



IJCSSES

Volume 5, No: 4
September 2022
ISSN 2513-8359

International Journal of Computer Science Education in Schools

Editors

Dr Filiz Kalelioglu

Dr Yasemin Allsop

www.ijcses.org

International Journal of Computer Science Education in Schools

September 2022, Vol 5, No 4

DOI: 10.21585/ijcses.v5i4

Table of Contents

	Page
Masiar BABAZADEH¹, Lucio NEGRINI¹ How is Computational Thinking Assessed in European K-12 Education? A Systematic Review	3 - 19
Carla DE LIRA¹, Rachel WONG², Olufunso OJE¹, Gabriel NKETAH¹, Olusola ADESOPE¹, Alireza GHODS¹ Summer Programming Camps – Exploring Project-Based Informal CS Education in a Rural Community	20 - 37
David AMIEL¹, Cynthia BLITZ¹ Computer Science Teacher Capacity: The Need for Expanded Understanding	38 - 47

How is Computational Thinking Assessed in European K-12 Education? A Systematic Review

Masiar BABAZADEH¹
Lucio NEGRINI¹

¹ Scuola Universitaria Professionale della Svizzera italiana, Dipartimento formazione e apprendimento, Laboratorio media e MINT, Locarno, Switzerland

DOI: 10.21585/ijcses.v5i4.138

Abstract

Computational thinking (CT) is seen as a key competence of the 21st century and different countries have started to integrate it into their compulsory school curricula. However, few indications exist on how to assess CT in compulsory school. This review analyses what tools are used to assess CT in European schools and which dimensions are assessed. We analysed 26 studies carried out in K-12 between 2016 and 2020 in Europe. The results indicate that 18 different tools have been used and they can be categorized into five groups: questionnaires, tests/tasks, observations, interviews and analysis of products. From the tools we analysed, more than 50 dimensions were assessed and the vast majority of those were closer to programming skills rather than CT per se. Based on these results it seems that a common operational definition of CT, a competence model that indicates which competences students should reach at which age, and a tool that allows all different facets of CT to be assessed are currently missing.

Keywords: computational thinking, assessment, k-12, computational thinking dimensions

1. Introduction

In recent years, in the wake of digitisation and automation of our society, computational thinking (CT) and more in general digital literacy have been seen as key competences of the 21st century (World Economic Forum, 2016). CT has been popularized by Wing (2006), and involves "solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science" (p.33). After Wing, different authors have proposed other definitions of CT or have tried to operationalise Wing's idea. For example, according to the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA), CT includes formulating problems; logically organising and analysing data; representing data through abstractions, models and simulations; automating problem resolution through algorithmic thinking; testing and improving the possible solutions and transferring the problem-solving process to a variety of problems (CSTA & ISTE, 2011).

Brennan and Resnick (2012) distinguish three dimensions of CT: computational concepts (the knowledge component) that include concepts that programmers use for example the variables; computational practices (the skills component) that include the problem solving practices that occurs in the process of programming; and computational perspectives (the attitude component) that include students' understandings of themselves, their relationships to others, and the technological world around them (Lye & Koh, 2014). Other authors define CT as "the ability to think with the computer-as-tool" (Berland & Wilensky, 2015, p.630) or as "students using computers to model their ideas and develop programs [...] and consider computer programming as one part of computational thinking" (Israel, Pearson, Tapia, Wherfel & Reese, 2015, p.264). Shute, Sun and Asbell-Clarke (2017) in a literature review, synthesize various definitions of CT and propose a framework for CT particularly for K-12. Their working definition of CT is: "The conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts"

(p.151) and formulated six main facets of CT: decomposition, abstraction, algorithm design, debugging, iteration, and generalization. According to Shute et al., (2017), "CT is primarily a way of thinking and acting, which can be exhibited through the use of particular skills, which then can become the basis for performance-based assessments of CT skills" (p.142). Consequently, computational thinking skills should be expressed in terms of competences, i.e. as a combination of knowledge, skills and attitudes that are expressed in a context (European Commission, 2016).

Due to the augmented importance of CT for our society, there have been increasing calls for CT and related concepts such as coding or programming to be integrated into European school curricula (Académie des Sciences, 2013; European Commission, 2016; Royal Society, 2012; Schweizerische Eidgenossenschaft, 2017). Several States therefore have, as part of curricula reforms, included CT and related concepts in compulsory schooling (e.g. England (UK), France, Finland, Italy, Germany, and Switzerland at a regional level) or are planning to introduce it in the next years (e.g. Norway, Denmark) (Bocconi et al., 2016; Bocconi, Chiocciariello & Earp, 2018).

Different approaches on how to introduce CT in schools exist (Chiocciariello & Olimpo, 2017). The most common approaches found in the literature are based on programming (e.g., Scratch), educational robotics, game design, or paper and pencil activities (Calmet, Hirtzig, & Wilgenbus, 2016). CT is often taught by asking students to create algorithms in order to solve exercises first, and open-ended problems after, or to create software artefacts like video games or animations for example. Those algorithms can be created without the use of technology (for example, by the means of unplugged activities where a student-programmer gives instructions to a student-robot to navigate through a maze), with visual programming languages or with textual programming languages (Da Cruz Alves, Gresse Von Wangenheim, & Hauck, 2019).

Despite the numerous suggestions from different didactic materials and tools to carry out CT in class, only a few indications exist on how to assess CT skills in K-12. A large body of literature published in recent years indicates different challenges in the development of widely accepted assessment methods and frameworks that encompass the complexity of CT (Brennan & Resnick, 2012; Denner, Werner, & Ortiz, 2012; Denning, 2017; Fronza, El Ioini, & Corral, 2017; Grover et al., 2017; Grover, Pea, & Cooper, 2015; Tikva & Tambouris, 2021; Zhong, Wang, Chen, & Li, 2016). These challenges are also due to the variety of CT definitions that make it difficult to develop a common and reliable assessment tool (Adams, Cutumisu, & Lu, 2019). The existing tools are often developed to measure single dimensions of CT highly linked to the tool adopted (e.g., programming concepts in Scratch, or debugging strategies with educational robots). This makes it even more difficult to measure CT in its entirety and therefore evaluate the effectiveness of the CT activities that are carried out in schools. This situation neither allows for an overview on the different approaches, nor to compare results across various studies (Shute et al., 2017). The assessment of CT is however essential in order to successfully implement CT in schools (Grover & Pea, 2013) since teachers need to collect evidence on what they propose to better understand their students' progress. Valid and reliable assessments tools also help to evaluate the effectiveness of different CT curricula (Basu, Rutstein, Xu, Wang & Shear, 2021). Tools to assess CT need to include all dimension of CT such as the assessment of understanding of programming or CT concepts alongside assessment of general problem-solving practices such as logical thinking, formulation of a problem as a set of computational steps, pattern recognition, abstraction and generalization, decomposition and modularization, data collection and organization; data-based decision making, and systematic incremental testing and debugging that are important in contexts beyond programming (Atmatzidou & Demetriadis, 2016; Barr, Harrison, & Conery, 2011; Basu, Rutstein, Xu, Wang & Shear, 2021; Csizmadia, Curzon, Dorling, Humphreys, Ng, Selby, & Woollard, 2015).

In the last few years, some reviews on CT assessment tools have been published. Those reviews have been carried out in Canada (Adams et al., 2019; Cutumisu, Adams, & Lu, 2019), Brazil (Da Cruz Alves et al., 2019) and United States (Tang, Yin, Lin, Hadad & Zhai, 2020). The review by Da Cruz Alves et al., (2019), however focuses on the tools that assess programming activities based on code and not on CT in a broader sense as defined in this paper. Tang et al., (2020) include studies done in colleges, high schools or teacher education, while Cutumisu et al., (2019) analyse studies done between 2014 and 2018. The cited reviews show a breadth of methods employed to assess CT (Cutumisu et al., 2019) that include four main forms: traditional assessment composed of selected- and/or constructed-response questions, portfolio assessments, surveys, as well as interviews, and claim that most of the CT assessment tools analyse concepts directly related to algorithms and programming (Tang et al., 2020). Another study of Çoban and Korkmaz (2021), highlights in the literature, "it has been seen that computational thinking is evaluated with different measurement tools. Many more methods such as scales, portfolio studies, coding, multiple choice tests, task-based tests, observations, and rubrics have been applied with different methodologies" (p. 2). There are

however no reviews focusing on contemporary studies conducted in European compulsory schools with a broader understanding of CT. A review on how CT is currently assessed in European schools and which tools are used can therefore help to make an overview on the different existing practices and to formulate implications for the field. In this paper, we intend to explore and describe the CT assessment approaches used in K-12 education in Europe focusing on the last years. The questions addressed are the following: (1) Which tools are used to assess CT in Europe? and (2) Which dimensions of CT are assessed?

This review will help to create an overview on different approaches and tools used to assess CT in K-12 in Europe and on the CT dimensions assessed. This knowledge could potentially help design and develop a reliable and valid CT assessment tool. The paper is structured as follows: Section 2 describes the methodology used to obtain the reviewed papers. Section 3 presents the results, aggregating the studies and illustrating the tools used, and the dimensions of CT assessed. Section 4 discusses the results and Section 5 presents the conclusions.

2. Methodology

For this systematic review we adopted the method for implementing reviews in the social sciences by Petticrew and Roberts (2006). Specifically, we followed these steps: (1) research questions were formulated; (2) the search terms were defined, and appropriate databases were selected; (3) inclusion and exclusion criteria were formulated; (4) the obtained papers were screened and selected; (5) the data to answer the research questions were extracted.

2.1 Definition of Search Terms and Selection of Database

For this review we have consulted seven databases: ERIC, PsycInfo, Scopus, Web of Science, Google Scholar, ACM Digital Library, and ProQuest Dissertations and Theses Global. The selection of these databases includes journals involving educational research as well as computer sciences research. We decided to also include Google Scholar and ProQuest Dissertations and Theses Global to also reach grey literature and dissertations. The search terms used are the following:

- Query 1: "computational thinking" AND ("K-12" OR "primary school" OR "secondary school") AND "assessment"
- Query 2: "computational thinking" AND ("assessment" OR "test" OR "evaluation" OR "exam" OR "measure") AND ("K-12" OR "primary school" OR "elementary school" OR "secondary school" OR "middle school")
- Query 3: "computational thinking" AND "assessment"

We started with Query 1; however, we noticed that the term assessment could also be used with a synonym like "test", "evaluation", "exam", or "measure". We decided therefore to do a second query including also these terms. The first two queries however did not allow studies carried out in educational systems that use other names to refer to K-12 education other than "primary", "elementary", "secondary", or "middle school". In order to be sure to also include educational systems that use other names we carried out a third query with only the terms "computational thinking" and "assessment". This query is a superset of the previous two queries, with a much broader spectrum of results. Even though such a query ended up including many out-of-scope papers, we decided to keep the results and filter them ourselves in order not to exclude a priori papers that use another terminology. The research in fact yielded a total of 30432 papers. After removing duplicates, we obtained a corpus of 13872 papers. The consultation of the databases has been concluded during the first week of December 2020.

2.2 Inclusion and Exclusion Criteria

To select the articles, we used the following inclusion criteria:

- We decided to include papers conducted from 2016 onward (10 years after Wing's seminal work). We are aware that this criterion could represent a limitation since we are excluding studies carried out before 2016. This decision was made in order to limit the number of papers and in order to include only actual and more recent assessment tools that are used in European schools.
- The study needs to be conducted in Europe since we are interested in the assessment tools used in European schools. This criterion was formulated in order to avoid including assessment tools that relate to instructional practices that are not present in European schools. We are aware that the school systems in Europe are different from each other, the instructional practices related to CT (robotics, coding, unplugged activities) are however very similar between European countries.

- The paper has to be written in English. This criterion could also represent a limitation since European literature could be published in different languages. The most common language used in this field of research however is English.
- The paper needs to have the full text available.
- The study has to be done in the context of formal K-12 education.
- The paper explains which dimensions of CT are assessed.
- The paper presents a tool/test to assess CT.
- The tool has been tested in class. The paper should present a tool that has been applied in class. Papers on theoretical reflections on how to assess CT without a tool tested in class have been excluded.

2.3 Screening of Papers

The screening process can be seen in Figure 1. All papers have been screened applying the inclusion criteria. A first screening round based mostly on the abstracts and metadata of the papers lead us to eliminate studies carried out before 2016, non-English papers and papers on studies conducted in non-European countries. After applying these first criteria, we obtained 175 papers. In a second round, the inclusion criteria were applied to the full-text versions of the 175 selected papers. The execution of this second round resulted in 26 papers that match the inclusion criteria.

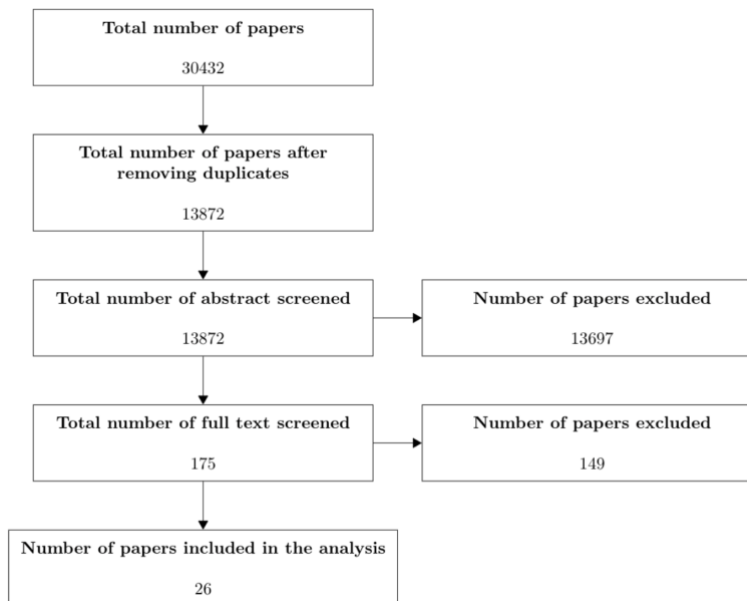


Figure 1. Screening process and stages.

2.4 Data Extraction

The 26 selected papers were read a second time more in depth and a series of information was selected in order to answer the research questions. For all 26 papers we extracted the following data:

- Authors and date
- Type of the paper (e.g., scientific article, conference paper, dissertation, ...)
- Nation where the study was conducted
- Name of the assessment tool implemented
- Form of the tool (e.g., paper based, computer based, test, questionnaire, ...)
- Length of the assessment (e.g., number of items)
- CT dimensions that were assessed
- Number of pupils in the study
- School grade in which the study was conducted

3. Results

The presentation of the findings was divided into three main topics: description of the selected studies, presentation of the assessment tools used in the different studies, and a report on the CT dimensions assessed by the different tools.

3.1 General Description of the Selected Studies

Of the 26 studies selected, 7 were conducted in Spain 4 of which were by the same research team. In Turkey 6 studies were carried out. In the UK there were 3. In Germany, Greece and Italy, 2 studies were carried out in each. Chiazzese, Arrigo, Chifari, Lonati & Tosto (2019) is an extended paper of the study Chiazzese, Arrigo, Chifari, Lonati & Tosto (2018). The last 4 studies were conducted in Czech Republic, Finland, Ireland, and in Slovenia. Seventeen studies were published as journal papers, 8 were conference papers and 1 as a chapter in a book. The number of pupils assessed with the CT tools ranges between 16 (Gillott, Joyce-Gibbons, & Hidson, 2020) and 1251 (Roman-Gonzalez, Perez-Gonzalez, & Jimenez-Fernandez, 2017). Sixteen studies have been carried out in primary schools and 10 in secondary education. The attribution to a grade is however dependent on the country where the study was conducted. In this study, in which we are aware that we could not represent all European school systems, we divided the school grades as follow: Preschool/Kindergarten (pupils aged below 5 years); Primary School (pupils aged between 6 and 11; Grades 1st-6th); Secondary School (pupils aged between 12-15; Grades 7th-10th). In the cases where the studies covered more grades, we have counted them according to the grade where the majority of the pupils were enrolled considering only the grades in compulsory schools. Table 1 shows all the selected studies ordered by school grade. The selected studies are also marked with an * in the reference list.

Table 1. Selected studies

Study	Nr. of pupils	Grade	Nation	Publication Type
Kalliopi and Michail, 2019	450	1st-2nd	Greece	Conference paper
del Olmo-Muñoz, Cózar-Gutiérrez, and González-Calero, 2020	84	2nd	Spain	Journal paper
Price and Price-Mohr, 2018	18	2nd-5th	UK	Journal paper
Leifheit, Jabs, Ninaus, Moeller and Ostermann, 2018	33	3rd- 4th	Germany	Conference paper
Chiazzese, Arrigo, Chifari, Lonati and Tosto, 2018	83	3rd-4th	Italy	Conference paper
Chiazzese, Arrigo, Chifari, Lonati and Tosto, 2019	51	3rd-4th	Italy	Journal paper
Bryndová and Mališů, 2020	90	3rd-8th	Czech Republic	Conference paper
Fagerlund, Häkkinen, Vesisenaho, and Viiri, 2020	57	4th	Finland	Journal paper
Kožuh, Krajnc, Hadjileontiadis, and Debevc, 2018	945	4th-6th	Slovenia	Journal paper
Pérez-Marín, Hijón-Neira and Bacelo, 2018	132	4th-6th	Spain	Journal paper
Yildiz Durak, 2018	110	5th	Turkey	Journal paper
Tonbuloğlu and Tonbuloğlu, 2019	114	5th	Turkey	Journal paper
Saez-Lopez, Roman-Gonzalez, and Vazquez-Cano, 2016	139	5th-6th	Spain	Journal paper
Allsop, 2019	30	5th-6th	UK	Journal paper
Kukul and Karatas, 2019	319	5th-7th	Turkey	Journal paper
Korucu, Gencturk, and Gundogdu, 2017	160	5th-8th	Turkey	Journal paper
Roman-Gonzalez, Perez-Gonzalez, and Jimenez-Fernandez, 2017	1251	5th-10th	Spain	Journal paper
Förster, Förster, and Löwe, 2018	22	6th	Germany	Conference paper

Segredo, Miranda, León and Santos, 2016	54	6th-14th	Spain	Conference paper
Román-González, Moreno-León, and Robles, 2017	148	7th-8th	Spain	Conference paper
Bati, Yetişir, Çalışkan, Güneş and Saçan, 2018	104	8th	Turkey	Journal paper
Román-González, Pérez-González, Moreno-León and Robles, 2018	314	8th-9th	Spain	Journal paper
Saritepeci and Durak, 2017	53	9th	Turkey	Chapter in a book
Garneli and Chorianoopoulos, 2018	34	10th	Greece	Journal paper
Gillott, Joyce-Gibbons and Hidson, 2020	16	10th-11th	UK	Journal paper
Lockwood and Mooney, 2018	292	10th-12th	Ireland	Conference paper

3.2 Which Tools are Used to Assess CT in Europe?

In this section we present a more in-depth analysis of the tools used to assess CT. Table 2 shows the different types of tools used in the 26 studies grouped by tool nature. Across the 26 articles, 18 unique forms of assessment were identified and grouped in five categories: test/tasks (Visual Blocks Creative Computing Test, CT Test, PCNT test, Bebras tasks, Code.org tasks, Scratch tasks, Alice tasks, Java tasks, Educational robotics tasks, CT activities, PhysGramming), questionnaires (CT Self-efficacy Scales, CT Ability Scale, CONT questionnaire and self-developed online questionnaires), observations, interviews, and analysis of products (manually, or automated with Dr Scratch). The most used tools are the CT Test (5 times), Bebras tasks (5 times), the Computational Thinking Ability Scale (4 times) Dr. Scratch (3 times), and Scratch tasks (3 times).

