Education and Information Technologies
https://doi.org/10.1007/s10639-022-11508-4

Assessing computational thinking across a STEM curriculum for pre-service teachers

Rachel F. Adler[1] · Joseph Hibdon[1] · Hanna Kim[1] · Scott Mayle[1] · Brittany Pines[1] · Sudha Srinivas[1]

Received: 28 March 2022 / Accepted: 4 December 2022

Abstract
In order to assess computational thinking (CT) modules embedded into multiple STEM courses for educators, we created a CT rubric which incorporates key CT components using Bloom's Taxonomy. We implemented the rubric in four different courses in our pre-service STEM education program for elementary and middle school teachers. We analyzed our rubric results in addition to a pre- and post-survey gauging students' CT skills using the same rubric items. Our rubric results show that students scored well after completing our modules with over 90% in proficiency or high proficiency in all areas in our rubric. We also report on improvements in our CT assignments and scores based on the rubric. In addition, students' self-efficacy in each CT item improved significantly from the beginning to end of the semester. Using the newly developed CT rubric allowed us to assess students' CT skills with a single method across a variety of STEM education courses.

Keywords Computational thinking · Bloom's taxonomy · Assessment · Pre-service teachers · Rubric

1 Introduction

The ubiquity of computing in today's society underscores the argument that computational thinking (CT) should be a core component of education for everyone, not just for future computer scientists (Wing, 2008). While studies have discussed incorporating CT into pre-service teachers' STEM curricula (Adler & Kim, 2018; Jaipal-Jamani & Angeli, 2017; Kim et al., 2018; Yadav et al., 2017) as well as directly into K-12 classrooms (Grover et al., 2015, 2014; Meerbaum-Salant et al., 2010), assessing CT

✉ Rachel F. Adler
r-adler@neiu.edu

[1] Northeastern Illinois University, 5500 N. St Louis Ave, 60625 Chicago, IL, USA

can be challenging (Brennan & Resnick, 2012), particularly that of assessing it in a consistent manner, and across different disciplines.

Another aspect that adds to the challenge of assessing CT is that some researchers assess CT solely through the examination of student projects or quizzes, many of which are based on a single (and specific) programming platform (Boe et al., 2013; Brennan & Resnick, 2012; Grover et al., 2015; Moskal et al., 2004; Werner et al., 2012). Based on previous literature that defined CT and Bloom's Taxonomy to assess educational learning into cognitive levels, we developed a CT rubric, with five main CT criteria, which can be used to assess students' CT growth in different disciplines and with different programming platforms. The rubric was implemented in courses for future K-8 STEM teachers.

Our university has an undergraduate math and science content preparation program for future elementary and middle school science and math teachers. In the last few years, we have added CT and coding into the curriculum through the incorporation of newly developed CT modules which are now integrated into the foundational-level science and math courses in the program as well as the Science Methods course. In addition, we have created a new introductory Computer Science course that will be required for students in this program. After identifying a common set of themes or criteria of CT that we believed our students should become proficient at, we designed a rubric that incorporated enough flexibility so that it would be widely applicable to assess CT in the courses in our teacher preparation program. Our motivation in developing the CT rubric was to create an assessment instrument that could be used to assess the diverse CT modules embedded in the math and science courses, the Science Methods course, and in the newly developed Computer Science course.

In this paper, we first define CT and determine its key components based on a survey of the literature. Next, we describe how others have assessed CT in order to formulate our own chosen method of assessment. Then we discuss Bloom's Taxonomy and describe how it helped shape the creation of our CT rubric. Following this, we describe the implementation of CT in four different courses in our program and results of incorporating the CT rubric into those courses. Finally, we summarize the implications and lessons learned and next steps.

