are reported.

In the present study, two researchers independently conduct the first five phases of the thematic analysis. Then we meet to discuss the individual analysis results. The discrepancies are discussed until we reach a consensus. The final coding scheme on K-12 CS educational ideology includes two concepts, i.e., K-12 CS competencies and K-12 programming languages (see Table 2).

Table 2. Coding themes

Theme	Concept	Code
K-12 CS educational ideology	K-12 CS competencies	Problem-solving with computational thinking Math background Foundational programming skills
	K-12 recommended programming languages	From block-based visual programs to syntax-based language Python, Java, C++ Specific language does not matter

3.5 Trustworthiness

Several strategies are used to ensure the trustworthiness of the study, such as credibility, dependability, transferability, and confirmability (Lincoln & Guba, 1985). First, credibility refers to what extent the data reflect the 'truth' of the phenomenon (Erlandson et al., 1993). In the present study, first-level member validation is conducted with all the interviewees to verify the accuracy of the transcripts. Among the 13 interviewees, 12 participants confirm the transcripts or make minor revisions. One participant does not respond to our request. Second, dependability refers to the replicability of the research in the same or similar contexts (Erlandson et al., 1993). This study ensures dependability by recording the procedures and problems of the project in documents. Third, transferability represents to what extent the study findings can be applied in other different contexts (Erlandson et al., 1993). In this study, a thick description of the research context, participants, and results is provided. Fourth, confirmability refers to the extent of avoiding biases (Erlandson et al., 1993). The present study documents all the research processes to make sure the original data sources can be traced back.

4. Findings

Regarding the context of this study, 12 out of 13 interviews believe that CS education is necessary for K-12 schools. The only exception is James, a professor in higher education, who thinks that math is better than CS to cultivate problem-solving skills and CT (at least for kindergarteners through to the eighth graders), and it is not the best way to force the students to learn CS which will bring burden to them.

Turning to the first research questions, thematic analysis results of the interview data related to K-12 CS ideology include two primary concepts: K-12 CS competencies and K-12 recommended programming languages. The following section will present each concept and code in detail.

4.1 Concept I: K-12 CS Competencies

The data analysis results in three primary codes – problem-solving with CT, math background, and foundational programming skills – that help construct the concept of K-12 CS competencies.

4.1.1 Code I: Problem-solving with Computational Thinking

11 out of the 13 interviewees emphasize that the core CS competency of K-12 students is problem-solving with CT. The data analysis results indicate that the skills of solving real-world problems are expected from CS students at all levels. For example, Aiden shares his opinions regarding the importance of problem-solving skills for CS students in general:

> "These things [hot fields in CS] go through cycles. Things that are hot today will not be hot tomorrow. So, a good way to prepare students is to give them this core competency so that they have really competent, independent, fundamental ideas of computer science, which is how the problem can be solved using our computing systems." (Aiden, a CS associate professor)

In particular, for K-12 students, problem-solving is considered as one of the core competencies in CS education as well. K-12 students are expected to master the core knowledge and skills in CS subjects. In addition, problem-solving skill is not only important for CS learning but also critical for learning in other subjects. For example, Eli, a K-12 CS teacher, expresses his opinion on CS education and highlights the importance of "solving problems and come up with solutions." Similarly, Kash emphasizes the importance of problem-solving skills in K-12:

> "I think at high school, instead of teaching them programming, it's better to teach them problem-solving because learning syntax is not a big deal. Whoever has dwelled more problem-solving skills are more successful because the fundamental concept of programming languages is the same. So, if we are building a problem-solving skill at high school, just teach them to have one simplest language, Python, that is more than enough rather than introducing too many programming languages." (Kash, a CS associate professor)

The approaches to solving problems vary. Among different approaches for problem-solving, in CS education, CT is one of the important methods. Five interviewees explicitly state that CT is an essential approach for problem-solving. Other interviewees implicitly explain the importance of CT without using the specific term CT. For example, Lawrence shares his opinions of CT and problem solving:

> "I feel like there's an advantage in students being exposed to computational thinking of solving a problem. When I say computational thinking, I mean solving a problem. The way that you do it computationally is to break it down into steps and solve it step by step. I think that's a little different from the kind of problem-solving techniques you learned in the other fields." (Lawrence, a CS associate professor)

Despite that the CT concept is used in CS education, as mentioned earlier, the definition and meaning of CT have not reached a consensus. CS educators have some fundamental understanding of CT. David, a CS associate professor, explained his understanding about CT "it's more like how to know, solve the problem using a computer, basically." And Aiden, explains his understanding of CT:

> "For this computational thinking, first of all, they need to develop some awareness whenever they encounter a problem. Once they have an awareness, the next step is to develop a mindset that problems can be solved using a computer so that it becomes second nature. When they encounter a new problem, they think I can do this, and

> then try to formulate some real solutions and maybe even develop basic programming skills for high school students." (Aiden, a CS associate professor)

In addition, CT is considered an important approach for problem-solving no matter whether the students will pursue a CS major in higher education or not after high school. Interviewees think that in real life, CT is helpful for people who work in different fields. For instance, Lucy and Lawrence express their thoughts on CT:

> "I think it's incredibly useful. These are skills that go beyond just the computer science field, but in everyday life in any field. They're going to understand how to break down a problem, how to work on the solution, and how to design something to be a solution for some tasks. This is incredibly important." (Lucky, a K-12 CS teacher)

> "I think it'd been exposed to computational thinking is valuable in the same way that students take chemistry in high school… I still like every basic knowledge about the world and how it works, and the scientific method is valuable. I think having some idea about how computational things work and how to do computational problem solving is useful. I think a lot of students are going to have to use computation later in life. So, these are useful skills for them." (Lawrence, a CS associate professor)

4.1.2 Code II: Math Background

Seven out of 13 interviewees highlight the importance of math background and consider math as the key cornerstone of CS education in both K-12 and higher education settings. One of the interviewees, Eli, states "Computer Sciences is another language, but it's inherently about. I mean, it's mathematical, it's algorithmic it's breaking things apart in baby steps. And then figuring out the variety of options." (Eli, a K-12 Middle school CS teacher). Similarly, James says, "but of course, learning, you know, studying math, learning math is key. Critical to good computational thinking." (James, a CS professor). Kash further emphasizes the importance of math:

> "From here, we have, you know, a clue that this guy is more fit for IT, but a person who has done some programming and has solved problems is really good at mathematics, so did this [being good at mathematics] is at least a clue for parents as well as, you know, the candidate themselves that they are maybe a better fit for, you know, computer science. So, I think teachers first need to focus on this thing." (Kash, a CS associate professor)

Despite that math is considered one of the foundational subjects in CS education, not all CS students have sufficient knowledge for CS learning. Six out of the 13 interviewees mention that a common challenge for some CS students is that they lack a math background. Per David, "as I said that they have to learn how to think computationally and solve problems. And that's the difficult part, and that requires a lot of math background." (David, a CS associate professor). He further explains:

> "I think the main issue is that the students that select especially at our university, that choose to go in computer science, they select the major but lack the appropriate background. So, they have, you know, are having a hard time, you know, with their first classes like the data structures, especially those that are used a lot [in other CS courses]. So, their math background is very poor, and they struggle with that. So that's one big challenge…misconception is very, you know, damaging in a way because they are disappointed because they think that they just have to learn the language, but that's just a tool, as I said to them, they have to learn how to think computationally and solve problems. And that's the difficult part, and that requires a lot of math background and upgrades, and they said [those are] the classes they avoid anyway so [in the past]." (David, a CS associate professor)

4.1.3 Code III: Foundational Programming Skills

Besides math, a few interviewees think another important component of K-12 CS education is programming skills. For example, Aiden states, "it's like building a foundation, a strong foundation of CS core competency comprising things like programming." (Aiden, a CS associate professor) In addition, Lucy says, "I think the goal that we're hitting on for middle to upper school has been programming and building algorithms, debugging, breaking down code." (Lucy, a K-12 Middle school CS teacher) Students without foundational programming skills usually encounter setbacks when they enter college, as elaborated by Lawrence:

> "So, about half of our students coming to our program are coming from community colleges, are transferring from some other colleges. And about half of this. I mean, it's every year. It's almost exactly 50%. It's been that

> way for several years, um, and about half of them are first-time [CS] students…I tell students that, you know, if this is your first time, you know, taking a programming course, you know, other people maybe have more experience than you. That doesn't mean that they're better at doing this, and you are right. This means that, you know, they've been doing it longer. So, I think sometimes students get discouraged if, either this is my hypothesis, they get discouraged if they see that it's easy for some students and it's hard for them, but it might be easy for the other students because they've already, like you said, taken it in high school. I don't know the exact numbers, but we definitely have a reasonable number of students who do not have any real exposure to programming before they join our program. But we also have students who have taken programming before in high school." (Lawrence, a CS associate professor)

4.2 Concept II: K-12 Recommended Programming Languages

To cultivate computational thinking for problem-solving, our interviewees also express their epistemology about the programming languages that might be used in K-12 education to serve this specific purpose. The section below demonstrates three code categories regarding programming teaching programming languages in K-12 CS education: (1) from block-based programming to syntax-based languages; (2) syntax-based languages: Python, Java, C++; and (3) specific programming language does not matter.

4.2.1 Code I: From Block-Based Programming to Syntax-Based Languages

Regarding specific programming languages that should be taught in K-12, eight out of 13 interviewees suggest starting from block-based visual programming tools, such as Scratch and code.org. For example, Aiden says, "so something like scratch will be very effective for young children. As for these young children, say grade six or below this kind of range. The priority should be about engagement, making it fun for them so they can see the problems can be solved for older children like high school children, then yes, absolutely." (Aiden, a CS associate professor) Eli, a K-12 teacher, says, "I used code.org or scratch. That's all block-based programming. I want something to be manageable or something to be user-friendly, and I want whenever they come up with a solution." Similarly, Kate, Lucy, Arthur, and Diego echo the idea of using Block-based programming tools to teach K-12 students CS subjects.

In addition, four interviewees also mention that it might be better to start with block-based visual programming tools, such as Scratch, then transit to syntax-based programming languages, such as Python and other languages. For example, David says,

> "I would say that if you start with a simple [programming language]. For elementary school, you have to choose something graphical. There are a lot of environments out there, like maybe Scratch and Alice, and there are a lot of others. And as you go up, let's say, middle school, you can start introducing nonvisual programming environments. And you can go, you know, it doesn't really matter, if Java or Python or C++ will be more difficult to learn, I think Python is good enough." (David, a CS associate professor)

This idea is separately advanced by other interviewees. Per, Kash, "for the sixth graders, definitely you know, it's good to introduce block-based (visual) programming ideas, but for a high school again, my opinion is to introduce Python." (Kash, a CS associate professor). Daxton holds a similar opinion:

> "They're going to have to know how to do sequence selection iteration, whether it's graphical or not. I think it [block-based visual programming] is good for K-2 to K-5. But once they get to K-6 through 12, I think it should be a text-based programming language." (Daxton, a CS instructor in higher education)

4.2.2 Code II: Syntax-based Languages: Python, Java, C++

In particular, the specific text-based programming languages that are encouraged included Python, Java, and C++, etc. For example, Kash says, "Python is appropriate for K-12 CS education. In Python, students don't receive too many syntax issues, and they can focus on improving problem-solving techniques." (Kash, a CS associate professor). Moreover, Kate and Lisa mention that their schools have already taught syntax-based programming language in high school.

> "At the high school, we use Python and Java. We use programming languages and tools that they can utilize. Now we teach an AP Computer Science class. So that does have to be the Java language because that's what the test is on. But those are all very marketable software tools that they can use, whether it's in college or if they

decide college is not for them. They can also use in the real world." (Kate, a K-12 high school CS teacher)

"We also use Python to begin to develop the understanding of what is the language and how do you learn it. By ninth grade, they're doing full-on Python. They can take Java after ninth grade. And so those are both options for continuation" (Lucy, a K-12 middle school CS teacher)

4.2.3 Code III: Specific Programming Language does not Matter

Overall, four interviewees think that the specific programming language is not that important compared to CT skills for problem-solving. K-12 students can learn CT skills without using particular "real" programming language, as indicated in previous cites from Aiden and Kash.