Table 2. Tool used by the analyzed papers.

Study	Name of Tool	Tool nature	Length of assessment
Fagerlund, Häkkinen, Vesisenaho, and Viiri, 2020	Scratch tasks	Analysis of products	-
Garneli and Chorianoopoulos, 2017	Dr. Scratch	Analysis of products	-
Förster, Förster, and Löwe, 2018	Dr. Scratch	Analysis of products	Automatic assessment over 24 program elements
Price and Price-Mohr, 2018	Java tasks	Interview and analysis of products	-
Allsop, 2019	Scratch and Alice tasks, observations and interviews	Observation, interview, analysis of products	-
Gillot, Gibbons, and Hidson, 2020	Scratch tasks and observations	Observations, interview	-
Kalliopi and Michail, 2019	PhysGramming	Tasks, observations and interview	-
Chiazzese, Arrigo, Chifari, Lonati and Tosto, 2018	Bebras task	Tasks	-
Chiazzese, Arrigo, Chifari, Lonati and Tosto, 2019	Bebras task	Tasks	10 items
del Olmo-Munoz, Cózar-Gutiérrez and González-Calero, 2020	Bebras task	Tasks	10 items
Lockwood and Mooney, 2018	Bebras task	Tasks	13 items
Segredo, Miranda, León and Santos, 2016	CT activities	Tasks	5 activities
Leifheit, Jabs, Ninaus, Moeller and Ostermann, 2018	Code.org tasks	Tasks	9 tasks
Bryndová and Mališů, 2020	Educational robotics tasks	Tasks	16 items

Saez-Lopez, Roman-Gonzalez, and Vazquez-Cano, 2016	Visual Blocks Creative Computing Test	Test	40 items
Korucu, Gencturk, and Gundogdu, 2017	CT Ability Scale	Questionnaire	22 items
Saritepeci and Durak, 2017	CT Ability Scale	Questionnaire	22 items
Yildiz Durak, 2018	CT Ability Scale	Questionnaire	22 items
Tonbuloglu and Tonbuloglu, 2019	CT Ability Scale and observations	Questionnaire and observations	22 items
Román-González, Moreno-León, and Robles, 2017	CT Test, Dr. Scratch and Bebras tasks	Questionnaire, analysis of product and tasks	28 items (CT Test)
Román-González, Perez-Gonzalez, and Jimenez-Fernandez, 2017	CT Test	Questionnaire	28 items
Román-González, Pérez-González, Moreno-León and Robles, 2018	CT Test	Questionnaire	28 items
Bati, Yetişir, Çalışkan, Güneş and Saçan, 2018	CT Test, observations and interviews	Questionnaire, observations and interviews	Depends on the module
Pérez-Marín, Hijón-Neira, and Bacelo, 2018	CT Test, PCNT Test, CONT questionnaire	Questionnaire	15 items (CONT), 28 items (CT Test), 14 Items (PCNT)
Kožuh, Krajnc, Hadjileontiadis, and Debevc, 2018	Online survey questionnaire	Questionnaire	13 items
Kukul and Karatas, 2019	CT Self-efficacy Scale	Questionnaire	18 items

Test/Tasks. The majority of the selected studies uses a test or a task to assess CT. Under this category we can find for example the Visual Blocks Creative Computing Test. This test has 40 items with a structured and progressive sequence. Students answer items related to sequences, loops, conditional statements, parallel execution, coordination, event handling, and keyboard input. Another frequently used test is the CT Test (Roman-Gonzalez et al., 2017), in which pupils have to solve 28 tasks. For example, a sequence of instructions is given to them and they have to decide how many times the sequence has to be executed in order to move a character from point A to point B on a grid. The CT test targets secondary school pupils. A similar test for primary school pupils is the PCNT Test (Pérez-Marín et al., 2018). Other often used tasks to assess CT skills are the Bebras tasks. The Bebras tasks are a large set of tasks used for the worldwide annual International Challenge on Informatics and Computational Thinking. The aim of the challenge is to increase pupils' engagement in informatics and to promote the development of computational thinking through the resolution of real-life and attractive problems (Chiazzese et al., 2018). Also the *code.org* tasks can be used to assess CT: the platform offers in fact different online courses for pupils which contain assessment tasks to be solved. In other cases, the researchers used tasks created with different programming languages for example Scratch, Alice or Java. In some cases, educational robotics tasks also can be used. Bryndová and Mališů (2020) for example use the robots Ozobot EVO and BIT. A last system that has been used to assess CT is PhysGramming (Kalliopi & Michail, 2019). PhysGramming is a digital environment that allows pupils to create their own games.

Questionnaires. Questionnaires related to CT are also often used to assess CT skills. Kukul et al. (2019) have for example developed the CT Self-Efficacy Scale. The original scale contained 51 items arranged as 5-point Likert scale (1 = "Completely Disagree" - 5 = "Fully Agree"). The scale was applied as a pilot to secondary school pupils and the items were reduced to 18 items. Example items on the scale are: "If there are sub-problems in the problem, I can manage the solution processes of these sub-problems"; "I can make connections between the current problem and previously encountered problems". Another questionnaire used is the CT Ability Scale developed by Korkmaz, Çakır, and Özden (2016). The scale was originally developed for university students and then was adapted for secondary school pupils. Also in this case, a five-point Likert scale is used. Examples of items in the test are "I believe that I can easily catch the relation between the figures"; "It is fun to try to solve the complex problems". Other questionnaires allowed open questions an example is the CONT questionnaire that measure knowledge of

programming concepts. An example of a question is "What do you think a program is? Can you give an example?" (Pérez-Marín et al., 2018).

Observations. In some studies, CT skills have been assessed by observing pupils solving different tasks. For example, Allsop (2019) collected data on CT skills by observing "the language children used for their 'self' explanations and group discussions, the gestures, the context of their relations with teacher, peers and technology in their classroom setting" (p.33).

Interviews. In order to assess CT, few studies also asked pupils to create a product (a coded story) and then interviewed them to let them explain their coded story (Price & Price-Mohr, 2018).

Analysis of products. Another method to assess CT skills is to analyse pupils' projects. An example of this are projects created in Scratch. The products can be analysed manually or automatically with Dr. Scratch for example. Dr. Scratch is an online analysis tool which can assess CT skills of a Scratch project based on the number of sprites, blocks, loops, and other concepts used in the project, and calculate a CT skills score. Dr Scratch however has some limitations as it cannot detect if the program is functioning as intended: a project with the appropriate blocks could get a high CT score, although it may lack functionality (Moreno-León & Robles, 2015).

3.3 Which Dimensions are Assessed?

The presented tools have been used to analyse different dimensions of CT. Table 3 shows an overview of the analysed dimensions according to used tools and study grouped by tool name. Analysed dimensions is composed of keywords taken from the papers where the tool was used. The analysed dimensions of a specific tool used in more than one paper could be different or have slightly different terminology among the papers, depending on the authors' research focus and on how they have used the instrument.

In terms of concepts and processes, we stayed with what the authors defined as a CT dimension in the reviewed papers. This could include programming constructs as well as non-programming terms and processes. We addressed this difference in Section 4 and Section 5 and we highlighted why this is still an issue to reach a common operational definition of CT.

Table 3. Tools used in the selected studies and dimensions analyzed by the tools.

Study	Name of Tool (or type if N.A.)	Analysed dimensions
Chiazzese, Arrigo, Chifari, Lonati and Tosto, 2018	Bebras tasks	Algorithmic thinking, Implementing simple algorithmic procedures, Logically analyzing data, Logically organizing data, Representing data through formal encoding
Lockwood and Mooney, 2018	Bebras tasks	Data ordering, Encoding, Gossip problem, If then else objects, Pattern matching attributes and variables, Stacks, Trees ciphering, Sorting
Del Olmo-Muñoz, Cózar-Gutiérrez, and González-Calero, 2020	Bebras tasks	Algorithmic thinking, Decomposition, Evaluation, Generalisation
Leifheit, Jabs, Ninaus, Moeller and Ostermann, 2018	Code.org tasks	Conditionals, Debugging, Events
Korucu, Gencturk, and Gundogdu, 2017	CT Ability Scale	Algorithmic thinking, Analytical Thinking, Collaboration, Creativity, Problem solving
Sartepeci and Durak, 2017	CT Ability Scale	Algorithmic thinking, Collaboration, Creativity, Critical thinking, Problem solving
Yildiz Durak, 2018	CT Ability Scale	Algorithmic thinking, Collaboration, Creativity, Critical thinking, Problem solving
Tonbuloglu and Tonbuloglu, 2019	CT Ability Scale	Algorithmic thinking, Collaboration, Creativity, Critical thinking, Problem solving
Segredo, Miranda, León and Santos, 2016	CT activities	Abstraction, Algorithmic thinking, Cognitive planning, Logical thinking

Kukul and Karatas, 2019	CT Self-efficacy Scale	Abstraction, Decomposition, Generalization, Reasoning
Román-González, Perez-Gonzalez, and Jimenez-Fernandez, 2017	CT Test	Computational concepts (sequences, loops, conditionals, operators), Computational practices (testing and debugging, reusing and remixing, abstracting and modularizing)
Román-González, Moreno-León, and Robles, 2017	CT Test, Dr Scratch, and Bebras tasks	Abstraction and problem decomposition, Data representation, Flow control, Logical thinking, Parallelism, Synchronization, User interactivity
Bati, Yetişir, Çalışkan, Güneş and Saçan, 2018	CT Test, observations, and interviews	Assessing different approaches/solutions to a problem, choosing effective computational tools, Creating abstractions, Developing modular computational solutions, Programming, Troubleshooting and debugging, Using problem solving strategies
Pérez-Marín, Hijón-Neira, and Bacelo, 2018	CT Test, PCNT Test, CONT questionnaire	Abstract and encapsulate, Incremental and iterative development, Mix and reuse, Test and Debugging
Román-González, Pérez-González, Moreno-León and Robles, 2018	CT Test	Computational concepts (sequences, loops, conditionals, operators), Computational practices (testing and debugging, reusing and remixing, abstracting and modularizing)
Garneli and Chorianopoulos, 2017	Dr Scratch	Data, Computational practices and perspectives, Conditionals, Events, Loops, Operators, Parallelism, Sequences
Förster, Förster, and Löwe, 2018	Dr Scratch	Abstraction and problem decomposition, Algorithmic notions of flow control, Data representation, Logical thinking, Parallelism, Synchronization, User interactivity
Bryndová and Mališů, 2020	Educational robotic tasks	Abstraction, Algorithmization, Decomposition, Evaluation, Generalization
Price and Price-Mohr, 2018	Java tasks	Abstraction, Algorithmic thinking, Decomposition
Kožuh, Krajnc, Hadjileontiadis, and Debevc, 2018	Online survey questionnaire	If-clause, Loops, Series of execute commands, Variables
Kalliopi and Michail, 2019	PhysGramming	Abstraction, Algorithmic thinking, Data analysis (identifying misconceptions, reconsider choices), Data collection, Data organization
Allsop, 2019	Scratch and Alice tasks, interviews, and observations	Abstractions, Conditionals, Events, Loops, Operators, Parallelism, Sequences, Variables
Fagerlund, Häkkinen, Vesisenaho, and Viiri, 2020	Scratch	Abstraction, Algorithms, Automation, Coordination, Creativity, Data, Logic, Modeling and design, Patterns, Problem decomposition
Gillott, Joyce-Gibbons, and Hidson, 2020	Scratch tasks, interviews, and observations	Abstraction, Algorithmic thinking, Computational concepts, Computational perspectives, Debugging/Testing, Decomposition, Evaluation, Formulate problems, Generalization/Reusing, Logical reasoning
Saez-Lopez, Roman-Gonzalez, and Vazquez-Cano, 2016	Visual Blocks Creative Computing Test	Conditional statements, Event handling, Experimentation, Iteration, Keyboard input, Sequence, Threads, User Interface Design

Table 4 shows the dimensions analysed in the selected papers ordered by number of appearances. The table doesn't take into account duplicates (that is, if a tool appeared in more than one study, the dimensions it analysed are counted only once). Whenever there are differences in the terminology used when referring to the same dimension, all the used terms are presented in the same row (e.g., abstract, abstraction). It is interesting to note that the tools

presented in the selected studies allow 59 different dimensions to be analysed. Only 21 dimensions however appear more than once, while all the others (38) are representative of a single tool. This gives an initial indication which dimensions are most frequently taken into account and thus associated with CT. If the dimensions tested by the tools are analysed according to school grades it can be noticed that the type of dimensions used do not differ based on students' grades.

Table 4. Dimensions analysed by the reviewed papers.

Dimension	Appearances	Dimension	Appearances
Abstract / Abstraction	8	Data collection	1
Algorithm / Algorithmic thinking / Algorithmization	8	Data ordering	1
Problem decomposition / Decomposition	7	Developing modular computational solutions	1
Conditionals / Conditional statements	6	Encapsulate	1
Generalization / Pattern recognition / Patterns	6	Encoding	1
Loops / Iterations	5	Execute commands	1
Events / Event handling	4	Experimentation	1
Logic / Logical thinking / Logical reasoning	4	Flow control	1
Sequences	4	Formulate problems	1
Creativity	3	Gossip problem	1
Debugging	3	Incremental development	1
Evaluation	3	Iterative development	1
Operators	3	Keyboard input	1
Parallelism	3	Mix / Remixing	1
Variables	3	Modeling and design	1
Analyze data / Data analysis	2	Programming	1
Computational practices and perspectives	2	Reasoning	1
Data	2	Reuse	1
Data representation	2	Simple functions	1
Organize data / Data organization	2	Sorting	1
Problem solving	2	Stacks	1
Analytical / critical thinking	1	Synchronization	1
Assessing different approaches/solutions to a problem	1	Test	1
Automation	1	Threads	1
Choosing Effective Computational Tools	1	Trees	1
Ciphering	1	Troubleshooting	1
Cognitive planning	1	User interactivity	1
Collaboration	1	User interface / User interface design	1
Computational Concepts	1	While conditional	1
Coordination	1		

The dimensions mentioned in Table 4 can be further divided into dimensions purely related to programming and informatics, and dimensions related in a broader sense to CT. To this respect, we decided to select the dimensions which can be associated with the definition of CT given Wing and Shute's seminal works and present them in Table 5. The table shows 19 dimensions in total related to Wing and Shute's works. Interestingly enough, more than half of the dimensions (11) are mentioned in more than one tool.

Table 5. Dimensions analysed that can be related to CT as seen in Wing and Shute's seminal work.

CT dimension (Wing, 2006 and Shute et al., 2017)	Appearances
Abstract / Abstraction	8
Algorithm / Algorithmic thinking / Algorithmization	8
Problem decomposition / Decomposition	7
Generalization / Pattern recognition / Patterns	6
Logic / Logical thinking / Logical reasoning	4
Creativity	3
Debugging	3
Evaluation	3
Analyze data / Data analysis	2
Organize data / Data organization	2
Problem solving	2
Analytical / Critical thinking	1
Assessing different approaches/solutions to a problem	1
Cognitive planning	1
Collaboration	1
Data collection	1
Formulate problems	1
Reasoning	1
Test	1

The remaining dimensions fall closer to programming skills rather than to the definition of CT we decided to focus on. For example, among these dimensions (Table 6) we find conditional statements, iterations, events, sequences, variables, execution of commands, data representation, and operators. While it may be argued which dimensions are purely related to programming rather than being adaptable in a broader sense to CT, it is clear that dimensions such as conditionals, loops, or programming are concepts that are mostly related to programming and informatics, rather than CT. To this end, debugging can be seen as the act of fixing an error in a mental algorithm/procedure (Shute et al., 2017), and thus can be seen as part of CT given a much inclusive definition.

Table 6. Dimensions analyzed that can be seen as purely related to programming.