2 Background

2.1 Defining CT

While there have been many attempts to define CT, there is no consensus. In one definition, CT is explained as "the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms" (Aho, 2012). In order to properly assess CT in multiple disciplines it is important to understand the key elements that are included in CT. Rose et al. (2017) examined seven existing definitions of CT (Angeli et al., 2016; Barr & Stephenson, 2011; Brennan & Resnick, 2012; Grover & Pea, 2013; Kalelioglu et al., 2016; Repenning et al., 2016; Seiter & Foreman, 2013) and found that recurring CT concepts include: abstraction and generalization; algorithms and procedures; data collection, analysis, and representation; decomposition; parallelism; debugging, testing, and analysis; and control structures. A CT guide for teachers in the U.K. breaks CT down into five main elements, which include algorithmic thinking, decomposition, generalization/patterns, abstraction, and evaluation (Csizmadia et al., 2015).

The International Society for Technology in Education (ISTE) describe CT as "knowing what steps to take to solve a problem and to apply that skill across disciplines" (Sykora, 2021). The ISTE and the Computer Science Teachers Association (CSTA) developed their own characteristics of CT, which include (ISTE and CSTA, 2011):

- "Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data.
- Representing data through abstractions such as models and simulations.
- Automating solutions through algorithmic thinking (a series of ordered steps).
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources.
- Generalizing and transferring this problem solving process to a wide variety of problems".

Brennan and Resnick's (2012) computational practices include being incremental and iterative, abstracting and modularizing, testing and debugging, or reusing and remixing, while Weintrop et al. (2016) include data practices, modeling and simulation practices, computational problem solving practices, and systems thinking practices in their definition.

Our aim was to incorporate the CT elements, from the works described above, into our pre-service STEM education courses. We argue that looking at the key terms that appear in many definitions of CT can help clarify what CT really is. Based on the literature, we have identified the major components of CT as decomposition, abstraction, models and simulations, analyzing data, algorithmic thinking, finding patterns, and generalization.

Decomposition involves breaking a complex problem into more manageable and solvable components, which can have particular relevance in scientific modeling (Sengupta et al., 2013). *Abstraction* can make problem solving easier by hiding unnecessary details (Csizmadia et al., 2015). When scientific concepts are *modeled* with a computer, students can repeatedly conduct experiments as well as visualize the *data* (Ornek, 2008).

The concept of *algorithmic thinking* is at the heart of CT. Ultimately, every multi-step task, whether scientific or non-scientific, needs to be broken down into a series of discrete steps through which the task is accomplished (Csizmadia et al., 2015). Solutions for one problem may contain *patterns* and similarities with others, thereby allowing one to *generalize* a problem to a different one (Csizmadia et al., 2015).

2.2 Assessing CT

To determine whether the integration of CT into courses is successful, it is important to assess the growth of students perceived and actual CT skills. While there is a growing emphasis on incorporating CT and coding into K-12 classrooms, properly assessing CT is quite difficult (Alves et al., 2019).

Manually assessing assignments may work well for an individual course, but there is no consistent, objective standard across different instructors and courses which is present when using automated assessment tools (Ala-Mutka, 2005; Von Wangenheim et al., 2018). Grading Scratch projects is more time-consuming than text-based programming which can often be graded using a script (Boe et al., 2013). Some studies have used assessment tools which evaluate a specific block-based programming platform. For example, to assess CT in after-school and elective technology classes for middle school, Werner et al., (2012) used Alice, a visual programming environment. Using the Fairy Assessment, students needed to modify code to solve tasks which related to two components of CT: thinking algorithmically and abstraction and modeling. Their results showed this to be a promising form of assessment. This assessment, however, is specific to the Alice programming environment. Scratch, another block-based programming language, also has assessment tools, such as Dr. Scratch, Hairball, and Scrape, which assess work after students submit their code. Brennan & Resnick (2012) used Scrape, which, based on Scratch's color coding of blocks, determines which blocks were used more frequently. While Scrape can check how many loops there are and whether nesting of loops occurs in a program, Hairball can also answer questions related to whether programs have unmatched broadcast/receive blocks, result in infinite loops, and do not initialize values properly (Boe et al., 2013). Dr. Scratch extends Hairball and Scrape to provide a free user-friendly web application for teachers which assigns a CT score based on abstraction, logical thinking, synchronization, parallelism, flow control, user interactivity, and data representation (Moreno-León & Robles, 2015). While these tools are useful to assess one programming platform, they are limited to that programming platform and would not be applicable to other programming environments.