Students can learn any programming language, such as Python, to learn CT skills. Once they master one programming language, the knowledge can be transferred when learning other programming languages. As Daxton, Lucy, and Kate explain below:

"I think, from what I've seen, there is a lot of emphasis on knowing what language to teach. That is not important. The language is coming today; you learn Python, but two years from now, Python will probably disappear, and other languages will come. So more important is to know one language. Don't focus on learning how to use that language to program things, so computational thinking is more important than the language itself. A language is a tool." (Daxton, a CS associate professor)

"We try very hard to create a basis of understanding the language, not a specific language, but just what a programming language is and does, and then that way, as languages change, students can still apply the same knowledge to any language." (Lucy, a K-12 CS teacher)

"We have a beginning and intermediate [class], and then we have the AP [CS] class. So, we have different levels. And once you learn how to do as…if statement, once you know how to do a for loop, you know, you can apply it with any language." (Kate, a K-12 high school CS teacher)

5. Discussion

The primary goal of the current study is to explore the necessity of K-12 CS education, K-12 CS students' core competencies, and programming languages that should be learned in K-12 from CS professors' and teachers' perspectives. The findings of this study reveal that while most interviewees believe that K-12 CS education is necessary, problem-solving skills using computational thinking are the top important competencies in K-12 CS education. In addition, K-12 students should have basic math background and foundational programming skills. Regarding the programming languages, this study found that interviewees suggested starting with a block-based visual programming language and then moving to textual languages, such as Python, Java, C++. However, the specific language was not considered as important as CT and problem-solving skills.

In terms of the importance of computational thinking and problem-solving skills in CS education, the finding of this study aligns with the statements from the prior researchers (Grover & Pea, 2013) that the problem-solving approach is often related to CT skills. Regarding the concepts of CT, some interviewees have a common understanding of using a computational approach, such as abstraction, decomposition, algorithm, and generalization, to solve problems, which aligns with the categories from Selby and Woollard (2013). In addition, CT skills not only could be used in the CS field but also be helpful for other subjects. Researchers explored approaches of integrating CT skills in K-12 through diverse approaches. Sengupta et al. (2013) proposed a theoretical framework for integrating computational thinking in K-12 science education. The framework includes three stages: (1) scientific inquiry, (2) algorithm design, and (3) engineering. Moreover, Yadav et al. (2016) provided suggestions for instructional technologies and training experts for integrating CT into other subjects in K-12. Kwon et al. (2021) implemented CT in primary education using problem-based learning approach and examined the development of CT skills maong students.

Interviewees in this study highlight the importance of math knowledge in CS education. Interviewees consider that math lays the foundations for advanced CS learning, which concurs with argument from Beaubouef (2002) and Konvalina, Wileman, and Stephens (1983). In reality, both CS and math subjects require students to have logical thinking skills. Beaubouef (2002) stated that math is critical in diverse perspectives in CS, including problem-sovling, programming, computer hardware and architecture, CS theory, and softeware engineering. Regarding whether math

should be the prerequisite of CS education, especially in K-12 education, no consensus has been achieved yet. Further research can examine the relationship between math and CS education.

The findings of this study also indicate that programming skill is important in K-12 CS education. This finding concurs with the statements from prior researchers, such as Grover and Pea (2013) and Lye and Koh (2014). Programming is an important tool to develop CT skills for problem-solving. Consequently, deciding on programming languages to be taught in K-12 CS education is essential. This study finds that interviewees hold different perspectives. Some suggest using block-based programming tools such as Scratch for each CT skill. Others suggest teaching some specific widely-used textual programming languages, such as Python, Java, C++, etc., as suggested by TIOBE (2021). Among these diverse opinions, interviewees in this study also suggest letting students start using block-based programming tools in lower grades and gradually introduce textual programming languages in higher grades. Despite that the last perspective compromises the first two opinions, more details need to be explored regarding when and how the transition from visual programming languages to textual programming languages should be put into practice.