Programming dimension	Appearances	Programming dimension	Appearances
Conditionals / Conditional statements	6	Experimentation	1
Loops / Iterations	5	Flow control	1
Events / Event handling	4	Gossip problem	1
Sequences	4	Incremental development	1
Operators	3	Iterative development	1
Parallelism	3	Keyboard input	1
Variables	3	Mix / Remixing	1
Computational practices and perspectives	2	Modeling and design	1
Data	2	Programming	1
Data representation	2	Reuse	1
Automation	1	Simple functions	1
Choosing effective computational tools	1	Sorting	1
Ciphering	1	Stacks	1
Computational concepts	1	Synchronization	1
Coordination	1	Threads	1
Data ordering	1	Trees	1
Developing modular computational solutions	1	Troubleshooting	1
Encapsulate	1	User interactivity	1

Encoding	1	User interface / User interface design	1
Execute commands	1	While conditional	1

4. Discussion

This paper reports the assessment tools and the assessed dimensions of 26 European studies conducted between 2016 and 2020. The results conform to the existing literature that shows a variety of CT assessments methodologies (e.g., Çoban, & Korkmaz, 2021) and indicates different CT assessment tools exist that can be categorized in five groups: questionnaires, tests/tasks, observations, interviews and analysis of products. The first two categories (questionnaires and test/tasks) were the two most common. Most studies use a single form of assessment (either questionnaire, test, interview, etc...) and often limit themselves to assess if students can recognize and recall knowledge out of context. These forms therefore do not allow assessment of the competences and in particular CT skills that have a multifaceted nature. This can also be noticed analysing the dimensions of CT that the reviewed tools allow to assess. Wing (2006) defined CT as a fundamental skill, a definition further expanded by Shute et al. (2017), clearly decoupling it from basic computer science. Nonetheless most of the dimensions that the review tools assess are related to programming skills, rather than effectively measuring the ability to solve problems through CT which goes beyond computer science. Our review focused on European K-12 education confirms some of the results found in other reviews carried out by researchers in non-European countries: a breadth of methods employed to assess CT (Cutumisu et al., 2019) and the majority of them analysing concepts directly related to algorithms and programming (Tang et al. 2020). The need for tools that allow to assess all facets of CT has been discussed already in other studies such as highlighted in Basu, Rutstein, Xu, Wang and Shear, 2021. This is also related to the different operational definitions of CT making it difficult to agree on the dimensions of CT and to develop a common and reliable assessment tool (Adams et al., 2019). In fact, in the 26 studies selected, as many as 59 different dimensions are associated with CT, however only 21 dimensions appear more than once, while the remaining 38 are representative of a single tool. The analysis of the tools also shows that they do not refer to a shared competence model of CT differentiated by age. The different studies assess CT in pupils in different grades, however it is not clear what competences pupils should reach at which age since the same dimensions and tools are used for pupils of different ages. The challenges in the development of assessment methods and frameworks that include all facets of CT is already mentioned by different authors (e.g. Brennan & Resnick, 2012; Denner, Werner, & Ortiz, 2012; Denning, 2017; Fronza, El Ioini, & Corral, 2017; Grover et al., 2017; Grover, Pea, & Cooper, 2015; Tikva & Tambouris, 2021; Zhong, Wang, Chen, & Li, 2016) are still present and should drive the need for future research in the field.

5. Conclusion

In this paper we focused on answering two main questions: “Which tools are used to assess CT in Europe?” and “Which dimensions of CT are assessed?”. In her 2006 paper, Wing describes CT as “[...] solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science”. The dimensions shown in Table 6 can be associated with the fundamental concepts of computer science, yet on an abstract level they can also be used to describe human behavior when solving a problem (Voskoglou & Buckley, 2012). We argue that the presented tools rely too much on computer science concepts rather than focusing on problem-solving skills in educational contexts (Rahman, 2019).

Based on these reflections we can formulate following issues that are present in the assessment of CT in European K-12 education and should drive future research in the field:

- A common operational definition of CT is still absent.
- A competence model that indicates which competences students should reach in CT at which age is absent.
- Still missing is a tool that takes into consideration all the different dimensions of CT and does not focus only on a few of them or just on programming skills.

In order to advance in this research field, we believe it would be important to define a competence model of CT according to the pupil’s age, i.e., a model of the pupil’s skills, knowledge and possible behaviours in a given context. Based on this model, assessment rubrics (Popham, 1997) could be defined.

An assessment rubric consists, in general, in a qualitative description of possible observable behaviours that can be observed during the accomplishment of a task, corresponding to different performance levels with respect to the

components of the competence being assessed. The different performance levels could be expressed in different ways (Dawson, 2017), for example, they could be defined through the amount of assistance needed during the resolution of the task. In this case, during the activity students can have access to different aid; the more aid they need, in form of hints, suggestions, or supplementary tools or instruments to produce an acceptable solution to the given problem, the less competent they are. With the help of these rubrics, students could potentially be assessed by being given tasks to be solved that include all dimensions of a shared operational definition of CT.

Statements on Ethics & Conflicts-References

This research did not receive any specific grant from funding agencies. Authors report no conflict of interest. All authors have equally participated in this article.

References

- Académie des Sciences (2013). *L'enseignement de l'informatique En France - Il Est Urgent de Ne Plus Attendre*. Paris: Académie des Sciences.
- Adams, C., Cutumisu, M. & Chang, L. (2019). Measuring K-12 Computational Thinking Concepts, Practices and Perspectives: An Examination of Current CT Assessments. In *Proceedings of Society for Information Technology & Teacher Education International Conference 2019*, (pp. 275–85). Las Vegas, NV, United States: Association for the Advancement of Computing in Education (AACE).
<https://www.learntechlib.org/p/207654>.
- *Allsop, Y. (2019). Assessing Computational Thinking Process Using a Multiple Evaluation Approach. *International Journal of Child-Computer Interaction* 19, 30–55. <https://doi.org/10.1016/j.ijcci.2018.10.004>.
- Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, 75(Part B), 661–670. doi:10.1016/j.robot.2015.10.008
- Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, 38(6), 20–23. <https://files.eric.ed.gov/fulltext/EJ918910.pdf>
- Basu, S., Rutstein, D.W., Xu, Y., Wang, H. & Shear, L. (2021). A principled approach to designing computational thinking concepts and practices assessments for upper elementary grades. *Computer Science Education*, 31(2), 169-198. DOI:10.1080/08993408.2020.1866939
- *Bati, K., Yetişir, M. I., Çalışkan, I. & Güneş, G. (2018). Teaching the Concept of Time: A Steam-Based Program on Computational Thinking in Science Education. *Cogent Education*, 5(1), 1–16.
<https://www.tandfonline.com/doi/abs/10.1080/2331186X.2018.1507306>.
- Berland, M., & Wilensky, U. (2015). Comparing Virtual and Physical Robotics Environments for Supporting Complex Systems and Computational Thinking. *Journal of Science Education and Technology*, 24, 628–47.
<https://doi.org/10.1007/s10956-015-9552-x>.
- Bocconi, S., Chiocciariello, A., & Earp, J. (2018). The nordic approach to introducing computational thinking and programming in compulsory education. *Report prepared for the Nordic@BETT2018Steering Group*. Retrieved from <http://edudoc.ch/record/131448doi:10.17471/54007>
- Bocconi, S., Chiocciariello, A., Dettori, G. Ferrari, A., Engelhardt, K., Kampylis, P., & Punie, Y. (2016). *Developing Computational Thinking in Compulsory Education – Implications for Policy and Practice*. JRC Science for Policy Report. Luxembourg: Publications Office of the European Union. <https://doi.org/10.2791/792158>.
- Brennan, K., & Resnick, M. (2012). *New Frameworks for Studying and Assessing the Development of Computational Thinking*. Vancouver, Canada. Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf.
- *Bryndová, L., & Mališů, P. (2020). Assessing the Current Level of the Computational Thinking Within the Primary and Lower Secondary School Students Using Educational Robotics Tasks. In *2020 the 4th International Conference on Education and Multimedia Technology*, (pp. 239–43). ICEMT 2020. New York, NY, USA: Association for Computing Machinery. <https://doi.org/10.1145/3416797.3416819>.

- Calmet, C., Hirtzig, M., & Wilgenbus, D. (2016). *1, 2, 3... Codez !: Enseigner l'informatique à l'école Et Au Collège (Cycles 1, 2 Et 3)*. Éducation. Paris: Editions Le Pommier.
- *Chiazzese, G., Arrigo, M., Chifari, A., Lonati, V. & Tosto, C. (2018). Exploring the Effect of a Robotics Laboratory on Computational Thinking Skills in Primary School Children Using the Bebras Tasks. In *Proceedings of the 6th International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM 2018)*, (pp. 27–30). <https://doi.org/10.1145/3284179.3284186>.
- *Chiazzese, G., Arrigo, M., Chifari, A., Lonati, V. & Tosto, C. (2019). Educational Robotics in Primary School: Measuring the Development of Computational Thinking Skills with the Bebras Tasks. *Informatics*, 6, 43. <https://doi.org/10.3390/informatics6040043>.
- Chiocciariello, A., & Olimpo, G. (2017). Editorial. *Italian Journal of Educational Technology*, 25, 3–6. <https://doi.org/10.17471/2499-4324/986>.
- Çoban, E., & Korkmaz, Ö. (2021). An alternative approach for measuring computational thinking: Performance-based platform. *Thinking Skills and Creativity*, 42, 100929. <https://doi.org/10.1016/J.TSC.2021.100929>
- Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational thinking. A guide for teachers*. Computing At School. Retrieved from https://eprints.soton.ac.uk/424545/1/150818_Computational_Thinking_1_.pdf
- CSTA and ISTE. (2011). *Operational Definition of Computational Thinking for k–12 Education*. 2011. <https://id.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf>.
- Cutumisu, M., Adams, C., & Chang L. (2019). A Scoping Review of Empirical Research on Recent Computational Thinking Assessments. *Journal of Science Education and Technology*, 28, 651–76. <https://doi.org/10.1007/s10956-019-09799-3>.
- Da Cruz Alves, N., Gresse Von Wangenheim, C. & Hauck, J.C.R. (2019). Approaches to Assess Computational Thinking Competences Based on Code Analysis in K-12 Education: A Systematic Mapping Study. *Informatics in Education*, 18(1), 17–39. <https://doi.org/10.15388/infedu.2019.02>.
- Dawson, P. (2017). Assessment Rubrics: Towards Clearer and More Replicable Design, Research and Practice. *Assessment & Evaluation in Higher Education*, 42(3), 347–60. <https://doi.org/10.1080/02602938.2015.1111294>.
- *del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J.A. (2020). Computational Thinking Through Unplugged Activities in Early Years of Primary Education. *Computers & Education*, 150, 103832. <https://doi.org/https://doi.org/10.1016/j.compedu.2020.103832>.
- Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, 58(1), 240–249. <https://doi.org/10.1016/j.compedu.2011.08.006>
- Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, 60(6), 33–39. <https://doi.org/10.1145/2998438>
- European Commission. (2016). *A New Skills Agenda for Europe. Working Together to Strengthen Human Capital, Employability and Competitiveness*. Brussels: European Commission. <https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX%3A52016DC0381>.
- *Fagerlund, J., Häkkinen, P. Vesisenaho, M., & Viiri, J. (2020). Assessing 4th Grade Students Computational Thinking Through Scratch Programming Projects. *Informatics in Education*, 611–40. <https://doi.org/10.15388/infedu.2020.27>.
- *Förster, E.-C., Förster, K.-T. & Löwe, T. (2018). Teaching Programming Skills in Primary School Mathematics Classes: An Evaluation Using Game Programming. In *2018 IEEE Global Engineering Education Conference (EDUCON)*, (pp. 1504–13). <https://doi.org/10.1109/EDUCON.2018.8363411>.

- Fronza, I., El Ioini, N., & Corral, L. (2017). Teaching computational thinking using agile software engineering methods: A framework for middle schools. *ACM Transactions on Computing Education, 17*.
<https://doi.org/10.1145/3055258>
- *Garneli, V., & Chorianopoulos, K. (2018). Programming Video Games and Simulations in Science Education: Exploring Computational Thinking Through Code Analysis. *Interactive Learning Environments, 26*(3), 386–401. <https://doi.org/10.1080/10494820.2017.1337036>.
- *Gillott, L., Joyce-Gibbons, A., & Hidson, E. (2020). Exploring and Comparing Computational Thinking Skills in Students Who Take GCSE Computer Science and Those Who Do Not. *International Journal of Computer Science Education in Schools, 3*, 3–22. <https://doi.org/10.21585/ijcses.v3i4.77>.
- Grover, S., Basu, S., Bienkowski, M., Eagle, M., Diana, N., & Stamper, J. (2017). A framework for using hypothesis-driven approaches to support data-driven learning analytics in measuring computational thinking in block-based programming environments. *ACM Transactions on Computing Education, 17*.
<https://doi.org/10.1145/3105910>
- Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher, 42*(1), 38–43. <https://doi.org/10.3102/0013189X12463051>.
- Grover, S., Pea, R., & Cooper, S. (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>
- Israel, M., Pearson, J.N., Tapia, T., Wherfel, Q.M., & Reese, G. (2015). Supporting All Learners in Schoolwide Computational Thinking: A Crosscase Qualitative Analysis. *Computers & Education, 82*, 263–79.
<https://doi.org/10.1016/j.compedu.2014.11.022>.
- *Kalliopi, K., & Michail, K. (2019). Assessing Computational Thinking Skills at First Stages of Schooling. In *Proceedings of the 2019 3rd International Conference on Education and e-Learning*, (pp. 135–39). ICEEL 2019. New York, NY, USA: Association for Computing Machinery.
<https://doi.org/10.1145/3371647.3371651>.
- Korkmaz, Ö., Çakır, R. & Özden, Y.M. (2016). Computational Thinking Levels Scale (CTLS) Adaptation for Secondary School Level. *Gazi Journal of Educational Science, 1*(2), 143–62.
- *Korucu, A. T., Gençturk, A. T. & Gundogdu, M. M. (2017). Examination of the Computational Thinking Skills of Students. *Journal of Learning and Teaching in Digital Age, 2*(1), 11–19. <https://eric.ed.gov/?id=ED572684>.
- *Kožuš, I., Krajnc, R., Hadjileontiadis, L. J. & Debevc, M. (2018). Assessment of Problem Solving Ability in Novice Programmers. *PLoS ONE, 13*(9). <https://doi.org/10.1371/journal.pone.0201919>.
- *Kukul, V., & Karatas, S. (2019). Computational Thinking Self-Efficacy Scale: Development, Validity and Reliability. *Informatics in Education, 18*(1), 151–64. <https://doi.org/10.15388/infedu.2019.07>.
- *Leifheit, L., Jabs, J., Ninaus, M., Moeller, K. & Ostermann, K. (2018). Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School. In *European Conference on Game-Based Learning (ECGBL)* (pp. 344–53). <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85058927915&partnerID=40&md5=3c9851a83ff1c00f11bc838a9ffa8587>.
- *Lockwood, J., & Mooney, A. (2018). Developing a Computational Thinking Test Using Bebras Problems. In *Joint Proceedings of the CC-TEL 2018 and TACKLE 2018 Workshops, co-located with 13th European Conference on Technology Enhanced Learning (EC-TEL 2018)*. <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85053103913&partnerID=40&md5=8e374245636670e245ebdafc427904dc>.
- 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>.

- Moreno-León, J., & Robles, G. (2015). Analyze Your Scratch Projects with Dr. Scratch and Assess Your Computational Thinking Skills. In *Proceedings of the 7th international Scratch conference (Scratch2015AMS)*, (pp. 12–15). Amsterdam, Netherlands.
- *Pérez-Marín, D., Hijón-Neira, R. & Bacelo, A. (2018). Can Computational Thinking Be Improved by Using a Methodology Based on Metaphors and Scratch to Teach Computer Programming to Children? *Computers in Human Behavior*, 105. <https://www.sciencedirect.com/science/article/pii/S0747563218306137>.
- Petticrew, M., & Roberts, H. (2006). *Systematic Reviews in the Social Sciences: A Practical Guide*. First. London: Blackwell Pub. <https://doi.org/10.5860/choice.43-5664>.
- Popham, W. J. (1997). What's Wrong—and What's Right—with Rubrics. *Educational Leadership*, 2nd series, 55(2), 72–75.
- *Price, C. B., & Price-Mohr, R. M. (2018). An Evaluation of Primary School Children Coding Using a Text-Based Language (Java). *Computers in the Schools*. 35(4), 284–301. <https://www.tandfonline.com/doi/abs/10.1080/07380569.2018.1531613>.
- Rahman, M. (2019). 21st Century Skill "Problem Solving": Defining the Concept. *Asian Journal of Interdisciplinary Research*, 2(1), 64–74. <https://doi.org/10.34256/ajir1917>.
- *Roman-Gonzalez, M., Perez-Gonzalez, J.C. & Jimenez-Fernandez, C. (2017). Which Cognitive Abilities Underlie Computational Thinking? Criterion Validity of the Computational Thinking Test. *Computers in Human Behavior*, 72, 678–91. <https://doi.org/10.1016/j.chb.2016.08.047>.
- *Román-González, M., Moreno-León, J. & Robles, G. (2017). Complementary Tools for Computational Thinking Assessment. In *Proceedings of the International Conference on Computational Thinking Education*, (pp. 154–59). <http://www.eduhk.hk/cte2017/doc/CTE2017Proceedings.pdf>.
- *Román-González, M., Pérez-González, J.-C. Moreno-León, J. & Robles G. (2018). Can Computational Talent Be Detected? Predictive Validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction*, 18, 47–58. <https://doi.org/10.1016/j.ijcci.2018.06.004>.
- Royal Society (2012). *Shut Down or Restart? The way forward for Computing in UK Schools*. London: The Royal Society.
- *Saez-Lopez, J.M., 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, 129–41. <https://doi.org/10.1016/j.compedu.2016.03.003>.
- *Sartepeci, M., & Durak, H. (2017). Analyzing the Effect of Block and Robotic Coding Activities on Computational Thinking in Programming Education. In I. Koleva & G. Duman (Eds.), *Educational research and practice*, (pp. 464–73). Sofia: St. Kliment Ohridski University Press.
- Schweizerische Eidgenossenschaft. (2017). *Bericht Über Die Zentralen Rahmenbedingungen Für Die Digitale Wirtschaft*. Schweizerische Eidgenossenschaft.
- *Segredo, E., Miranda, G. León, C. & Santos, A. (2016). Developing Computational Thinking Abilities Instead of Digital Literacy in Primary and Secondary School Students. In *Smart Education and e-Learning*, (pp. 235–45). https://link.springer.com/chapter/10.1007/978-3-319-39690-3_21.
- Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying Computational Thinking. *Educational Research Review*, 22, 142–58. <https://doi.org/10.1016/j.edurev.2017.09.003>.
- Tang, X., Yin, Y., Lin, Q. Hadad, R. & Zhai, X. (2020). Assessing Computational Thinking: A Systematic Review of Empirical Studies. *Computers & Education*, 148. <https://doi.org/https://doi.org/10.1016/j.compedu.2019.103798>.
- *Tonbuloğlu, B., & Tonbuloğlu, I. (2019). The Effect of Unplugged Coding Activities on Computational Thinking Skills of Middle School Students. *Informatics in Education*, 18, 403–26. <https://doi.org/10.15388/infedu.2019.19>.