Other studies used coding-specific assessments, with quizzes and tests using Scratch code (Grover et al., 2015; Grover & Basu, 2017). However, while that is useful for assessment of CT in a particular course, they are not reusable across other platforms as they are tied to specific programming languages (Basu et al., 2021). Tang et al. (2020) has called for more CT assessments that are applicable across platforms.

Using open-ended questionnaires (Yadav et al., 2014, 2018), problem-solving scenarios (Dagiene & Futschek, 2008; Snow et al., 2017), or multiple choice style quizzes (Chen et al., 2017; Román-González et al., 2017) to assess CT skills are useful in that they are not dependent on specific platforms and can test students' CT abilities (Basu et al., 2021). However, they would not be suitable for grading a specific coding assignment or project.

Rubrics can be an effective way to assess CT across multiple platforms and courses in a consistent manner (Bort & Brylow, 2013; Cateté et al., 2016; Grover et al., 2018; Seiter & Foreman, 2013; Sherman & Martin, 2015), though they can be more time-consuming to grade (Rochford & Borchert, 2011). Rubrics are effective in measuring

Table 1 CT Assessment Tools and Features

	Assess CT Skills	Assess Student Coding Assignment	Consistent Across Different Programming Environments
Manual Program Analysis	Yes	Yes	No
Program Assessment Tool	Yes	Yes	No
Coding-Specific Assessment	Yes	No	No
General CT Quiz	Yes	No	Yes
CT Rubric	Yes	Yes	Yes

many components of a topic and can include specific standards which leads to consistent CT scores across various assignments and instructors (Docktor et al., 2016). This can minimize some of the subjectivity that arises when multiple instructors are looking at similar skills. In addition, rubrics can be used effectively to evaluate criteria even when handed to someone who is not an expert in that field (Becker, 2003).

Table 1 summarizes the above CT assessment methods with benefits and drawbacks of each assessment method. While interviewing students about their final projects and code can assess students' CT and programming knowledge (Brennan & Resnick, 2012; Grover et al., 2014), in practice, it is time-consuming and often not feasible or scalable to interview every student in order to determine their CT skills (Basu et al., 2021). In fact in some studies only a sample of students in a course were interviewed (Bagley & Rabin, 2016; Cetin & Andrews-Larson, 2016).

While many studies have used one form of assessment, others have found that, rather than relying solely on one approach, a combination of approaches is preferable. Brennan & Resnick (2012) used three methods for assessment: Scrape to assess Scratch projects, interviews regarding the Scratch projects they created, and the use of design scenarios where students need to understand and remix an existing project. Grover et al. (2014) used multiple methods to assess Scratch programming through quizzes which include Scratch code snippets, pre- and post-tests, Scratch assignments, and final project interviews.

2.3 Developing a CT rubric with Bloom's Taxonomy Framework

While there are benefits and drawbacks to the different methods of assessments, for our purposes a rubric was the best option to examine CT skills across different courses and programming environments. To determine appropriate items for our rubric we used Bloom's Taxonomy as our framework. Researchers have used Bloom's Taxonomy in relation to programming and CT (Gouws et al., 2013; Lister & Leaney, 2003; Meerbaum-Salant et al., 2010; Selby, 2015). Bloom's Taxonomy is a tool often used for assessment through rubrics (Crowe et al., 2008).