Although the interviewees share opinion regarding diverse programming languages, some also emphasize the specific programming languages taught is not that important as long as students can learn CT skills. They highlight that once students learn one programming language to develop their CT skills, they can easily transfer what they have learned to new programming languages. Future research may further examine whether using different programming languages influence their outcome of obtaining CT skills and how to efficiently and effectively transfer between different programming languages.

6. Limitations and Future Research

Some limitations exist in this study. First, this study used the self-reported interview data from volunteers as the data source, which may have bias. Further research can incorporate other data sources, such as policy documents, reports, and observations to confirm or refine findings from this study. Second, the interviewees are from the CS professors and K-12 CS teachers in Michigan State. The generalization of the study findings from this study should be cautious. The status of K-12 CS education in different states is heterogeneous, which may influence their CS professors' and teachers' perspectives. Last, the participants of this study are CS professors and teachers, which leave the key stakeholders of K-12 CS education, students, outside of the conversation. Future research can further explore students' opinions of K-12 CS education.

7. Conclusions

This study's findings indicate the core of CS education includes problem-solving and CT skills, math background, and foundational programming skills. CT is considered an important skill to solve problems, which supports Wing's (2006) definition. Therefore, CT is critical in K-12 CS education. Math may be one of the foundation subjects for CS education. In addition, pre-college experiences in computer programming are important. However, the specific programming language is not the critical element as long as students master CT and problem-solving skills. K-12 students may start from the visual programming languages and then transfer to textual programming languages. The study findings deepen our understanding of K-12 CS education, which helps educators and policymakers making decisions regarding K-12 CS education.

References

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. Doi: 10.1145/1929887.1929905

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017, June). Learnable programming: Blocks and beyond. In *the Communications of the ACM, 60*(6), 72–80. https://doi.org/10.1145/3015455

Beaubouef, T. (2002). Why computer science students need math. *ACM SIGCSE Bulletin, 34*(4), 57-59. https://doi.org/10.1145/820127.820166

Bernard, H. R., & Ryan, G. W. (2009). *Analyzing qualitative data: Systematic approaches.* SAGE publications.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., & Punie, Y. (2016). Developing

computational thinking in compulsory education. *European Commission, JRC Science for Policy Report,* 68. https://komenskypost.nl/wp-content/uploads/2017/01/jrc104188_computhinkreport.pdf

Bower, M., Wood, L. N., Lai, J. W., Highfield, K., Veal, J., Howe, C., ... & Mason, R. (2017). Improving the computational thinking pedagogical capabilities of school teachers. *Australian Journal of Teacher Education (Online)*, *42*(3), 53-72. https://search.informit.org/doi/abs/10.3316/informit.767807290396583

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology, 3*(2), 77-101.

Chen, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2019). The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages. *Computer Science Education, 29*(1), 23-48. https://doi.org/10.1080/08993408.2018.1547564

Chou, P.-N. (2018). Skill development and knowledge acquisition cultivated by maker education: Evidence from Arduino-based educational robotics. *EURASIA Journal of Mathematics, Science and Technology Education, 14*(10), 1–15. https://doi.org/10.29333/ejmste/93483

Çiftci, S., & Bildiren, A. (2020). The effect of coding courses on the cognitive abilities and problem-solving skills of preschool children. *Computer Science Education, 30*(1), 3-21. https://doi.org/10.1080/08993408.2019.1696169

Code.org, CSTA, & ECEP Alliance. (2020). 2*020 State of Computer Science Education: Illuminating Disparities*. https://advocacy.code.org/stateofcs

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

CSTA (n.d.). *Computer science standards.* CSTA. Retrieved from https://www.csteachers.org/page/standards

Dehouck, R. (2016). The maturity of visual programming. http://www.craft.ai/blog/the-maturity-of-visual-programming/

Enbody, R. J., & Punch, W. F. (2010, March). Performance of Python CS1 students in mid-level non-Python CS courses. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 520-523). https://doi.org/10.1145/1734263.1734437

Erlandson, D. A., Harris, E. L., Skipper, B. L., & Allen, S. D. (1993). *Doing naturalistic inquiry: A guide to methods*. Sage.