- Tikva, C., & Tambouris, E. (2021). Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature Review. *Computers & Education*, 162, 104083. <https://doi.org/10.1016/j.compedu.2020.104083>
- Voskoglou, M., & Buckley, S. (2012). Problem Solving and Computational Thinking in a Learning Environment. *Egyptian Computer Science Journal*, 36(4), 28–46.
- Wing, J. (2006). Computational Thinking. *Communications of the ACM*, 49, 33–35. <https://doi.org/10.1145/1118178.1118215>.
- World Economic Forum. (2016). *New Vision for Education: Fostering Social and Emotional Learning Through Technology*. World Economic Forum.
- *Yildiz Durak, H. (2018). The Effects of Using Different Tools in Programming Teaching of Secondary School Students on Engagement, Computational Thinking and Reflective Thinking Skills for Problem Solving. *Tech Know Learn*, 25, 179–95. <https://doi.org/10.1007/s10758-018-9391-y>.
- Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research*, 53. <https://doi.org/10.1177/0735633115608444>

Copyrights

Copyright for this article is retained by the author(s), with first publication rights granted to the journal.

This is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/3.0/>).

Summer Programming Camps – Exploring Project-Based Informal CS Education in a Rural Community

Carla DE LIRA¹
Rachel WONG²
Olufunso OJE¹
Gabriel NKETAH¹
Olusola ADESOPE¹
Alireza GHODS¹

¹ Washington State University, Pullman, WA, United States of America

² University of Tennessee, Knoxville, TN, United States of America

DOI: 10.21585/ijcses.v5i4.145

Abstract

Current research has not fully explored how summer programming camps can help students increase motivation and interest to pursue computing career, and their programming knowledge. Informal CS education through summer programming camps provides K-12 students the opportunity to learn how to code through fun and interactive activities outside of their typical classroom experiences. In this study, we examined the effectiveness of a weeklong summer programming camp for promoting students' motivation and interest in programming, and their programming knowledge. Participants were 19 middle school students from rural Washington. Students participated in a project-based learning approach through game development in Python. Using a within-subjects design, we analyzed students' pre and post motivation and knowledge assessment scores. Results from the analysis indicated a significant improvement in post-test programming knowledge scores ($d = 0.93$). The findings also indicated that students were able to achieve basic abstraction and algorithmic thinking but not code analysis and debugging skills. On their motivation to pursue computing careers, the results did not show any difference before and after the camp due to their prior existing interest in attending the camp.

Keywords: Computer science education, pre-college programs, STEM, programming camps, K-12 education

1. Introduction

The number of individuals graduating with a Science, Technology, Engineering, and Mathematics (STEM) major remains low despite the increase in STEM jobs in the United States (Bureau of Labor Statistics, 2019; National Science Board, 2016; Xianglei & Weko, 2009). Two plausible reasons include the lack of interest in pursuing STEM-related courses, and the lack of early opportunities and exposure to STEM (Tai et al., 2016.). This issue is further exacerbated in computing where there is rapid demand for talent in the tech industry in the United States, but not enough of graduates in computing-related degrees (Zweben & Bizot, 2020). According to the Bureau of Labor Statistics, computing occupations, such as software developers and computer programmers, are projected to grow 13% between 2020 and 2030 (Bureau of Labor Statistics, 2019). It is then important to ensure that we spark interest in computing early among K-12 students in hopes that they become the next generation to maintain and develop our technological infrastructures. Early exposure to computing opportunities, especially for girls, is important as it may increase a child's interest in computing, improve their perceptions, and eliminate gender stereotypes (Bagiati et al., 2010; Tai et al., 2016) In fact, it has been shown that early exposure to computing prior to high school yields a higher chance that their interest in computing maintains into higher education (Christensen et al., 2014; Hirsch et al., 2017; Taub et al., 2012) Out-of-school activities or informal learning experiences through STEM camps is one

potential way to provide early exposure to STEM (Bell et al., 2009, p. 20; Cabrera et al., 2021; Mohr-Schroeder et al., 2014), especially computing (DeWitt et al., 2017b; Frye et al., 2016; Master et al., 2017).

Informal learning environments go beyond the traditional classroom and provide a casual learning experience for students (Roberts et al., 2018). Within computer science (CS) education, these learning opportunities commonly introduce computing concepts in a hands-on approach or relatable manner that may be beneficial for supporting formal computer science (CS) education in the future (DeWitt et al., 2017a; Franklin et al., 2013; Lakanen & Kärkkäinen, 2019; Xianglei & Weko, 2009). Currently, many school districts in the United States still do not incorporate programming as part of their STEM curriculum, due to the lack of resources, such as finding teachers who can teach it (Warner et al., 2019). Informal CS learning opportunities may be the only time students in a particular region would be able to engage in programming outside of the classroom and possibly prior to college (Warner et al., 2019). Informal STEM learning opportunities, such as programming camps, are often offered during the summer after the school year (Frye et al., 2016; Roberts et al., 2018; Webb & Rosson, 2011). Since knowledge loss typically occurs over summer breaks due to the lack of access to learning opportunities (McCombs et al., 2011), free informal STEM opportunities, like programming camps, are particularly important for students from low socioeconomic backgrounds who otherwise may not have access (Lusa Krug et al., 2021). Since programming camps can use various STEM concepts as a context for learning how to code (LePendur et al., 2020; Nite et al., 2020), these programs can provide the opportunity to engage in STEM topics covered during the school year while also introducing coding concepts.

Given the positive effects of summer programming camps on students' interest in computing, the research team developed a free summer camp for middle school students in rural Eastern Washington. A one-time programming camp was previously offered in the region, albeit only for middle-school-aged girls. Due to the lack of programming resources in the area, this camp was designed for middle school students. The study has two broad aims.

First, we are interested in examining the impact of participation in a week-long summer programming camp on students' motivation in programming and interest in pursuing a programming-related career. Research suggests that even a short week-long exposure to STEM activities may increase students' interest in STEM and positively influence their perceptions about STEM (National Science Board, 2016; Xianglei & Weko, 2009). We have also seen this reflected in longitudinal studies. Girls who were exposed to computing at a programming camp maintained an interest in programming over time (Outlay et al., 2017).

We are also interested in examining whether participation in the week-long camp is sufficient exposure to increase students' knowledge of programming and their ability to apply programming concepts. Franklin et al.'s study found that exposing students to two weeks of programming was sufficient for imparting computer science knowledge (Franklin et al., 2013). Programming provides the opportunity to exercise several computational thinking skills, such as understanding abstraction, problem formulation, and debugging for K-12 students (Lye & Koh, 2014). Thus, we are also interested in assessing students' computational thinking (CT) skills based on their programming knowledge performance. Despite the little research on learning to code through informal learning environments (i.e., programming camps), preliminary research indicates that informal learning experiences are effective in teaching code to students (Akcaoglu, 2014; Denner et al., 2012; Wang & Frye, 2019; Zamin et al., 2018). It is less clear as to how informal learning experiences in computing are effective in teaching computational thinking skills, especially since there is still ongoing discussion among scholars as to what CT comprises and ongoing efforts to measure CT skills (Shute et al., 2017; Werner et al., 2012)

Second, we are interested in the effectiveness of a hands-on project-based approach in helping students learn and retain programming concepts. In this approach, key concepts are interwoven into each step of the project that students are required to work on. Essentially, students learn and apply those key concepts simultaneously. To address these two broad aims, this study seeks to answer the following research questions:

RQ 1) How does participation in project-based learning influence students' motivation and interest before and after a short informal programming camp?

RQ 2) How does project-based learning influence students' programming knowledge before and after a short informal programming camp?

2. Related Work

Informal learning experiences are frequently offered outside of the classroom and structured curricula (Franklin et al., 2013; Roberts et al., 2018). Examples of informal learning environments include after-school programs, museum/field trips, and summer camps (Hofstein & Rosenfeld, 1996). In such environments, instructors and organizers are typically concerned with engaged participation, affective outcomes, and developing interest among students with loose learning objectives set for the duration of the informal learning opportunity (Hofstein & Rosenfeld, 1996; Stewart & Jordan, 2017).

Such opportunities are valuable for a couple of reasons. The emphasis on engaging participants and developing interest is especially important for female students who tend to lose interest in STEM while in middle school and through post-graduate education (Bagiati et al., 2010; Master et al., 2017). In addition, without sufficient exposure to STEM opportunities, students may develop a negative attitude towards STEM (Weinberg et al., 2011). Existing studies provide insights on the positive impact informal STEM opportunities have on students in future college major choices and interest in a STEM-related field (Miller et al., 2018; Weinberg et al., 2011).

In K-12 computer science education, there has been a gradual increase in recent years in summer programming camps as a popular form of an informal learning opportunity to stimulate interest in pursuing computer science (Bell et al., 2009; Bureau of Labor Statistics, 2019). These programming camps offer students the opportunity to delve into computing concepts that are largely not covered in many K-12 school curricula in the nation, especially in elementary and middle schools (Fields et al., 2015; Frye et al., 2016). The likelihood of a K-12 school curriculum that covers computing concepts becomes less in rural communities (Code Advocacy Coalition, 2018). For students in these underserved areas, a programming camp provides a learning opportunity in STEM that may be fun and engaging through an informal learning environment (Roberts et al., 2018).

2.1 Structure of Programming Camps for Middle School Students

One of the aims of programming camps is to provide students with an opportunity where they can learn problem-solving skills, have fun with programming tasks, and interact with their peers with similar interests (Adams, 2010). These camps cater to a range of students from elementary school (Chaudhary et al., 2016) to high school (Al-Bow et al., 2009). However, there has been a focus to provide programming opportunities particularly to middle school students (DeWitt et al., 2017b). Choices made in middle school can impact future education and career pursuits (Al-Bow et al., 2009; Wang et al., 2019). A major predictor of a student pursuing a STEM career upon graduating high school is their interest at the start of high school (Lakanen & Kärkkäinen, 2019; McCombs et al., 2011). Since interest in STEM careers may decline as a student matures (Ayar & Yalvac, 2016), it is crucial to spark interest in STEM in middle school students before they start high school (Hofstein & Rosenfeld, 1996; Xianglei & Weko, 2009).

Programming camps for middle school students are often in the form of hands-on workshops that utilize block programming languages, such as Scratch, or text-based languages, such as Python (Bryant et al., 2019). Such programs provide guidance in completing coding activities (Austin & Pinkard, 2008; Bagiati et al., 2010; Bell et al., 2009; Stewart & Jordan, 2017; Wang et al., 2019; Xianglei & Weko, 2009). Such camps have been found to be effective in generating interest in computer science and teaching students of varying backgrounds how to code (Maiorca et al., 2021; Weinberg et al., 2011).

Interestingly, although programming camps generate interest in computer science, little research has been conducted to examine how well these camps promote the acquisition and retention of students' programming knowledge. More specifically, there is a lack of research on the effective teaching methods in these informal learning environments. Thus, this study seeks to explore whether a project-based programming camp is able to foster learning of challenging programming concepts.

2.2 Project-based Programming Camps

In K-12 computer science education, there have been some efforts to discuss how to support students' growth in programming knowledge through project-based learning in informal learning environments, such as programming camps (Fields et al., 2015). Project-based learning is one of the most common teaching approaches in introducing K-12 students to STEM fields ("2018 NSSME+," 2018; Adams, 2010; Austin & Pinkard, 2008; Burack et al., 2018; DeWitt et al., 2017; Jones, 2019). This approach allows students to apply taught concepts to real-world experience through a project (Hugerat, 2016; Webb & Rosson, 2011). Project-based learning differs from traditional learning in that the project plays the main role in the curriculum. Students learn about concepts as they progress in their

project, which is often student-driven with some guidance from instructors/organizers. Project-based learning in STEM is also an effective way to promote K-12 students' STEM career interest (Al-Bow et al., 2009; DeWitt et al., 2017b). However, the effectiveness of project-based learning in gaining skills to prosper in STEM is largely unexplored within informal learning environments. As concepts in a project-based learning approach are introduced as students need them, it is unclear whether such concepts are retained at the end of their learning experience.

2.3 Project-based Programming Camps for Increasing Motivation & Interest

Project-based programming camps are typically organized to provide programming knowledge for middle school students to start working on their projects by the first or second day. Webb and Rosson held a week-long programming camp for middle school girls using Alice, a visual block programming environment, to gradually introduce programming concepts that they would need to create their individual 3D story (Webb & Rosson, 2011). At the end of the camp, they found that students were more interested in pursuing computer programming. In a shorter two-day programming camp, this method of gradually introducing just enough programming concepts to middle school students was also effective in promoting interest in computing careers (Outlay et al., 2017).

Another characteristic of project-based programming camps is the ability for students to share their completed projects at the end of the camp to instructors, friends, and even family (Bryant et al., 2019). In other camps, students have also created research posters to showcase their projects (Wang et al., 2019). Incorporating a project presentation component in a project-based programming camp might enhance students' sense of accomplishment by the end of the program (Sadler et al., 2018; Weinberg et al., 2011).

In general, project-based programming camps have been found to be very effective in generating middle school students' interest and motivation in computing careers. By providing as-needed information and concepts so students can complete their projects helps to build their confidence from the very beginning. The presentation component also allows them to share their success with others (Adams, 2010; National Science Board, 2016; Stewart & Jordan, 2017).

2.4 Project-based Programming Camps for Increasing Programming Knowledge

The desired outcomes for middle school students attending programming camps are an increased interest in programming careers, increased programming knowledge, and enjoyment in completing programming activities. Research highlights three different ways instructors can assess participants' knowledge. Ericson and McKlin utilized a 10-item multiple choice pre and post survey to assess middle and high school participants' programming knowledge. Results showed significant increases from pretest to posttest across different programming concepts, such as loops, variables, conditional statements etc., (Ericson & McKlin, 2012). In another study, students were asked to rate how much they knew about programming on a scale of 0 (nothing) to 5 (expert) after the camp. Seventy-three percent reported an increase in programming knowledge while 27% reported no change (Mohr-Schroeder et al., 2014). Unlike Ericson and McKlin, Franklin et al. analyzed participants' programming projects on Scratch to assess whether students acquired programming concepts (Franklin et al., 2013). This assessment allowed researchers to conclude that at the end of their two-weeklong camp, students successfully mastered event-driven programming, message passing, state initialization, and say/sound synchronization (Franklin et al., 2013). Interestingly, less attention has been paid to the assessment of more foundational type concepts such as variables, loops, conditional statements, data structures, and functions.

3. Method

In the present study, we examined the impact of a one-week project-based informal computer programming summer experience on students' perceptions of programming and programming knowledge in rural Washington where the availability of such opportunities is sparse.

3.1 Sample Information and Research Design

Nineteen middle school students ($M_{age} = 12.72$; $SD_{age} = 0.96$; Girls = 13, Boys = 6) participated in the summer programming camp. Majority of the students identified as Asian ($n = 10$), followed by Caucasian ($n = 3$), and Black ($n = 1$). The other students either preferred not to answer ($n = 4$) or indicated that their race/ethnicity was not listed ($n = 1$). Students self-selected based on their interest (or parents' interest) to attend the week-long summer programming camp at a large pacific northwestern university. Students who were a part of this programming camp had some familiarity with programming concepts before the week-long program. To examine the effect of the

programming camp exposure on students' programming motivation and knowledge, we used as a within-subjects research design. Students completed pre- and post- motivation surveys and learning assessments. This study was deemed exempt by the University's Institutional Review Board.

Table 1. Daily Breakdown of Programming Camp

Day	Programming Concept	Activities
1	A quick introduction to Python, run code from the command line, introduction to variables, lists, and Turtle library	Make snake game screen using Turtle library to make simple shapes.
2	Introduction to loops, conditional statements, user input, generating randomness	Implement functionality for snakehead placement and apple placement.
3	Introduction to functions	Implement functions and further snake game improvements.
4	No new content coverage	Finish snake game.
5	No new content coverage	Have fun and help students make improvements to the snake game.

3.2 Data Collection Tools

To examine the influence of the programming camp on students' motivation and interest in programming, we administered a 20-item survey. The 20-item survey was adapted from existing measures on students' interest and motivation in STEM (see Glynn et al., 2011; Korkmaz, 2017; Yadav et al., 2011), in addition to a handful of researcher-developed questions. The items were further divided into 5 subscales, Career Interest, Interest, Value, Critical Thinking, Proficiency. Students were asked to indicate the extent to which they agreed or disagreed with each of the following items using a 5-point Likert scale, where 1 = Strongly Disagree and 5 = Strongly Agree. See Table 2 for the list of items that were included on the survey and for each subscale.

To assess programming knowledge, we administered the same 10-item multi-step programming knowledge assessment before and after the camp (see Appendix A). The 10 items used in this programming knowledge test were researcher-developed. The items were developed around the concepts that were taught in the programming camp. The 10 items on the pre- and post-programming knowledge assessment were categorized into 3 broad categories related to CT skills: basic abstraction/operations, code analysis, and code writing. We decided to center on these CT skills, since we are able to connect the programming questions which exercise abstraction (basic abstraction/operations), problem formulation (code writing), and debugging/analysis (code analysis). However, we recognize that there is still ongoing discussion on what constitutes as CT skills within literature (Shute et al., 2017).