Bloom's Taxonomy was developed to help articulate definitions and classifications of terms in education such as "thinking" and "problem solving" (Bloom et al., 1956). This framework helps determine whether students learned what was expected of them (Bloom et al., 1956). In 2001, an updated version of Bloom's Taxonomy was created using more verbs as terms to describe the categories and their subcategories

Table 2 Revised Bloom's Taxonomy (Anderson & Krathwohl, 2001)

Revised Bloom's Taxonomy	Explanation	Subcategories
Remember	Retrieve relevant knowledge from long-term memory	Recognizing Recalling
Understand	Construct meaning from instructional messages, including oral, written, and graphic communication	Interpreting Exemplifying Classifying Summarizing Inferring Comparing Explaining
Apply	Carry out or use a procedure in a given situation	Executing Implementing
Analyze	Break material into constituent parts and determine how parts relate to one another and to an overall structure or purpose	Differentiating Organizing Attributing
Evaluate	Make judgements based on criteria and standards	Checking Critiquing
Create	Put elements together to form a coherent or functional whole; reorganize elements into a new pattern or structure	Generating Planning Producing

(Anderson & Krathwohl, 2001), see Table 2. Note that in the revised Bloom's Taxonomy, "create" is listed as the highest order in the taxonomy, though in the original the highest was "evaluation". We use evaluate as the highest in the taxonomy in our rubric.

When designing the rubric, we did not use the lowest level (remember), since we were testing students' ability to incorporate and apply CT. While "remember" could have worked well for our Computer Science course, in the other three courses these modules were embedded into the content and our focus was on having the students use CT, without necessarily recalling facts for each of those courses. Table 3 shows our CT rubric. We limited it to five main CT elements: Decomposition, Variable manipulation, Analysis of program output or data, Algorithmic thinking, and Generalizability to other problems and areas, based upon discussion of CT in the literature (ISTE and CSTA, 2011; Sengupta et al., 2013; Weintrop et al., 2016; Wing, 2008). We removed jargon that could confuse instructors who may not be proficient in technical CT terms.

While focusing on characteristics of our assignments, we tried to ensure that these categories would be applicable for CT assignments in other topics and disciplines. In order to score well on the CT rubric, students need to decompose a problem into its components, use and modify variables or parameters in programs and simulations, explain program output and/or data, develop a series of steps to create or modify code, and describe broader applications of the program for other topics or disciplines.

In Table 4 we show how each item in our rubric corresponds with the higher order elements in Bloom's Taxonomy as well as the CT skills used for each item.

Below we outline each of the CT criteria and how they relate to one of the levels in the taxonomy.

Table 3 CT Rubric

\ Quality Criteria \	No/Limited Proficiency (1 pt.)	Some Proficiency (2 pts.)	Proficiency (3 pts.)	High Proficiency (4 pts.)
Deconstruct a problem into smaller, more manageable parts	Not able to break down the problem	Recognizes some parts of the problem but unable to identify key contributing components	Identifies the key components that contribute to the problem	Identifies the key components that contribute to the problem and describes their significance
Ability to manipulate variables/ parameters for desired result	Unable to differentiate between different variables/ parameters or determine their effect upon the model	Understands significance of variables/parameters, but cannot properly change them to cause variations in the model	Has good grasp of significance of variables/parameters; is able to manipulate models accordingly after initial instruction	Has excellent grasp of variables/parameters; is able to change models correctly with little guidance
Analyze and interpret program output or data	Unable to describe program output or data	Describes program output or data, but incorrectly interprets the meaning	Correctly interprets the meaning of the program output or data	Correctly interprets the meaning of the program output or data and draws conclusions within context of the program's limitations
Use algorithmic thinking to modify or construct computer code	Cannot develop steps to modify or construct computer code	Begins to develop steps to modify or construct computer code, but some steps are missing or not in a logical order	Series of steps to modify or construct computer code is complete and in logical order	Series of steps to modify or construct computer code is complete, in logical order, and efficient
Generalize to another problem or real-world situation	Cannot describe how the program could relate to another problem or situation	Can relate the program to another problem or situation, but cannot identify underlying pattern(s)	Identifies pattern(s) in the problem solving process and relates the pattern(s) to another problem or situation	Identifies and analyzes pattern(s) in the problem solving process and can justify generalization to another problem or situation