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

Gal-Ezer, J., & Stephenson, C. (2014). A tale of two countries: Successes and challenges in K-12 computer science education in Israel and the United States. *ACM Transactions on Computing Education (TOCE), 14*(2), 1-18. https://doi.org/10.1145/2602483

Giorgi, A. P., & Giorgi, B. M. (2003). The descriptive phenomenological psychological method. In P. M. Camic, J. E. Rhodes, & L. Yardley (Eds.), *Qualitative research in psychology: Expanding perspectives in methodology and design* (pp. 243–273). American Psychological Association

Gretter, S., & Yadav, A. (2016). Computational thinking and media and information literacy: An integrated approach to teaching twenty-first century skills. *TechTrends, 60*(5), 510–516. https://doi.org/10.1007/s11528-016-0098-4

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

Guest, G. (2012). *Applied thematic analysis*. Sage.

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

Irish, T., & Kang, N. H. (2018). Connecting classroom science with everyday life: Teachers' attempts and students' insights. *International Journal of Science and Mathematics Education, 16(*7), 1227-1245. Doi: 10.1007/s10763-017-9836-0

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

K-12 Computer Science Framework Steering Committee. (2016). K-12 computer science framework. ACM. doi:https://doi.org/10.1145/3079760

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys, 37*(2), 83–137. https://doi.org/10.1145/1089733.1089734

Konvalina, J., Wileman, S. A., & Stephens, L. J. (1983). Math proficiency: A key to success for computer science students. *Communications of the ACM, 26*(5), 377-382. https://doi.org/10.1145/69586.358140

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior, 72,* 558-569. https://doi.org/10.1016/j.chb.2017.01.005

Kwon, K., Jeon, M., Guo, M., Yan, G., Kim, J., Ottenbreit-Leftwich, A. T., & Brush, T. A. (2021). Computational thinking practices: Lessons learned from a problem-based curriculum in primary education. *Journal of Research on Technology in Education,* 1-18. https://doi.org/10.1080/15391523.2021.2014372

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

Lindh, J., & Holgersson, T. (2007). Does lego training stimulate pupils' ability to solve logical problems?. *Computers & Education, 49*(4), 1097-1111. https://doi.org/10.1016/j.compedu.2005.12.008

Lockwood, J., & Mooney, A. (2017). Computational thinking in education: Where does it fit? A systematic literary review. *arXiv preprint arXiv:1703.07659.*

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. https://doi.org/10.1016/j.chb.2014.09.012

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM Sigcse Bulletin, 39*(1), 223-227. https://doi.org/10.1145/1227504.1227388

Ministry of Education. (2014). Computer science: A new curriculum in reform. http://cms.education.gov.il/NR/rdonlyres/0E091CFA-8E73-4C24-96A7-0A6D23E571EA/189697/resource_849760831.pdf

Organisation for Economic Co-operation and Development. (2018). The future of education and skills: Education 2030. *OECD Education Working Papers 23.* https://doi.org/10.1111/j.1440-1827.2012.02814.x

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

Patton, M. Q. (2015). Qualitative research & evaluation methods: Integrating theory and practice: The definitive text of qualitative inquiry frameworks and options (4th ed.). Thousand Oaks, California: SAGE Publications, Inc.