The basic abstraction/operations category contained questions related to basic programming operations like printing out values, mathematical operations, string concatenation, variables etc. In the code analysis category, snippets of code were provided to the participants for analysis. They were required to write out the output of the code snippet. This category also tested if participants could detect issues with the code snippet. The code snippets focused mainly

on loops and conditional statements. In the code writing category, challenges were presented to the participants, and they were required to write code to solve the challenge. For example, the participants were asked to write code that would print a phrase 10 times. The expectation was for the participants to use loops rather than write a print statement 10 times.

Table 2. Survey Items for Pre- & Post- Motivation

Career Interest
19. My knowledge of computer programming will help me choose a career in computing.
20. I am interested in a career in programming.
11. The computer programming skills I learn will be useful in my life.
14. I put effort into learning about computer programming.
12. I believe I can master computer programming knowledge and skills.
10. I will use computer programming skills in the future.
4. I will take more computer programming courses if I have the opportunity.
Interest
13. I enjoy learning about computer programming.
5. I have a special interest in mathematical processes.
1. Learning computer programming is interesting.
Value
2. Having an understanding of computer programming is valuable.
15. Understanding computer programming is important to me.
Critical Thinking
3. The challenge of solving problems using computer programming skills is appealing to me.
8. It is fun to try to solve complex computer programming problems.
9. I am willing to learn challenging computer programming problems.
18. What I already know about computer programming will help me think critically.
Proficiency
17. I am proficient in computer programming.
16. I know how to write computer programs.
7. I can easily understand the relationship between figures.
6. I can better learn instructions with the help of mathematical symbols and concepts.

3.3 Scoring

To calculate the total for each subscale on the motivation survey, we summed the students' responses for the items on each subscale. Both the pre- and post- motivation surveys had high reliability ($\alpha = .94$, and $\alpha = .96$, respectively). The individual subscales for the pre- and post- surveys also had moderate to high reliability with Cronbach's alpha ranging from .60 to .90.

To score the programming knowledge test, we graded the assessments based on an answer key developed by the instructors. We used the overall score of the assessments to determine whether there was a difference before and after the programming camp. First, the team evaluated and grouped questions based on programming concepts and the question type: basic concepts, original code, and code analysis. Second, two members of our team rated the mastery level for students' answers to each programming concept question from 1 (low understanding) to 5 (high understanding). Full agreement in inter-rater reliability was obtained between the two graders. Scores were obtained for the three broad categories basic operations (15 points), code analysis (15 points), and code writing (20 points), and the sum of these three categories provided the overall score (50 points). There was moderate to high reliability for the pre-assessment categories: basic operations, $\alpha = .50$, code analysis, $\alpha = .60$, and code writing, $\alpha = .84$. There

was also moderate to high reliability for the post-assessment categories: basic operations, $\alpha = .59$, code analysis, $\alpha = .46$, and code writing, $\alpha = .87$. Finally, there was moderate to high reliability for the pre- and post- assessment as a whole, $\alpha = .83$ and $\alpha = .74$, respectively. Examples of low and high rated answers are included in Appendix B.

3.4 Procedure

The computer programming camp was held in a spacious computer lab on the campus of a large university in the Pacific Northwest of the United States. Laptops equipped with the appropriate software were provided to students during the duration of the programming camp. On the first day, students completed both the pre-motivation survey and the pre-knowledge assessment test before commencing the camp activities. Each day, the instructors started with an overview of the day's lesson. Lessons on the programming concepts were interwoven into a hands-on project-based activity of building a snake game app from starter code. Instructors started with a brief lecture on core concepts for the day before walking through their own example code as students paid attention. Following this, students were given ample time to apply their newly acquired programming knowledge to the development of their game app. Instructors and teaching assistants provided one-on-one instructional support as needed. Each day comprised of at least two lectures and two sessions of individual coding time to develop the game app. This process is important because it allows students to examine how their knowledge of programming translates directly into the design and functionality of their game, which is likely to increase their appreciation and interest in programming. On the last day of the camp, students completed the post-motivation survey and the post-knowledge assessment test. An outline of each day's programming content coverage as it relates to the activities and project is provided on Table 1.

4. Results

To address our research question, separate analyses were conducted for the motivation and knowledge assessment measures. The results section is organized around these two analyses.

4.1 Motivation Analysis

To address RQ1, we analyzed data from pre-and post- motivation surveys which were administered on the first and last day of the programming camp. Nineteen students completed the pre- and post-motivation survey before and after the camp. There were 2 missing data entries for the pre-motivation survey and 2 missing data entries for the post-motivation survey. As the data was missing at random, we employed the EM algorithm to compute missing data points. The data was normally distributed for each of the motivation subscales. Table 3 provides the descriptive statistics for each individual subscale's score.

Table 3. Descriptive Statistics for Motivation Subscales

Assessment	Pre		Post		Cohen's <i>d</i>
	M	SD	M	SD	
Career Interest	26.62	4.80	26.95	4.60	0.16
Interest	12.58	1.68	12.21	1.87	-0.32
Value	8.42	1.35	8.05	1.35	-0.44
Critical Thinking	16.47	2.80	16.00	2.92	-0.39
Proficiency	13.21	2.37	14.53	2.41	0.99

Paired-samples t-tests were conducted to assess students' change in motivation score for the five subscales (career interest, interest, value, critical thinking, proficiency). Results showed that there were significant differences in students' score for the proficiency subscale, $t(18) = 4.29, p < .001$. Specifically, students self-reported higher proficiency after the programming camp ($M = 14.53, SD = 2.41$) as compared to before the camp ($M = 13.21, SD = 2.37; d = 0.99$). There were no significant differences for career interest, $t(18) = 0.59, p < .57$; interest, $t(18) = -1.38, p = .19$; value, $t(18) = -1.93, p < .07$; and critical thinking, $t(18) = -1.69, p = .11$.

Table 4. Descriptive Statistics for Programming Knowledge Assessments

Assessment	Pre		Post		Cohen's <i>d</i>
	M	SD	M	SD	
Overall	21.37	9.76	30.05	8.66	0.93
Basic Operations	10.84	3.67	13.26	2.90	0.71
Code Analysis	4.89	3.74	6.74	3.69	0.50
Code Writing	5.63	4.30	10.05	5.40	0.88

4.2 Programming Knowledge Analysis

To address RQ2, paired-samples t-tests were conducted to analyze the differences between the pre- and post-programming knowledge assessments. Both the overall assessment scores and the scores for each of the three categories were analyzed separately. Each of the knowledge assessment categories were considered (Basic Operations, Code Writing, Code Analysis, and Overall scores). Normality and outlier tests were performed on the overall scores (each category is a sub-score of the overall score). No outliers were detected. The assumption of normality was also not violated, as assessed by Shapiro-Wilk's test ($p = .313$). There were no outliers in the data, as assessed by inspection of a boxplot. Overall results indicate a statistically significant mean increase in programming knowledge, $t(18) = 5.82, p < .01$ (See Table 4 for descriptive statistics).

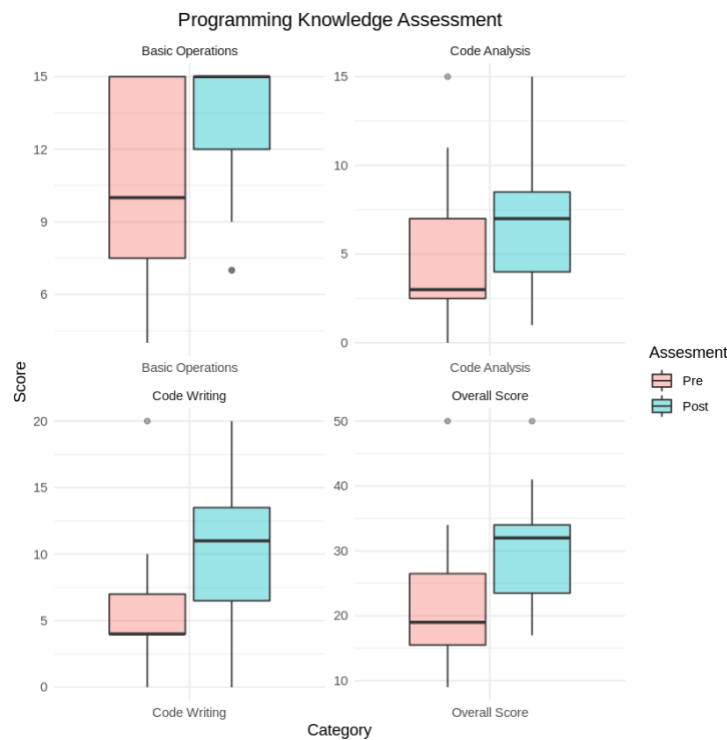


Figure 1. Programming Knowledge Assessment Boxplot

The results show that the students had significantly higher scores from the post-assessment. The mean difference between the pre- and post-assessment score also seem to suggest that the participants gained a lot of programming knowledge from the camp (Figure 1).

The results also show that the participants performed better in Basic Operations and Code Writing sections of the assessment. However, the increase in Code Analysis after the camp was moderate. A review of the activities done during the camp shows that the camp focused more on code writing and not analyzing existing code. In our future camps, we plan to bring code analysis more into focus in the curriculum.

5. Discussion

5.1 Motivation

Based on the results of our survey, no significant changes were observed in several aspects of students' motivation, such as career interest, interest in computing, the value of learning computing, perception of their critical thinking skills, and perception of proficiency in coding. However, students' perception of proficiency in coding did increase slightly. This finding is not surprising. Most students started the camp with basic understanding of programming. However, the snake game activity required them to integrate both their prior programming knowledge as well as the new programming knowledge taught at the camp for their app. Interestingly, we did not observe significant changes in the other motivation subscales across time. It may be possible that the short duration of the camp precluded students from fully exploring the possibilities of programming and the applicability of programming in their current lives.

5.2 Programming Knowledge Analysis

Based on the overall pre-and post- total score, our project-based curriculum for the weeklong programming camp was effective in increasing programming knowledge. This aligns with the expectations set by other studies which have used project-based approaches in two-week programming camps (Franklin et al., 2013) and other longer informal learning opportunities (Wang et al., 2019). The snake game allowed students in our programming camp to incrementally learn programming concepts while making progress and seeing their game come to fruition. Since the snake game required the usage of several core programming concepts, such as variables, basic operations, loops, data structures, conditional statements, and functions, students had to learn how to implement them for their snake game to work. For example, the students needed to know what purpose variables served in the snake game, such as an integer that kept track of scores, which instructors covered during the camp.

Testing for computational thinking skills in an informal education setting, like a programming camp, is very seldomly done in research (Tang et al., 2020). Although we did not explicitly test for all individual computational thinking skills directly, we were able to group questions based on three types of computational thinking skills, such as basic abstraction concepts, analysis/debugging of existing code, and algorithmic/logical thinking by original code construction (Tang et al., 2020). The programming camp was successful in increasing overall knowledge of core programming concepts; however, the results of our programming knowledge pre-and post- scores with grouped questions showed that there are some computing skills, such as code analysis and debugging, where students had some trouble answering.

The following sub-sections will discuss the role of computational thinking skills covered using the results of different sub-group question types such as basic abstraction concepts, code analysis/debugging, and original code construction.

5.2.1 Basic Abstraction Concepts

Students were asked to answer basic variable and variable manipulation questions using math operations. In terms of teaching the basic programming concepts, such as variables, the increase of understanding in these concepts covered in these questions could be attributed to the fact that students had a good starting point on how variables and operations could work within the context of the snake game. Since variables and performing mathematical and logical operations on variables is a level of abstraction K-12 students may not be familiar with, research shows that program execution or deep familiarity of the context in which these concepts will be used, such as a game, can help students visualize how these programming concepts work (Mladenović et al., 2021). Since the instructors had continuously demonstrated the snake game throughout the programming camp, students were able to make connections on how abstraction was used in creating game components, such as displaying scores, updating snake tail length, and changing values for their game customizations on the fly. For teaching students about basic abstraction concepts, like variables, helping students visualize their final project outcome by demonstrating the game can support their learning of programming concepts. Although visualization in the form of Powerpoint

animations, whiteboard examples, or sketching can provide support for students to learn basic abstraction concepts (Mladenović et al., 2021), our results show that demonstrating the project they will work on, like the snake game, and explicitly connecting it to programming concepts can be used as a visual aide to support their learning as well. This is aligned with several empirical studies which emphasized rich, visual coding experiences for students to learn basic abstraction concepts (Tang et al., 2020).

5.2.2 Code Analysis and Debugging

The second category of questions required students to analyze and debug code. Between pre-and post- scores for this group, developing this computational thinking skill did not change significantly. The lack of change between pre- and post- scores could be the fact that the instructors did not explicitly ask students to analyze pre-existing code or teach the specific code scenarios, such as the undefined variables and starting a while loop with a met condition. Although there have been many calls-to-actions to teach K-12 students a more systematic approach to debugging and analyzing code, it is common to not extensively cover debugging strategies when the goal is to generate interest in code (Michaeli & Romeike, 2019). However, since students do face issues while coding, Debugging and analysis of code on an on-demand basis to help students fix their code is the common approach, since students may sometimes attempt debugging techniques unsuccessfully, leaving them to feel helpless when they are unable to make progress (Michaeli & Romeike, 2019). In our case, we did not have enough time to strategically teach the students systematic ways to debug their code on their own, so it is reflected in our results for this question group.

Since creating code and debugging code are different skills (Michaeli & Romeike, 2019), it probably is the case that our students did not have ample time to develop their debugging skills, especially when looking at code that was not written by them. Analyzing and debugging others' code also requires more practice and development of their debugging skills both systematically and unsystematically (Bryant et al., 2019; Wilson, 2020). Reading and understanding code that was not written by them requires training students to decipher syntax and semantic meaning of the code (Lynch et al., 2019). It requires outlining and coming up with the conceptual picture of the code's intention, which requires practice (Busjahn & Schulte, 2013). It is to no surprise that students who attend programming camps with a short duration like ours likely did not develop these advanced computing skills due to lack of time to practice in class.

5.2.3 Algorithmic and Logical Thinking Skills

Questions in this third category required students to construct original code based on a particular prompt. For example, writing a loop that prints a string five times or constructs a function that adds two integers. According to the pre-and post-scores within this question group, there was a significant increase in students' algorithmic and logical thinking skills through constructing original code. Although instructors provided pieces of code to students, students were guided through the process of constructing original code for the snake game through daily incremental progress. Creating original code based on the prompt involves the development of algorithmic thinking skills, such as defining the problem, gathering relevant and applicable concepts, thinking of the logical steps, and writing the code (Braswell, 2020; Young et al., 2017). Each day, students were tasked to complete progress on another snake game milestone, such as the snake game interface on Day 1 or game piece placements on Day 2. To complete this functionality, instructors introduced the relevant programming concepts needed to complete those game milestones for that day, such as loops and conditional statements on Day 2. Using those concepts, students constructed the next snake game milestone with instructor guidance in algorithmically thinking through the problem. Although there are not many studies in informal learning context on project-based curriculums for developing algorithmic thinking skills, there are several studies in K-12 education that show that project-based curriculum can help teach students algorithmic thinking skills (Chiazzese et al., 2018; Garneli et al., 2015; Karaman et al., 2017).

6. Limitations

Although our one-week programming camp provided opportunities to learn how to code and explore computing to middle school students, we recognize that our single study, sample size, and location may not be generalizable to other groups. This may limit potential replications of our project-based short programming camp experience. Currently, we are in the process of collecting more programming camp data to strengthen our developing findings on programming knowledge and motivation to learn coding.

Secondly, our programming knowledge assessment reflected our curriculums' content coverage, but we realize that it may not have been appropriate to test students to analyze and debug code during the assessment since we did not

intentionally cover it during the camp. In the next programming camp, we plan to either simplify our programming camp assessment questions to include and cover simpler forms of code analysis and debugging and/or remove these questions.

Thirdly, we recognize that the participants self-selected to participate in the programming camp and students already came in with some basic understanding. Maintaining interest after sparking initial interest increases the likelihood of future pursuit of a related career (Christensen et al., 2014; Hidi & Renninger, 2006; Hirsch et al., 2017; Taub et al., 2012); however, we must be careful to not generalize by saying that the programming camp was successful in promoting interest where no interest may have existed in middle school students who attend the camp. Since they were self-selected, it is not surprising that their interest and motivation was high.

7. Conclusion

Informal learning environments, such as programming camps, can provide the opportunity to empower students to create a project from the ground up while learning basic programming concepts. However, instructors need to balance content coverage in terms of introducing other fundamental computational thinking skills, such as debugging and analyzing code. To keep students interested in the programming camp, we may need to temporarily forgo teaching them (and testing them) on more complex computational thinking skills such as reading code that was not created by them and debugging skills.

Regardless of making cuts to content coverage, a week-long project-based programming camp can inspire and teach students to code in a short amount of time. Although our programming camp did not significantly change students' attitudes towards pursuing computing due to students coming in with high interest in programming, we did significantly increase their programming knowledge and their perceptions of their ability to code, which could support their self-efficacy to jumpstart and continue exploring the tech field in high school, and, hopefully, into college.

Acknowledgments

This work was supported by the Boeing Distinguished Professorship funds granted to Dr. Olusola Adesope by Washington State University.