Table 4 CT Rubric Items with Corresponding Revised Bloom's Taxonomy Elements

CT Criteria	Revised Bloom's Taxonomy	CT Skills
Deconstruct a problem into smaller, more manageable parts	Understand	Abstraction; Decomposition
Ability to manipulate variables/ parameters for desired result	Apply	Models and Simulations
Analyze and interpret program output or data	Analyze	Data analysis
Use algorithmic thinking to modify or construct computer code	Create	Algorithmic Thinking
Generalize to another problem or real-world situation	Evaluate	Generalization; Finding Patterns

2.3.1 Deconstruct a problem into smaller, more manageable parts (***decomposition***)

Our first level follows the Revised Bloom's *understand* level. At this level, we ask our students to identify the key components of a problem and to explain their significance. In order to deconstruct a problem into its components, students need to use CT techniques such as abstraction and decomposition. This moves students beyond the ability to recall information, instead requiring that they understand how to classify and explain why they chose these components.

2.3.2 Ability to manipulate variables/parameters for desired result (***variables***)

Next, we expect our students to reach the *apply* level in the Revised Bloom's Taxonomy. In order for students to be able to manipulate parameters and variables, they will need to use the information they have learned and execute problems through the manipulation of parameters and variables.

2.3.3 Analyze and interpret program output or data (***data***)

This item corresponds to the *analyze* level in the Revised Bloom's Taxonomy. At this level we ask our students to make a connection between what they did and the results they see. They will need to differentiate what is relevant to them. Depending on the model, results may either be in the form of program output or data.

2.3.4 Use algorithmic thinking to modify or construct computer code (***algorithmic thinking***)

Create is one of the higher levels of the Revised Bloom's Taxonomy. At this point, students are asked to construct and produce something new. This may involve adding or modifying code to form a new creation.

2.3.5 Generalize to another problem or real-world situation (***generalization***)

Our last level corresponds to the *evaluate* level in the Revised Bloom's Taxonomy. At this level we want students to defend and justify how they could generalize their solution in order for it to be transferable to problems in other disciplines. They need to check what is consistent between the different applications. To do this, they need to find patterns that are similar to other problems or situations.

Note that the same CT skills may appear in more than one element of the rubric. While we distinguish them in different places, many of the CT skills are relevant in multiple places and recur at multiple levels. For example, abstraction is listed under deconstructing a problem into smaller problems but it can also be used when generalizing to another problem and when creating unique code or algorithms.

In order to test this rubric, we implemented it in assignments across multiple courses and disciplines: Physics, Mathematics, Science Methods, and a Computer Science course for pre-service teachers.

Table 5 Participants

	Fall 2018	Spring 2019	Fall 2019
Geometry Concepts for Educators	5	2	5
Physics Concepts for Educators	8	3	4
Science Methods	16	18	20
Computer Science for All	16	13	15
Total	**45**	**36**	**44**

3 Research Methods

In order to assess students' CT skills, we used a combination of methods: (1) a CT rubric and (2) pre- and post-surveys containing items similar to our rubric gauging participants' views on their CT skills. The CT rubric allowed us to analyze student artifacts and answers to assignment questions while also comparing and contrasting the CT skills across disciplines using different programming platforms. Examining artifacts after the fact is not enough to gauge students' understanding of the code (Brennan & Resnick, 2012). Therefore, in addition to using submitted programming projects, our courses incorporated answers to questions relating to the development of code, or classroom observation and instructor notes, into the CT rubric scores.

Pre- and post-survey questions match our rubric criteria in order to gauge students' perceptions of their CT knowledge and to determine whether they felt their CT skills increased. We also asked open-ended questions regarding their CT skills and ability to use it in their future classrooms. Approval for this research was obtained at our university's Institutional Review Board, and we received informed consent from the students.