Saez-Lopez, J., Roman-Gonzalez, M., & Vazquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two-year case study using Scratch in five schools. *Computers & Education, 97*, 129–141. https://doi.org/10.1016/j.compedu.2016.03.003

Schmidt, A. (2016). Increasing Computer Literacy with the BBC micro: bit. *IEEE Pervasive Computing, 15*(2), 5-7. Doi: 10.1109/MPRV.2016.23

Seehorn, D., Pirmann, T., Batista, L., Ryder, D., Sedgwick, V., O'Grady-Cunniff, D., Twarek, B., Moix, D., Bell, J., Blankenship, L., Pollock, L., & Uche, C. (2016). CSTA K-12 Computer Science standards 2016 revised. ACM Press. https://dl.acm.org/doi/pdf/10.1145/2593249?casa_token=zOwW-U2zltcAAAAA:RR8hxGKWuykHfnSlZpB_7z4pMY1oFKSWIm9W8txVT-NE4KLKx4JlagcXvX1w0z84VvEIScrM3xln

Selby, C., & Woollard, J. (2013). *Computational thinking: The developing definition.* https://eprints.soton.ac.uk/356481/

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

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. Educational Research Review,

22, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Michigan Department of Education (2020, May). State of Computer Science Education in Michigan. https://www.michigan.gov/documents/mde/State_of_Computer_Science_Education_in_Michigan_Report_709699_7.pdf

The Horizon Report. (2017). K–12 edition. https://www.nmc.org/nmchorizon-k12/

TIOBE index. (2021). https://www.tiobe.com/tiobe-index

Tran, Y. (2019). Computational thinking equity in elementary classrooms: What third-grade students know and can do. *Journal of Educational Computing Research, 57*(1), 3-31. https://doi.org/10.1177/0735633117743918

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. Doi: 10.1007/s10639-015-9412-6

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when?. *Education and Information Technologies, 22*(2), 445-468. Doi: 10.1007/s10639-016-9493-x

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

Wong, G. K. W., & Cheung, H. Y. (2020). Exploring children's perceptions of developing twenty-first century skills through computational thinking and programming. *Interactive Learning Environments, 28*(4), 438-450. https://doi.org/10.1080/10494820.2018.1534245

World Bank. (2019). Children learning to code: Essential for 21st century human capital.

World Economic Forum. (2015). New vision for education unlocking the potential of technology.

Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: A meta-analysis study. *Computer Science Education, 29*(2-3), 177-204. https://doi.org/10.1080/08993408.2019.1565233

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,* 565-568. https://doi.org/10.1007/s11528-016-0087-7

Yu, P., & Hai, T. (2005). A focus conversation model in consumer research: The incorporation of group facilitation paradigm in in-depth interviews. *Asia Pacific Advances in Consumer Research, 6,* 337–344. https://www.acrwebsite.org/volumes/11931

Zhu, M., & Wang, C. (2023). K-12 Computer Science Teaching Strategies, Challenges, and Teachers' Professional Development Opportunities and Needs. *Computers in the Schools*, 1-22. https://doi.org/10.1080/07380569.2023.2178868

Appendix

Semi-structured Interview Questions – K-12

1. Please briefly introduce yourself.
2. Have you heard of CS standards in Michigan? Does your school make plans to meet the standards?
3. What is your understanding of CS education?
4. Which goals and which competencies are intended in K-12-CS Education?
5. What learning content will be/is delivered in K-12 CS Education?
6. Which programming languages and tools are used in K-12 schools?
7. Which types of assessments were used
8. Who is teaching CS?
9. What are the challenges/concerns about teaching CS in K-12?
10. Who do you seek help from when you encounter challenges?

11. What types of resources, support, or additional teacher training are provided in K-12 CS education?

Semi-structured Interview Questions - Higher education

1. Please briefly introduce yourself.
2. What are the future job opportunities for CS students after they graduate?
3. What goals and competencies are intended in each program/CS education in higher education?
4. What are the common programming languages and tools taught in the CS field in higher education?
5. What are the effective instructional strategies for teaching CS students in higher education? Would you mind giving me an example?
6. What are the challenges that you encountered teaching CS students in higher education?
7. Could you please describe a typical successful learner in CS?
8. Do you think CS in K-12 is necessary? Why?
9. If we plan to offer CS curricula in K-12, what competency do you think students could learn in K-12 to help students learn better in college?
10. What languages or tools should be taught in K-12?
11. What knowledge and skills do you think K-12 CS teachers should have to teach students successfully?
12. If they do not have such knowledge and skills, how do you think we can provide support to K-12 CS teachers?
13. Do you have any suggestions for K-12 CS educators?