References

- 2018 NSSME+. (n.d.). *NSSME*. Retrieved April 5, 2021, from <http://horizon-research.com/NSSME/2018-nssme>
- Adams, J. C. (2010). Scratching middle schoolers' creative itch. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 356–360. <https://doi.org/10.1145/1734263.1734385>
- Akcaoglu, M. (2014). Learning problem-solving through making games at the game design and learning summer program. *Educational Technology Research and Development*, 62(5), 583–600. <https://doi.org/10.1007/s11423-014-9347-4>
- Al-Bow, M., Austin, D., Edgington, J., Fajardo, R., Fishburn, J., Lara, C., Leutenegger, S., & Meyer, S. (2009). Using game creation for teaching computer programming to high school students and teachers. *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, 104–108. <https://doi.org/10.1145/1562877.1562913>
- Austin, K., & Pinkard, N. (2008). The organization and management of informal and formal learning. *Proceedings of the 8th International Conference on International Conference for the Learning Sciences - Volume 3*, 5–7.
- Ayar, M. C., & Yalvac, B. (2016). Lessons Learned: Authenticity, Interdisciplinarity, and Mentoring for STEM Learning Environments. *International Journal of Education in Mathematics, Science and Technology*, 4(1), 30–43.
- Bagiati, A., Yoon, S. Y., Evangelou, D., & Ngambeki, I. (2010). Engineering Curricula in Early Education: Describing the Landscape of Open Resources. *Early Childhood Research and Practice* , 12(2). <https://files.eric.ed.gov/fulltext/EJ910909.pdf>
- Bell, P., Lewenstein, B., Shouse, A. W., & Feder, M. A. (2009). *Learning science in informal environments: People, places, and pursuits* (p. 12190). National Academies Press. <https://doi.org/10.17226/12190>

- Braswell, K. M. (2020). From Camp to Conferences: Experiences in Leveraging Tech Conferences to Inspire Black and Latinx Girls to Pursue Coding and Tech Careers. *2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*, 1, 1–4. <https://doi.org/10.1109/RESPECT49803.2020.9272429>
- Bryant, C., Chen, Y., Chen, Z., Gilmour, J., Gumidyala, S., Herce-Hagiwara, B., Koures, A., Lee, S., Msekela, J., Pham, A. T., Remash, H., Remash, M., Schoenle, N., Zimmerman, J., Dahlby Albright, S., & Rebelsky, S. A. (2019). A Middle-School Camp Emphasizing Data Science and Computing for Social Good. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 358–364. <https://doi.org/10.1145/3287324.3287510>
- Burack, C., Melchior, A., & Hoover, M. (2018). *Do After-school Robotics Programs Expand the Pipeline into STEM Majors in College? (RTP)*. 2018 ASEE Annual Conference & Exposition, Salt Lake City, Utah. <https://doi.org/10.18260/1-2--30341>
- Bureau of Labor Statistics. (2019). *Computer and Information Technology* (Occupational Outlook Handbook). U.S. Department of Labor. <https://www.bls.gov/ooh/computer-and-information-technology/home.htm>
- Busjahn, T., & Schulte, C. (2013). The use of code reading in teaching programming. *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, 3–11. <https://doi.org/10.1145/2526968.2526969>
- Cabrera, R., de los Ángeles Carrión, M., & Carrión, A. (2021). Camps IEEE Ecuador: A proposal to increase children's interest in STEM areas. *2021 IEEE XXVIII International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, 1–4. <https://doi.org/10.1109/INTERCON52678.2021.9532982>
- Chaudhary, V., Agrawal, V., Sureka, P., & Sureka, A. (2016). An Experience Report on Teaching Programming and Computational Thinking to Elementary Level Children Using Lego Robotics Education Kit. *2016 IEEE Eighth International Conference on Technology for Education (T4E)*, 38–41. <https://doi.org/10.1109/T4E.2016.016>
- Chiazzese, G., Arrigo, M., Chifari, A., Lonati, V., & Tosto, C. (2018). Exploring the Effect of a Robotics Laboratory on Computational Thinking Skills in Primary School Children Using the Bebras Tasks. *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality*, 25–30. <https://doi.org/10.1145/3284179.3284186>
- Christensen, R., Knezek, G., Tyler-Wood, T., & Gibson, D. (2014). Longitudinal analysis of cognitive constructs fostered by STEM activities for middle school students. *Knowledge Management and E-Learning*, 6, 103–122.
- Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, 58(1), 240–249. <https://doi.org/10.1016/j.compedu.2011.08.006>
- DeWitt, A., Fay, J., Goldman, M., Nicolson, E., Oyolu, L., Resch, L., Saldaña, J. M., Sounalath, S., Williams, T., Yetter, K., Zak, E., Brown, N., & Rebelsky, S. A. (2017a). Arts Coding for Social Good: A Pilot Project for Middle-School Outreach. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 159–164. <https://doi.org/10.1145/3017680.3017795>
- DeWitt, A., Fay, J., Goldman, M., Nicolson, E., Oyolu, L., Resch, L., Saldaña, J. M., Sounalath, S., Williams, T., Yetter, K., Zak, E., Brown, N., & Rebelsky, S. A. (2017b). What We Say vs. What They Do: A Comparison of Middle-School Coding Camps in the CS Education Literature and Mainstream Coding Camps (Abstract Only). *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 707. <https://doi.org/10.1145/3017680.3022434>
- Ericson, B., & McKlin, T. (2012). Effective and sustainable computing summer camps. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 289–294. <https://doi.org/10.1145/2157136.2157223>
- Fields, D. A., Quirke, L. C., & Amely, J. (2015). Measuring learning in an open-ended, constructionist-based programming camp: Developing a set of quantitative measures from qualitative analysis. *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 15–17. <https://doi.org/10.1109/BLOCKS.2015.7368993>

- Franklin, D., Conrad, P., Boe, B., Nilsen, K., Hill, C., Len, M., Dreschler, G., Aldana, G., Almeida-Tanaka, P., Kiefer, B., Laird, C., Lopez, F., Pham, C., Suarez, J., & Waite, R. (2013). Assessment of computer science learning in a scratch-based outreach program. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 371–376. <https://doi.org/10.1145/2445196.2445304>
- Frye, M. T., Nair, S. C., & Meyer, A. (2016). Evaluation of miniGEMS 2015 – Engineering Summer Camp for Middle School Girls. *2016 ASEE Annual Conference & Exposition Proceedings*, 7.
- Garneli, V., Giannakos, M. N., & Chorianopoulos, K. (2015). Computing education in K-12 schools: A review of the literature. *2015 IEEE Global Engineering Education Conference (EDUCON)*, 543–551. <https://doi.org/10.1109/EDUCON.2015.7096023>
- Hidi, S., & Renninger, K. A. (2006). The Four-Phase Model of Interest Development. *Educational Psychologist*, 41(2), 111–127. https://doi.org/10.1207/s15326985ep4102_4
- Hirsch, L. S., Berliner-Heyman, S., & Cusack, J. L. (2017). Introducing Middle School Students to Engineering Principles and the Engineering Design Process Through an Academic Summer Program. *INTERNATIONAL JOURNAL OF ENGINEERING EDUCATION*, 33(1, B), 398–407.
- Hofstein, A., & Rosenfeld, S. (1996). Bridging the Gap Between Formal and Informal Science Learning. *Studies in Science Education*, 28(1), 87–112. <https://doi.org/10.1080/03057269608560085>
- Hugerat, M. (2016). How teaching science using project-based learning strategies affects the classroom learning environment. *Learning Environments Research*, 19(3), 383–395. <https://doi.org/10.1007/s10984-016-9212-y>
- Jones, S. (2019, January 8). STEM Instruction: How Much There Is and Who Gets It. *Education Week*. <https://www.edweek.org/teaching-learning/stem-instruction-how-much-there-is-and-who-gets-it/2019/01>
- Karaman, S., Anders, A., Boulet, M., Connor, J., Gregson, K., Guerra, W., Guldner, O., Mohamoud, M., Plancher, B., Shin, R., & Vivilecchia, J. (2017). Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at MIT. *2017 IEEE Integrated STEM Education Conference (ISEC)*, 195–203. <https://doi.org/10.1109/ISECon.2017.7910242>
- Lakanen, A.-J., & Kärkkäinen, T. (2019). Identifying Pathways to Computer Science: The Long-Term Impact of Short-Term Game Programming Outreach Interventions. *ACM Transactions on Computing Education*, 19(3), 20:1-20:30. <https://doi.org/10.1145/3283070>
- LePendu, P., Cheung, C., Salloum, M., Sheffler, P., & Downey, K. (2020). Summer Coding Camp as a Gateway to STEM. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 1351. <https://doi.org/10.1145/3328778.3372637>
- Lusa Krug, D., Bowman, E., Barnett, T., Pollock, L., & Shepherd, D. (2021). Code Beats: A Virtual Camp for Middle Schoolers Coding Hip Hop. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 397–403. <https://doi.org/10.1145/3408877.3432424>
- 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>
- Lynch, K., Hill, H. C., Gonzalez, K. E., & Pollard, C. (2019). Strengthening the Research Base That Informs STEM Instructional Improvement Efforts: A Meta-Analysis. *Educational Evaluation and Policy Analysis*, 41(3), 260–293. <https://doi.org/10.3102/0162373719849044>
- Maiorca, C., Roberts, T., Jackson, C., Bush, S., Delaney, A., Mohr-Schroeder, M. J., & Soledad, S. Y. (2021). Informal Learning Environments and Impact on Interest in STEM Careers. *International Journal of Science and Mathematics Education*, 19(1), 45–64. <https://doi.org/10.1007/s10763-019-10038-9>
- Master, A., Cheryan, S., Moscatelli, A., & Meltzoff, A. N. (2017). Programming experience promotes higher STEM motivation among first-grade girls. *Journal of Experimental Child Psychology*, 160, 92–106. <https://doi.org/10.1016/j.jecp.2017.03.013>

- McCombs, J. S., Rand Education (Institute), & Wallace Foundation (Eds.). (2011). *Making summer count: How summer programs can boost children's learning*. RAND.
- Michaeli, T., & Romeike, R. (2019). Current Status and Perspectives of Debugging in the K12 Classroom: A Qualitative Study. *2019 IEEE Global Engineering Education Conference (EDUCON)*, 1030–1038. <https://doi.org/10.1109/EDUCON.2019.8725282>
- Miller, K., Sonnert, G., & Sadler, P. (2018). The Influence of Students' Participation in STEM Competitions on Their Interest in STEM Careers. *International Journal of Science Education, Part B: Communication and Public Engagement*, 8(2), 95–114. <https://doi.org/10.1080/21548455.2017.1397298>
- Mladenović, M., Žanko, Ž., & Aglič, M. (2021). The impact of using program visualization techniques on learning basic programming concepts at the K–12 level. *Computer Applications in Engineering Education*, 29, 145–159. <https://doi.org/10.1002/cae.22315>
- Mohr-Schroeder, M., Jackson, C., Miller, M., Walcott, B., Little, D., Speler, L., Schooler, W., & Schroeder, D. (2014). Developing Middle School Students' Interests in STEM via Summer Learning Experiences: See Blue STEM Camp. *School Science and Mathematics*, 114. <https://doi.org/10.1111/ssm.12079>
- National Science Board. (2016). *Science and Engineering Indicators 2016*. (NSB-2016-1). National Science Foundation.
- Nite, S. B., Bicer, A., Currens, K. C., & Tejani, R. (2020). Increasing STEM Interest through Coding with Microcontrollers. *2020 IEEE Frontiers in Education Conference (FIE)*, 1–7. <https://doi.org/10.1109/FIE44824.2020.9274273>
- Outlay, C. N., Platt, A. J., & Conroy, K. (2017). Getting IT Together: A Longitudinal Look at Linking Girls' Interest in IT Careers to Lessons Taught in Middle School Camps. *ACM Transactions on Computing Education*, 17(4), 20:1-20:17. <https://doi.org/10.1145/3068838>
- Roberts, T., Jackson, C., Mohr-Schroeder, M. J., Bush, S. B., Maiorca, C., Cavalcanti, M., Craig Schroeder, D., Delaney, A., Putnam, L., & Cremeans, C. (2018). Students' perceptions of STEM learning after participating in a summer informal learning experience. *International Journal of STEM Education*, 5(1), 35. <https://doi.org/10.1186/s40594-018-0133-4>
- Sadler, K., Eilam, E., Bigger, S. W., & Barry, F. (2018). University-led STEM outreach programs: Purposes, impacts, stakeholder needs and institutional support at nine Australian universities. *Studies in Higher Education*, 43(3), 586–599. <https://doi.org/10.1080/03075079.2016.1185775>
- 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>
- Stewart, O. G., & Jordan, M. E. (2017). "Some explanation here": A case study of learning opportunities and tensions in an informal science learning environment. *Instructional Science*, 45(2), 137–156. <https://doi.org/10.1007/s11251-016-9396-7>
- Tai, R. H., Liu, C. Q., Maltese, A. V., & Fan, X. (n.d.). *Planning Early for Careers in Science*. 2.
- Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, 148, 103798. <https://doi.org/10.1016/j.compedu.2019.103798>
- Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS Unplugged and Middle-School Students' Views, Attitudes, and Intentions Regarding CS. *ACM Transactions on Computing Education*, 12(2), 8:1-8:29. <https://doi.org/10.1145/2160547.2160551>
- Wang, C., & Frye, M. (2019). miniGEMS 2018 Summer Camp Evaluation: Empowering Middle School Girls in STEAM. *2019 IEEE Integrated STEM Education Conference (ISEC)*, 149–155. <https://doi.org/10.1109/ISECon.2019.8881981>
- Wang, C., Frye, M., & Nair, S. (2019, April 5). *The Practices of Play and Informal Learning in the miniGEMS STEAM Camp*. 2018 Gulf Southwest Section Conference. <https://peer.asee.org/the-practices-of-play-and-informal-learning-in-the-minigems-steam-camp>

- Warner, J. R., Fletcher, C. L., Torbey, R., & Garbrecht, L. S. (2019). Increasing Capacity for Computer Science Education in Rural Areas through a Large-Scale Collective Impact Model. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 1157–1163. <https://doi.org/10.1145/3287324.3287418>
- Webb, H. C., & Rosson, M. B. (2011). Exploring careers while learning Alice 3D: A summer camp for middle school girls. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 377–382. <https://doi.org/10.1145/1953163.1953275>
- Weinberg, A. E., Basile, C. G., & Albright, L. (2011). The Effect of an Experiential Learning Program on Middle School Students' Motivation toward Mathematics and Science. *RMLE Online: Research in Middle Level Education*, 35(3), 1–12.
- Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: Measuring computational thinking in middle school. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 215–220. <https://doi.org/10.1145/2157136.2157200>
- Wilson, K. (2020). Exploring the Challenges and Enablers of Implementing a STEM Project-Based Learning Programme in a Diverse Junior Secondary Context. *International Journal of Science and Mathematics Education*. <https://doi.org/10.1007/s10763-020-10103-8>
- Xianglei, C., & Weko, T. (2009). *Students who study science, technology, engineering, and mathematics (STEM) in postsecondary education*. National Center for Education Studies. <http://nces.ed.gov/pubsearch/pubsinfo.asp?pubid=2009161>
- Young, J. R., Ortiz, N., & Young, J. L. (2017). STEMulating Interest: A Meta-Analysis of the Effects of Out-of-School Time on Student STEM Interest. *International Journal of Education in Mathematics, Science and Technology*, 5(1), 62–74.
- Zamin, N., Ab Rahim, H., Savita, K. S., Bhattacharyya, E., Zaffar, M., & Katijah Mohd Jamil, S. N. (2018). Learning Block Programming using Scratch among School Children in Malaysia and Australia: An Exploratory Study. *2018 4th International Conference on Computer and Information Sciences (ICCOINS)*, 1–6. <https://doi.org/10.1109/ICCOINS.2018.8510586>
- Zweben, S., & Bizot, B. (2020). *Total Undergrad CS Enrollment Rises Again, but with Fewer New Majors; Doctoral Degree Production Recovers From Last Year's Dip*. 61.

Appendix A

Pre- and Post-Knowledge Assessment Administered

Boeing Programming Boot Camp for Middle Scholar 2019

1- What are these variables (integer, float, or string)?

n = 10

x = 0.98

s = 'dog'

ANSWER: n is an Integer; x is a float, s is a string

2- What is the result of code below?

20-2*(3+5)

ANSWER = 4

3- What is the result of code below?

```
x = 10
y = 30
z = 400
z - y * x
ANSWER = 100
```

4- what would this code print?

```
while n > 10:
    print(n)
    n = n+1
ANSWER = it prints nothing
```

5- what dose the code below prints?

```
jar = ['candy', 'gums', 'm&m']
hungry = True
for x in jar:
    if x == 'gums':
        print('jane is happy')
    elif x == 'candy':
        print('alex is happy')
    elif x == 'm&m':
        if hungry:
            print('I need food')
        else:
            print('party time')
    else:
        print('marry is happy')
```

ANSWER:

```
alex is happy
jane is happy
I need real food
```

6- what would this code print?

```
n = 10
```

while n > 10:

print(n)

n = n+1

ANSWER: it prints nothing

7- Write a loop that prints 'GO COUGES!' five times:

ANSWER1:

n = 0

while n < 5:

print('GO COUGS!')

n = n+1

ANSWER2:

for i in range(5):

print('GO COUGS!')

BOTH ANSWERS ARE CORRECT

8- Answer the following questions based on the below list

names = ['kris', 'aj', 'jake', 'robert', 'liz']

a- write a loop to print all the elements of the list \$x\$:

ANSWER1:

for name in names:

print(name)

ANSWER2:

n_names = len(names)

for idx in range(n_names):

print(names[idx])

ANSWER3:

idx = 0

n_names = len(names)

while idx < n_names:

print(names[idx])

idx = idx+1

ANY OF THESE ANSWERS ARE CORRECT

b- what is the result of the code below?

```
print(len(x))
```

ANSWER = 5

9- write a function that takes two variables in its argument and returns the addition.