3.1 Participants

All participants were registered for one of four courses (Physics, Geometry, Science Methods, Computer Science) at an urban public university in the Midwest. In total, data for 125 students (80 female, 45 male) was analyzed using the CT rubric (45 in Fall 2018, 36 in Spring 2019, and 44 in Fall 2019; see Table 5). Students taking multiple courses in the same semester are included separately for each course.

3.2 CT tasks

In order to examine the validity of our rubric we tested it on four courses that future middle school math and science teachers enroll in: Physics, Geometry, Science Methods, and an introductory Computer Science course. The first three courses were modified to include CT/coding modules, while the Computer Science course was new and designed to provide education students with a strong foundation in CT and coding.

3.2.1 Geometry concepts for educators

Modules were developed for the *Geometry Concepts for Educators* course to reinforce students' skills by coding geometrical concepts in Scratch. In addition, a final project had students: (a) choose a piece of art that is culturally or personally relevant to them and explore the geometry concepts in that work of art, (b) abstract the artwork and geometry concepts using Scratch, and (c) collaborate with a classmate to merge their artworks into a collaborative work of art. At the end of the course, students presented a poster of their individual and combined artworks (Fig. 1). Using the CT rubric, the instructor graded their final projects.

3.2.2 Physics concepts for educators

The *Physics Concepts for Educators* course used VPython, which operates the Python programming language using 3D graphics. We used trinket [https://trinket.io/], a web-based coding environment, so that students did not need to install software and could run code from any browser or device. Physics concepts such as the notion of vectors and vector addition, kinematics, and conservation of mechanical energy, were selected for further illustration through embedding computational simulations. In one project, we focused on assessing CT in a vector addition simulation. This represents one of the first exercises in this course involving the distinctions between distance and displacement, and likewise between speed and velocity, physical concepts that students have difficulty differentiating. In order to learn about displacement and velocity vectors, students applied vectors in a real-world setting by placing the displacement vectors on top of a scaled map of the U.S. (Fig. 2). The simulation provided students with the components of the vector, allowing students to analyze and extract data from the map, while also providing a direct visual of the extracted information. The simulation was designed to allow students to import a map of their own choice so that they could utilize vectors within a personally relevant setting. One example of a student-selected assignment consisted of using a Marauder's Map of Hogwarts and its surroundings from Harry Potter. Through the simulation the student was able to visualize the net displacements as one travelled from Hogwarts to the Whomping Willow, Forbidden Forest, Hagrid's hut, and finally when returning back to Hogwarts. In addition to using the simulation in various travel configurations, students were asked to learn about the code and make small edits to it, from adding comments to modifying actual code and parameters in it.

3.2.3 Science methods

In the *Science Methods* course, students learned about earthquakes, such as their causes and underlying mechanisms, and earthquake-related structural stability issues. Students constructed earthquake simulator robots (Fig. 3) using LEGO Mindstorms EV3 kits with custom-made building instructions. Students then programmed a robot, and by using a loop and variables they were able to see the power of the motor controlling the robot increase in intensity from 1 to 5, thus simulating the magnitude of an earthquake. By the end of the simulation, students had designed and built their own unique buildings using LEGO pieces in order to test if their structures were strong enough to withstand earthquakes of different magnitudes. This inquiry-based approach allowed pre-service teachers to generalize the problem to a real-world situ-