ANSWER:

```
def add(n1, n2):  
    return n1+n2
```

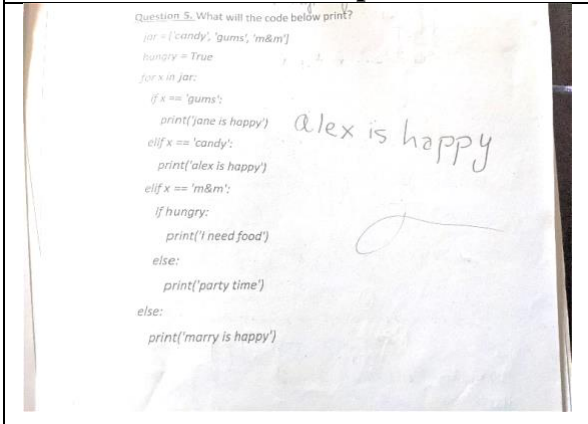
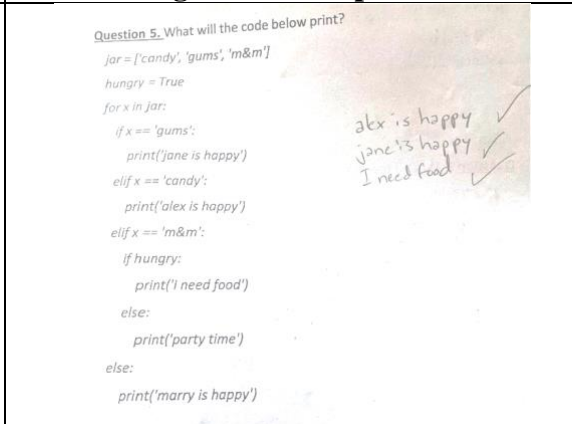
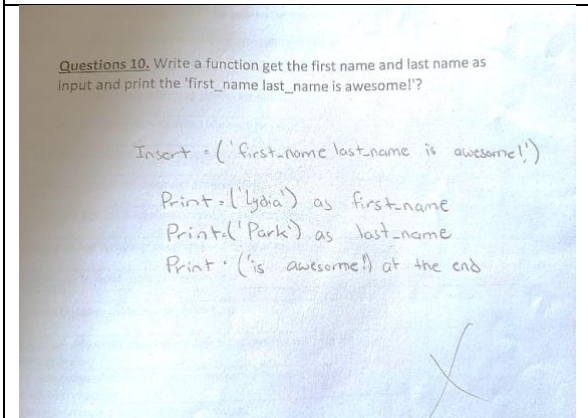
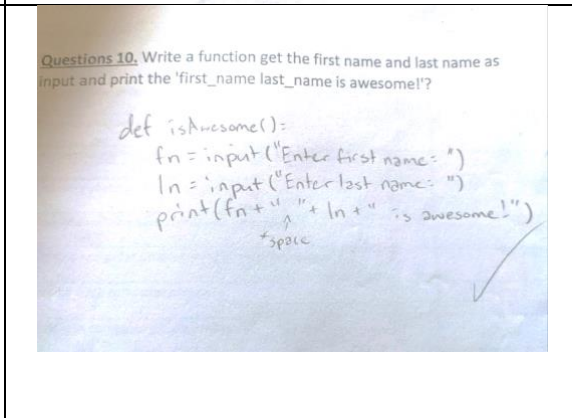
10- Write a function get the first name and last name as input and print the 'first_name last_name is awesome!'

ANSWER:

```
def awesome(first_name, last_name):  
  
    print(first_name + ' ' + last_name + 'is awesome!')
```

Appendix B

Samples of Low and High Rated Responses to Programming Questions

Low Rated Responses	High Rated Responses
 <p>Question 5. What will the code below print?</p> <pre>jar = ['candy', 'gums', 'm&m'] hungry = True for x in jar: if x == 'gums': print('jane is happy') elif x == 'candy': print('alex is happy') elif x == 'm&m': if hungry: print('i need food') else: print('party time') else: print('marry is happy')</pre> <p>alex is happy</p>	 <p>Question 5. What will the code below print?</p> <pre>jar = ['candy', 'gums', 'm&m'] hungry = True for x in jar: if x == 'gums': print('jane is happy') elif x == 'candy': print('alex is happy') elif x == 'm&m': if hungry: print('i need food') else: print('party time') else: print('marry is happy')</pre> <p>alex is happy ✓ jane is happy ✓ I need food ✓</p>
 <p>Questions 10. Write a function get the first name and last name as input and print the 'first_name last_name is awesome!'</p> <pre>Insert = ('first_name last_name is awesome!')</pre> <p>Print ('Lydia') as first_name Print ('Park') as last_name Print ('is awesome!') at the end</p>	 <p>Questions 10. Write a function get the first name and last name as input and print the 'first_name last_name is awesome!'</p> <pre>def isAwesome(): fn = input("Enter first name: ") ln = input("Enter last name: ") print(fn + " " + ln + " is awesome!")</pre> <p>*space</p>

Computer Science Teacher Capacity: The Need for Expanded Understanding

David AMIEL¹
Cynthia BLITZ¹

¹ Rutgers University Center for Effective School Practices, New Brunswick, USA

DOI: 10.21585/ijcses.v5i4.151

Abstract

With the increasing need for the incorporation of computer science (CS) concepts into elementary and secondary education, it is imperative that the teaching workforce is adequately prepared to ensure that instruction in CS is robust, relevant, and aligned with appropriate learning standards, where appropriate. This paper shares results from a recent survey administered to current computer science educators across the K-12 space in the state of New Jersey. Using these results and recent literature, this research distills actionable, assessed needs to guide the provision of professional learning to ensure that educators have the necessary tools and knowledge to ensure robust and equitable implementation of computer science education. Results point towards a need to expand the present understanding of computer science by effectively differentiating CS from technology-based instruction and addressing an overrepresentation of analytical content domains, reaffirm a commitment to equity by acknowledging the persistent gaps in participation of marginalized student groups, and critically examine when and where the use of technology is necessary in delivering CS instruction.

Keywords: computer science education, elementary education, secondary education, professional learning, diversity equity and inclusion

1. Introduction

As a relatively new field, computer science (CS), and the education of it, are continuously expanding to meet growing economic, social, and cultural needs surrounding students' computational thinking and overall digital literacy (Webb et al., 2017). As the climate surrounding computing changes, society relies on the next generation of students to become competitive innovators and cultural drivers in a reality that is increasingly dependent on technology (Webb et al., 2017). At the same time, computer science education benefits students by challenging the way they think, and by teaching them to approach problems in novel and rigorous ways that are innately rooted in logical reasoning (Nager & Atkinson, 2016). According to the U.S. Bureau of Labor Statistics, jobs in the field of computer and information research are projected to increase by 22% by 2030, faster than the average for all other occupations (Bureau of Labor Statistics, 2022). Additionally, wages have been steadily increasing in the field, making computer science education an excellent opportunity for economic advancement (Nager & Atkinson, 2016).

Despite the value of computer science education (CSE), a recent study revealed that only 19 percent of high school seniors had taken any computer science courses (Nager & Atkinson, 2016). The study also noted a gender gap with only 14% of females having taken a computer science course, compared to 24% of males (Nager & Atkinson, 2016); female students of color are represented even less (Code.org et al., 2021). Researchers attribute this disparity to additional barriers faced by BIPOC learners, including lack of social encouragement, self-efficacy beliefs, academic exposure, and career perception (Nager & Atkinson, 2016). These participation gaps are also

visible within the current computing workforce, in which women account for only 25% of positions across the world (Scott et al., 2017). Women of color are even more staunchly underrepresented in the workforce than their white counterparts, with African American, Latinx, Native American, Native Hawaiian, and Pacific Islander women in the U.S. comprising 20% of the population but only holding only 4% of positions in the computing workforce (Scott et al., 2017).

Even considering the growing centrality of CS in education, many current computer science teachers have never received adequate pre-service training in computational methods (Nager & Atkinson, 2016), meaning they may be unfamiliar with CS content knowledge and pedagogical practices. Many computer science teachers are also new to the field of education itself, with under 20% having more than 10 years of experience (Code.org et al., 2021). At this time, most states still do not require individuals teaching computer science courses to hold a certification in computer science (Code.org et al., 2021) and, at the time of writing, only 46% of computer science teachers across the country held a credential in computer science, with 23% of these teachers holding a CTE credential (Code.org et al., 2021), opposed to a credential dedicated to computer science.

Thus, pre-service teacher preparation and in-service professional development are promising avenues to both prepare teachers to deliver CS instruction and to introduce pedagogical practices that have shown promise in effectively conveying concepts in CS and computational thinking while broadening participation in the field, effectively tackling persistent participation and achievement gaps. At this time, many efforts have occurred at a national scale, including CSforAll and Code.org, among others.

1.1. The Case in New Jersey

Computer science has only recently become a standard discipline in public schools in both the U.S. and across the world (Andrew et al., 2016). Specifically in New Jersey (NJ), although computer science is not a graduation requirement, high schools are required to *offer* CS classes. Since this mandate was enacted in January 2018 (and going into effect that fall), school-level implementation of classes have been a challenge: in 2020, only 68% of high schools offered any form of computer science course (Code.org et al., 2021). Recently, NJ has adopted student learning standards in computer science and design thinking, which span the entire K-12 space.

For the first time, starting in the 2022 school year, CSE will become a mandatory part of education in the K-8 space in NJ schools. Similar trends are unfolding nationwide, with schools across the United States beginning to adopt computer science across the K-12 spectrum (Code.org et al., 2021). Organizations like *CS is Elementary* are working to spread awareness and increase adoption of computer science instruction at lower grade levels. Given that unlike other subject areas, computer science does not benefit from decades of research on best practices in content delivery and teacher preparation, efforts seeking to collect information about where computer science is being taught and what that instruction looks like will be important in understanding where and how CSE is successful.

Alongside such learning standards, educators will require support as they prepare to implement CSE in their classes. This professional learning, although inclusive of content and pedagogical knowledge, must also extend beyond the traditional scope of professional development to increase understanding of the broad scope and definition of the discipline itself. This is especially important considering that the New Jersey standards, as well as those from the national Computer Science Teachers Association standards (CSTA), include elements that extend beyond coding and analysis, which are commonly misunderstood to represent the *entirety* of CS.

Additionally, the timely implementation of computer science standards provides a platform to address the persistent issues of equity and participation in computer science education. As all students (regardless of race, gender, socioeconomic status, or ethnicity) will be required to receive computer science instruction. This provides educators with opportunity to broaden the CS pipeline from younger grade levels and promote more authentic and responsive engagement with the subject.

To support this work, New Jersey has piloted three professional learning hubs centered at universities across the state. These hubs, based on the model of the United Kingdom's Teaching School Hubs (Department for Education), are designed to provide professional learning to educators across the state to equip them with the content and pedagogical knowledge (inclusive of culturally responsive and equitable practices) needed to ensure the success of the standards' implementation.

2. Methods

The purpose of this research is to generate a snapshot of computer science educators in New Jersey that can inform the allocation of resources for the intentional, effective provision of professional learning. To this end, a 20-

question survey was developed and distributed to attendees of an annual computer science education summit. The research acknowledges a sampling bias of those completing the survey, who are significantly more likely to themselves be computer science teachers, or at least involved with CSE in some way. However, although the survey was administered to a convenience sample, given that the purpose of their insight is to inform resource allocation, we are confident that the assessed needs of this sample will provide, at the very least, a lower bound for the more broadly needed supports (as those completing the survey are likely to be among those receiving such supports). The survey contained several demographic questions, a series of scale items adapted from a similar instrument (Banilower et al., 2018), as well as items directly related to the implementation of NJ CS Standards. *Appendix A* contains the survey questions that were used in the analyses discussed in this paper. This research sought to address the following questions:

1. What are teachers' perceptions of their own knowledge of various content domains of computer science, and how confident are teachers in delivering CS instruction? (*Instructional Practices*)
2. What are teachers' perceptions of the barriers to implementing rigorous computer science instruction? (*Institutional Practices*)
3. How can professional learning better be tailored to bolster teacher capacity using findings related to teachers' perceptions of the above instructional and institutional practices in CSE?

All routine data analyses took place in SPSS Version 28, including the creation of scale item aggregate scores, the generation of frequency tables, and the calculation of reported means and standard deviations. This survey is part of a larger campaign to gather robust and relevant information on the landscape of computer science educators across the state and disseminate such information to educational agencies, policymakers and advocates, and professional learning providers to maximize the cumulative impact of efforts to support computer science education. This study received approval from the Rutgers University Institutional Review Board.

3. Results

3.1. About the Sample

The survey was sent to a total of 93 individuals and received a total of 41 responses; after removing abandoned submissions and those that did not pass screening questions, 29 valid responses remained and were used in the analysis discussed throughout this paper. After an initial response rate of 44%, 31% of responses were used in analysis. Of this sample, 83% (n=24) identified as female, and the remaining 17% (n=5) identified as male. The vast majority, 90%, of respondents identified as White or Caucasian, while 3.3% identified as Black or African American, 3.3% Asian American or Pacific Islander, and 3.3% as mixed race (Black & Caucasian). 10% reported that they have no formal training or education specifically in computer science, 35% reported formal training or education less than an undergraduate minor in CS, 7% reported obtaining an undergraduate CS minor, 38% reported an undergraduate CS major, and 10% reported their highest education in computer science is a graduate degree.

Figure 1 shows a heatmap of valid survey responses received across the state, with yellow and orange indicating a higher concentration of responses. Each data point represents the geographic region of where the individual teaches computer science, not necessarily their place of residence. Of these, 52% of individuals reported teaching in a district where most students are from poor or working-class families, 41% reported teaching in a district where most students are from middle class families, and the remaining 7% reported teaching in predominately wealthy districts.

Most respondents indicated teaching computer science most recently at the high school level, with 33% indicating they teach computer science in grades 11-12 and 30% indicating they teach computer science in grades 9-10. Outside of high school grades, 17% indicated they most recently taught computer science in grades 6-8, 11% in grades 3-5, and 9% in grades PreK-2. Further, the majority (55%) indicated that they exclusively teach computer science, with 21% indicating that 0% - 50% of their course load is computer science and 24% reporting between 50% and 75%.

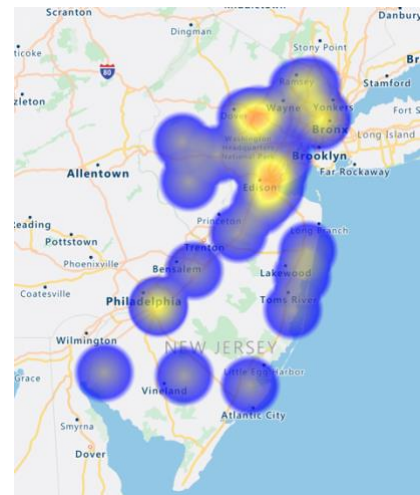


Figure 2. Heatmap of Analyzed Responses

Table 1. Teachers' Perceptions of Content Knowledge of CS Standard Domains (N=29)

	I have not heard of these concepts (1)	I have heard of these concepts (2)	I am familiar with these concepts (3)	I have mastered these concepts (4)
Algorithms & Programming	0%	10%	45%	45%
Computing Systems	0%	7%	55%	38%
Data & Analysis	0%	17%	48%	35%
Effects of Computing	7%	17%	52%	24%
Engineering Design	3%	17%	59%	21%
Ethics of Computing	0%	24%	48%	28%
Impacts of Computing	0%	14%	45%	41%
Interaction of Humans & Machines	3%	7%	59%	31%
Nature of Technology	3%	28%	48%	21%
Networks & the Internet	0%	7%	66%	28%

3.2. Instructional Practices

Participating educators shared insights on their self-perceptions of knowledge for each content domain covered in the 2020 New Jersey CS Student Learning Standards. On a scale from 1 (“I have not heard of these concepts”) through 4 (“I have mastered these concepts”), educators reported they were most comfortable with algorithms and programming, with an average weighted score of 3.34, with standard deviation of 0.67. From highest to lowest weighted score, educators rated their confidence with the various disciplinary domains as computing systems (3.31, SD=.60), impact of computing (3.28, SD=.70), networks and the internet (3.20, SD=.55), data and analysis (3.17, SD=.71), interaction of humans and machines (3.17, SD=.71), ethics of computing (3.03, SD=.73), engineering design (2.96, SD=.73), effects of computing (2.93, SD=.84), and nature of technology (2.86, SD=.79). *Table 1* further details the distribution of responses for each content area.

Participants also shared how prepared they feel in delivering computer science instruction. On a scale of 1 (“Not adequately prepared”) to 4 (“Very prepared”), teachers rated, on average, that they were most prepared to teach the relevance of computer science (3.6 weighted score, SD=.56), followed by teaching introductory computer science (3.55, SD=.57), ensure equal student participation (3.55, SD=.69), garner student interest in computer science (3.52, SD=.51), foster group interactions with students (3.45, SD=.69), reach students from traditionally underrepresented populations (3.41, SD=.73), teach using a guided inquiry approach (3.14, SD=.74), and utilize culturally relevant teaching and associated pedagogies (2.93, 70); teachers reported they felt least prepared to teach computer science to English Language Learners (2.34, SD=.81). *Table 2* shows the distribution of responses related to participants’ preparation for computer science instruction.

Table 2. Teacher-Reported Levels of Preparation to Deliver Computer Science Instruction (N=29)

	Not Adequately Prepared (1)	Somewhat Prepared (2)	Moderately Prepared (3)	Very Prepared (4)
Differentiate Instruction	0%	28%	35%	38%
Teach ELLs	10%	55%	24%	10%
Generate Student Interest	0%	0%	48%	52%
Reach Underrepresented Minorities	0%	14%	31%	55%
Teach Relevance of CS	0%	3%	31%	66%
Use Guided Inquiry	0%	21%	45%	35%
Ensure Equal Participation	0%	13%	24%	66%
Foster Group Interactions	0%	13%	35%	55%
Teach Introductory Concepts	0%	3%	38%	59%
Utilize CRT/Pedagogies	0%	28%	52%	21%

3.3. Institutional Practices

Participating educators shared their perceptions of challenges to implementing rigorous computer science education at the school level. On average, the greatest challenges shared was rapidly changing technology (with a weighted score of 1.97/3, $SD=.68$), closely followed by a lack of hardware and software resources (1.93, $SD=.70$). Other prominent challenges included difficult subject matter (1.76, $SD=.64$) and a lack of support from the school and staff (1.76, $SD=.64$). *Table 3* further details these responses.