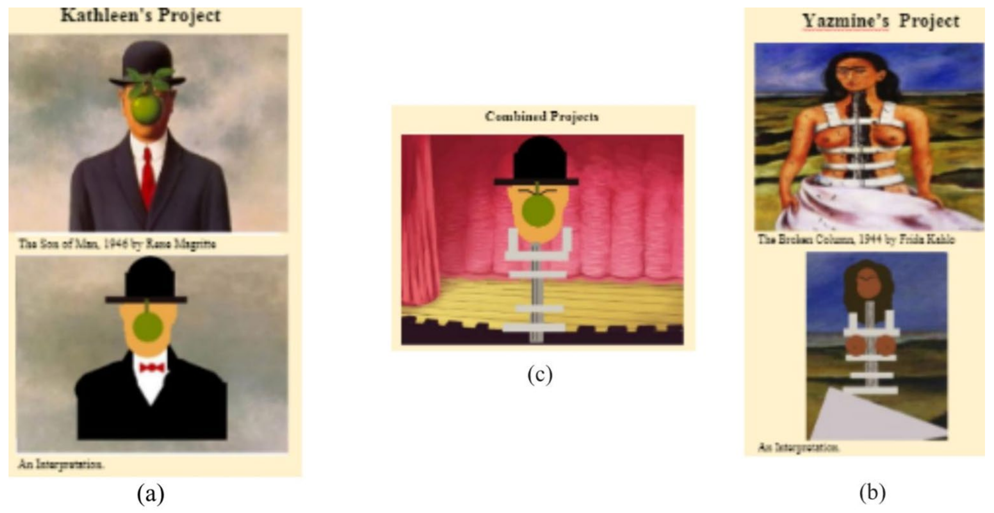


Fig. 1 Sample group geometry project showcasing students' abstracted art interpretations: (a) Kathleen's Scratch Project where the bottom represents an interpretation in Scratch of "The Son of Man" by René Magritte shown above it; (b) Yazmine's project, where the bottom represents an interpretation of "The Broken Column" by Frida Kahlo shown above it; (c) The students' combined Scratch project, merging two pieces of different artwork together

ation, finding solutions based on the module, and allowing them to learn how to guide their future pupils in developing CT skills using robots.

3.2.4 Computer science for all

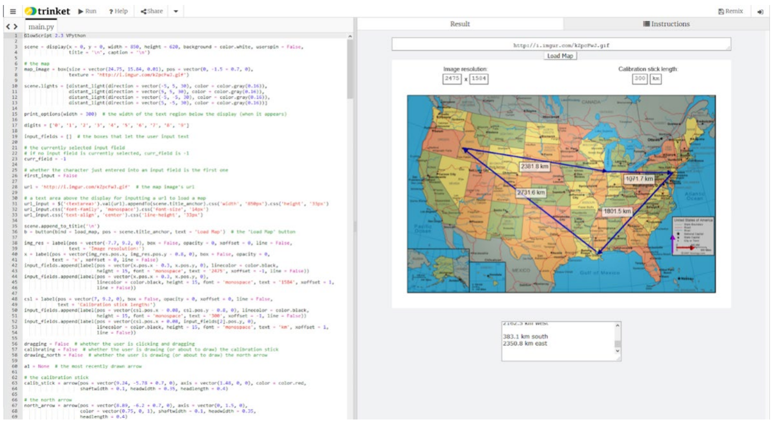
The *Computer Science for All* course was designed to provide students with a foundation in Computer Science. The course primarily focuses on two visual-programming platforms: Scratch and Lego Mindstorms EV3 Robotics. This introductory Computer Science course is an important addition to our pre-service curriculum because it will help improve their CS skills so they can later use them in their future STEM and Science Methods classrooms. The rubric was used for the final project at the end of the semester. For the project, students had to choose one of the two visual programming environments, then delve into a topic of their choice and present it to the class. One group created a math pong game targeted for third grade, in which each time a ball hits one of the colored blocks on top, the user needs to answer a different math problem "hidden" underneath (Fig. 4).