Educators also reflected on student interest and broadening participation in computer science education at their institutions. At the high school level¹, less than 20% of respondents indicated that the demographic composition of their computer science classrooms was identical or nearly identical to the demographic composition of their school. Around 30% indicated that the two demographic groups were slightly different, and 56% indicated that they were moderately different. It is also worth noting that the participants that indicated a (nearly) identical match also reported they teach in districts with relatively homogeneous demographic composition. Further, participants shared that in the last 5 years, the demographic composition of the computer science classroom is becoming more like the demographic composition of the school. All educators responded that their classrooms were moving more towards equal demographic composition, with the most common response being a 7 on a scale of 1 (“much less similar”) to 10 (“much more similar”).

When considering student interest in computer science education, most teachers indicated that there is strong student interest at their school for computer science (with a combined score of 3.79/4, $SD=.94$), but “enrolling in computer science is a student priority” received the lowest responses in the category (2.93, $SD=.75$). Teachers indicated that students taking introductory computer science tend to move to more advanced classes (3.72, $SD=.59$), student dropout of computer science classes is low (3.41, $SD=1.0$), and that there is demand for more computer science classes (3.44, $SD=.95$).

4. Discussion

With the insights from the results discussed above, the research sought to distill actionable, assessed “needs” to inform the path forward in the provision of professional development to K-12 educators surrounding computer science education. These needs, situated with relevant literature, offer suggestions for resource allocation and better articulation of programming to maximize the impact of efforts by organizations in this space.

4.1. Encompassing Need: Expand the Present Understanding of Computer Science

The idea that computer science is synonymous with programming remains a common misconception in computer science education. This pervasive myth is true for many prospective computer science students, who often do not consider that while programming is indeed an important component, professionals in the field also engage in activities that include hardware and software design, research into the impact on society (Barr & Stephenson, 2011; Denning, 2004; Denning et al., 2017; Pardaboyevich et al., 2021) and routine engagement in problem solving (Peckham et al., 2000). Survey results show that the majority of computer science teachers also have a background in teaching mathematics (with a large number currently teaching both subjects), which could contribute to the overrepresentation of these analytical concepts. Further, teachers rated their understanding of content knowledge for Algorithms & Programming most highly out of all content domains, whereas Nature of Technology, Effects of

Table 3. Teachers’ Perceptions of Challenges to Implement CSE (N=29)

	Minor/no Challenge	Moderate Challenge	Great Challenge
Lack of Student Interest	48%	38%	14%
Rapidly Changing Technology	24%	55%	21%
Difficult Subject Matter	28%	59%	14%
Lack of School/Staff Support	35%	55%	10%
Lack of Student Knowledge	35%	55%	10%
Lack of Curriculum Resources	45%	38%	17%
Lack of Software/Hardware	28%	52%	21%

¹ To control for the fact that computer science instruction, when offered at the K-8 level, is often a mandatory elective, only responses that indicated teaching at the 9-12 level were considered for this question.

Computing, and Ethics of Computing (all non-analytic) received the lowest ratings of confidence (Nature of Technology received an average rating of 2.86, which is over 1.5 deviations below average). These results are not unique to this survey; a recent study assessed the perceptions of digital technology teachers attending a computer science workshop in which a majority of teachers surveyed indicated that they felt that computer science is mostly about programming (Prieto-Rodriguez & Berretta, 2014).

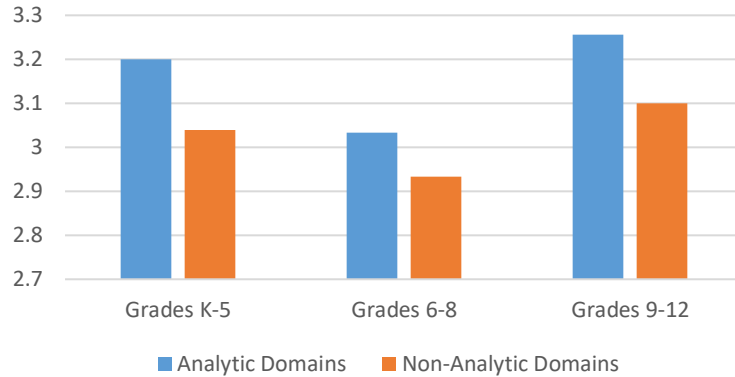


Figure 3. Self-Rating of Content Knowledge Across Grade Band & Analytic v. Non-Analytic Content Domains

As illustrated in *Figure 2*², teachers rated their content knowledge, on average, nearly a full standard deviation

lower in non-analytic domains when compared to analytic ones. This difference is seen in all grade bands, though is most distinct in grades K-5. It is worth noting that teachers indicating they currently teach introduction to computer science rate their understanding of analytic concepts 0.26 points higher than non-analytic – a full deviation above the mean. This is not to say that these teachers do not understand non-analytic concepts but suggest that an overemphasis of more mathematical aspects of computer science can begin at the introductory level, as teachers may feel more comfortable in providing deeper insight, answering questions more completely, or offering more thoughtful discussion in those areas where they have a stronger understanding.

These findings present an urgent need to expand the current understanding of computer science as a discipline, both to students and to educators. Survey results, combined with analyses of current institutional practices, show that computer science is often contained in the mathematics department and that CS courses are classified as mathematics electives (Blitz et al., 2021); this can present computer science to students as a close alternative to math. The overemphasis of analytic and mathematical concepts in computer science education can not only deter students from participating in computer science but has also constructed a false barrier-to-entry for educators, especially in the K-8 spaces, who may be concerned about their own knowledge of the subject or not see the value in introducing computer science in younger grades (Barr & Stephenson, 2011; Denning, 2004; Denning et al., 2017). Professional learning providers therefore must challenge these conceptions of computer science and more broadly and intentionally disseminate information that highlights computer science as a multidisciplinary area of study centered around problem solving, computational thinking, and the creation of technology to meet today’s demands.

4.2 Ancillary Need: Reaffirm Commitment to Equity

Expanding the present understanding of computer science in this way will be a fundamental part of bringing computer science to the K-8 space (and in New Jersey, in the implementation of upcoming standards), and it will also be a powerful tool in moving forward the work of ensuring equitable participation in computer science. It will be important for educators to authentically engage young learners with the full spectrum of computer science to encourage long-term participation. In this way, computer science can ‘start broad’ to invite engagement, rather than starting with narrow specialization such as programming or web design, as is often the case when computer science is first offered at the high school level.

Across the board, teachers felt strongly that they were able to teach the relevance of computer science to their students (with a combined score of 3.62/4, but an incomplete understanding of computer science may be hindering these efforts (especially when considered alongside respondents identifying “lack of student interest” as the most minor challenge to computer science education). Survey results show that over 40% of educators report that their computer science classroom is at least moderately different in demographics from their overall student population; when considering only high school classes, where CS is most likely to be an ‘opt-in’ course, that number increases

² *Figure 2* shows the average scores received from 29 survey participants when asked to rate their level of mastery of content knowledge in the various domains of computer science and design thinking. Content domains were categorized as analytic (engineering design, algorithms and programming, data and analysis, networks and the internet, and computing systems) or non-analytic (nature of technology, effects of computing, ethics of computing, human-computer interaction, and impacts of computing), and ratings were averaged across each category.

to 56%. However, results also show that educators feel strongly that there is student interest in computer science. These results point to a need to reaffirm educators' commitment to equity in CSE and broadening participation in computing; professional development should not only increase awareness of the persistent inequities in the field but equip educators with culturally relevant pedagogies in CS to tackle such disparities.

4.3. Ancillary Need: Equip Educators with Ways to Leverage Existing, Available Technology

Recent years have seen a sharp increase in the availability of “plug and play” curricula, which are implementation-ready programs to bring computing instruction to classrooms. To educators, the ongoing release of new curricular options using rapidly developing computer technologies, although to the same end, can be overwhelming (Pardaboyevich et al., 2021). In fact, rapidly changing technology was cited as the number one challenge to teaching computer science among both new and veteran teachers. Although most of these curricula teach to the same set of standards (those created by CSTA), many do so through different approaches. Some take an approach through physical computing, others through app development, others center around drag-and-drop programming, and so on.

As such, although rapidly changing technology is perceived as a challenge (one that was cited as at least a moderate challenge by around 75% of survey respondents), it may be a manufactured one. To respond, professional learning providers should clearly articulate that although there are many curricular and technological resources available, it is likely the case that only one, if any, will be needed to deliver quality CSE, even as new resources become available. A school's curriculum (whether purchased as a pre-created curriculum or developed) does not need to respond to each release of a new platform; similar research has shown that teachers require professional development that is aligned with their curricular needs (Qian et al., 2018), which does not always call for the use of new, or any, computing devices.

Following closely behind is the lack of software or hardware resources, which was cited as at least a moderate challenge by 73% of survey respondents. As work is done to frame computer science education as a discipline centered around problem solving and computational thinking in addition to programming, it will be necessary to understand that the newest instructional or curricular resources are not always necessary to implement and sustain a rigorous computer science education. Additionally, a promising way to engage students in computational thinking without the need for expensive equipment is through the use of “CS Unplugged” activities (Bell et al., 2009). Although these instructional practices have demonstrated efficacy across the board, they can be particularly helpful in under resourced schools (Bell et al., 2009).

5. Conclusion

As computer science is being recognized as a driving force in our world, a useful lens to understand and process the world around us, and a vehicle for financial growth, computer science education is becoming an increasing part of students' academic experience. This increase is facilitated by efforts such as the implementation of learning standards, increased resource allocation for clubs or extra-curricular activities, and national efforts like the CSforAll program or the nonprofit *CS is Elementary*. However, to ensure that these efforts are fully realizing their potential (and that all students are effectively reached and engaged in the process), it is necessary for educators to have a more complete understanding of computer science, which will increase the efficacy in the adoption of learning standards, promote equity in CSE and broadening participation in computing, and dismantle challenges perceived by CS educators. Professional learning providers, such as regional learning Hubs or curriculum developers, and other advocating agencies must act quickly to fill in gaps in understanding as educational interventions are implemented and scaled.

5.1. Implications for Future Work & Study Limitations

Results from the present study offer a promising direction for the provision of professional learning to educators in computer science education. As providers work to address the research-identified dearth of rigorous professional learning opportunities for computer science teachers (Yadav et al., 2016), it will be crucial to ensure that educators have a complete, nuanced understanding of what computer science is as a discipline. First, teachers' under-confidence in non-analytic content areas of computer science illustrate the need for an intentional emphasis on aspects of CS that extend beyond algorithms and computer programming, such as problem solving, computational thinking, and the social impacts and implications of a technology-driven world. In doing so, professional learning has the potential to address misconceptions about computer science as a discipline and tackle false barriers to entry caused by inaccurate equivalences between CS and mathematics. Further, in broadening the present understanding of computer science, educators, especially in elementary spaces and those where CS instruction is mandatory for all

students, will be better equipped to cast a wider net while delivering CS instruction, effectively broadening the CS pipeline at its early entry-points. Additionally, building awareness of the breadth CS will help address teachers' primary concerns of rapidly changing technology and lack of hardware and software resources by better equipping educators to teach CS in ways that do not solely rely on (ever-changing) computing devices or canned curricula.

Although we make the case that an expanded teacher understanding of the complexities and full spectrum of computer science can – directly or indirectly – address many of the present challenges in CSE, we also acknowledge that our study is exploratory in nature and thus presents a set of limitations. First, we recognize that our survey sample is likely a biased one, as recruitment for survey respondents took place at a computer science education summit. However, given that our sample is likely more CS-inclined, we propose that the challenges distilled from these findings can serve as a lower bound towards generalization. Second, we caution readers to consider our proposed need for an expanded understanding as a backdrop for interpreting survey results.

Finally, as both a limitation of this study and a promising avenue for future work, results illustrate the need for a better understanding of the current CS teacher landscape. Input from various educational stakeholders is necessary in determining the best direction for professional learning, and although this study hopes to provide a solid foundation, we recognize our limited sample and call for future research to validate and expand findings discussed herein. We also acknowledge that this study, along with several points from the survey instrument, are narrowly focused on the current CSE ecosystem in New Jersey and may not be directly generalizable to other geographic regions, though we hope that our approach and findings may be important markers.

Acknowledgements

The authors would like to acknowledge the funding received from the National Science Foundation under grant award 1837305 through the CSforAll RPP Program, which supported, in part, the development and administration of the survey in the above research.

References

- Andrew, F., Mary, W., Margaret, C., Charoula, A., Joyce, M.-S., Joke, V., & Jason, Z. (2016). Arguing for Computer Science in the School Curriculum. *Journal of Educational Technology & Society*, 19(3), 38-46. <http://www.jstor.org/stable/jeductechsoci.19.3.38>
- Banilower, E. R., Smith, P. S., Malzahn, K. A., Plumley, C. L., Gordon, E. M., & Hayes, M. L. (2018). *Report of the 2018 NSSME+*. I. Horizon Research.
- 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>
- Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer Science Unplugged: school students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13.
- Blitz, C., Allen, V., & Amiel, D. (2021). *Recruiting Diverse Learners to High School Computer Science* Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, Virtual Event, USA. <https://doi.org/10.1145/3408877.3439565>
- Bureau of Labor Statistics. (2022). Occupational Outlook Handbook, Computer and Information Research Scientists. In: U.S. Department of Labor.
- Code.org, CSTA, & Alliance, E. (2021). *2021 State of computer science education: Accelerating action through advocacy*. <https://advocacy.code.org/stateofcs>
- Denning, P. J. (2004). The field of programmers myth. *Commun. ACM*, 47(7), 15–20. <https://doi.org/10.1145/1005817.1005836>
- Denning, P. J., Tedre, M., & Yongpradit, P. (2017). Misconceptions about computer science. *Commun. ACM*, 60(3), 31–33. <https://doi.org/10.1145/3041047>
- Department for Education. *Teaching school hubs*. <https://www.gov.uk/guidance/teaching-school-hubs>
- Nager, A., & Atkinson, R. D. (2016). The Case for Improving U.S. Computer Science Education. *SSRN Electronic Journal*. <https://doi.org/https://doi.org/10.2139/ssrn.3066335>
- Pardaboyevich, R. F., Abdunazirovich, U. S., & Saydullayevich, S. Q. (2021). Teaching Computer Science at School - Current Challenges And Prospects. *JournalNX - A Multidisciplinary Peer Reviewed Journal*, 6(11), 217-221. <https://doi.org/https://repo.journalnx.com/index.php/nx/article/view/146>

- Peckham, J., DiPippo, L., Reynolds, J., Paris, J., Monte, P., & Constantinidis, P. (2000). A first course in computer science: the discipline is more than programming. *J. Comput. Sci. Coll.*, 15(5), 220–227.
- Prieto-Rodriguez, E., & Berretta, R. (2014, 22-25 Oct. 2014). Digital technology teachers' perceptions of computer science: It is not all about programming. 2014 IEEE Frontiers in Education Conference (FIE) Proceedings, Qian, Y., Hambrusch, S., Yadav, A., & Gretter, S. (2018). Who Needs What: Recommendations for Designing Effective Online Professional Development for Computer Science Teachers. *Journal of Research on Technology in Education*, 50(2), 164-181. <https://doi.org/10.1080/15391523.2018.1433565>
- Scott, A., Martin, A., McAlear, F., & Koshy, S. (2017). Broadening participation in computing: examining experiences of girls of color. *ACM Inroads*, 8(4), 48–52. <https://doi.org/10.1145/3149921>
- 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 21st century: Why, what and when? *Education and Information Technologies*, 22(2), 445-468. <https://doi.org/10.1007/s10639-016-9493-x>
- Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, 26(4), 235-254. <https://doi.org/10.1080/08993408.2016.1257418>

Appendix A

Selected Survey Questions

1. [Multiple Choice Matrix] How would you rate your level of content knowledge in each of the following areas related to computer science? (*Cols: I have heard of these concepts, I am familiar with these concepts, I understand these concepts, I have mastered these concepts*)

- Computing Systems
- Networks and the Internet
- Impacts of Computing
- Data & Analysis
- Algorithms & Programming
- Engineering Design
- Interaction of Technology and Humans
- Nature of Technology
- Effects of Technology on the Natural World
- Ethics & Culture

2. [Multiple Choice Matrix] How prepared do you feel to do each of the following in your computer science instruction? (*Cols: Not adequately prepared, Somewhat prepared, Moderately prepared, Very prepared*)

- Plan differentiated instruction for your students
- Teach computer science to English language learners
- Encourage students' interest in computer science
- Teach computer science to underrepresented student populations (females, racial or ethnic minorities, students who are economically disadvantaged)
- Teach students the relevance of computer science in their daily lives

- Teach computer science using a guided inquiry approach
- Ensure that every student in the class participates in the learning activities
- Foster group interactions during the learning activities
- Plan and facilitate learning activities focused on introductory computer science concepts
- Utilize culturally relevant pedagogical practices

3. [Multiple Choice Matrix] How would you rate each of the following potential challenges to teaching CS? (*Cols: Minor/No challenge, Moderate challenge, Great challenge*)

- Lack of student interest/enrollment
- Rapidly changing technology
- Difficult subject matter
- Lack of support/interest from school staff
- Lack of student subject knowledge
- Lack of curriculum resources
- Lack of hardware/software resources

4. [Multiple Choice Matrix] How much do you agree or disagree with the following statements? (*Cols: Strongly Disagree, Disagree, Agree, Strongly Agree*)

- There is strong interest from students in the computer science courses offered in my school.
- Enrolling in computer science courses is a top priority for students in my school.
- Students who enroll in introductory computer science courses typically want to take an advanced course.

- Student dropout from computer science courses offered in my school is low.
- There is student demand for more computer science courses in my school.

5. [Multiple Choice] How well does the current demographic composition of your computer science classroom match the demographic composition of your school?

- They are (nearly) identical
- They are slightly different
- They are moderately different
- They are significantly different
- They are dramatically different

6. [Sliding Scale] In the last 5 years, has your classroom demographic makeup become more or less similar to the demographic makeup of your school?

(Scale from 1-10, Labels: 1= Classroom has become much less similar, 5 = No change, 10 = Classroom has become much more simila