4 Results

4.1 Analysis of rubric results

In order to measure the reliability of our CT rubric we had two independent evaluators complete the Fall 2018 rubrics for each course. Due to the small sample size for Geometry and Physics, interrater reliability results were only calculated for Science Methods and Computer Science for All. Evaluators were given student submissions and the assignment and rated each of five rubric items (decomposition, variables, data, algorithmic thinking, generalization) based on 1="No/Limited Proficiency" 2="Some Proficiency" 3="Proficiency" 4="High Proficiency" for a given student. The evaluators independently rated two projects from each course with the rubric, discussed disagreements, repeated the process, and then independently calculated ratings for ten submissions from each course. The percentage of agreement from data from these two courses was 91%, and a weighted Cohen's Kappa yielded a reliability of 90%. (Note that for the Science Methods course, while evaluators rated decomposition, variables, data, and algorithmic thinking, generalization was omitted by the evaluators as this was observed during class discussions by the instructor and the raters did not have access to that data.) For Computer Science for All, all five categories were rated for each student, as, in addition to their code, students submitted a written assignment answering questions based on each rubric item (see Appendix A). We also examined reliability by calculating the Cronbach's alpha of the five CT rubric items which was reliable at 0.80.

To address the construct validity of the rubric, we consulted experts who had a background in CT and/or CS pedagogy (face validity) and sought their feedback on each of the individual CT rubric rows and columns. The first implementation of the rubric during the Spring 2018 semester was treated as a pilot during which we sought



```
GlowScript 2.3 VPython

scene = display(x = 0, y = 0, width = 850, height = 620, background = color.white, userspin = False,
                title = '\n', caption = '\n')

# the map
map_image = box(size = vector(24.75, 15.84, 0.01), pos = vector(0, -1.5 + 0.7, 0),
                texture = 'http://i.imgur.com/k2pcFwJ.gif')

scene.lights = [distant_light(direction = vector(-5, 5, 30), color = color.gray(0.16)),
                distant_light(direction = vector(5, 5, 30), color = color.gray(0.16)),
                distant_light(direction = vector(-5, -5, 30), color = color.gray(0.16)),
                distant_light(direction = vector(5, -5, 30), color = color.gray(0.16))]

print_options(width = 300)  # the width of the text region below the display (when it appears)

digits = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

input_fields = []  # the boxes that let the user input text

# the currently selected input field
# if no input field is currently selected, curr_field is -1
curr_field = -1

# whether the character just entered into an input field is the first one
first_input = False

url = 'http://i.imgur.com/k2pcFwJ.gif'  # the map image's url

# a text area above the display for inputting a url to load a map
url_input = $('<textarea>').val(url).appendTo(scene.title_anchor).css('width', '850px').css('height', '33px')
url_input.css('font-family', 'monospace').css('font-size', '14px')
url_input.css('text-align', 'center').css('line-height', '33px')

scene.append_to_title('\n')
b = button(bind = load_map, pos = scene.title_anchor, text = 'Load Map')  # the 'Load Map' button

img_res = label(pos = vector(-7.7, 9.2, 0), box = False, opacity = 0, xoffset = 0, line = False,
                text = 'Image resolution:')
x = label(pos = vector(img_res.pos.x, img_res.pos.y - 0.8, 0), box = False, opacity = 0,
          text = 'x', xoffset = 0, line = False)
input_fields.append(label(pos = vector(x.pos.x - 0.3, x.pos.y, 0), linecolor = color.black,
                          height = 15, font = 'monospace', text = '2475', xoffset = -1, line = False))
input_fields.append(label(pos = vector(x.pos.x + 0.3, x.pos.y, 0),
                          linecolor = color.black, height = 15, font = 'monospace', text = '1584', xoffset = 1,
                          line = False))

csl = label(pos = vector(7, 9.2, 0), box = False, opacity = 0, xoffset = 0, line = False,
            text = 'Calibration stick length:')
input_fields.append(label(pos = vector(csl.pos.x - 0.08, csl.pos.y - 0.8, 0), linecolor = color.black,
                          height = 15, font = 'monospace', text = '300', xoffset = -1, line = False))
input_fields.append(label(pos = vector(csl.pos.x + 0.08, input_fields[2].pos.y, 0),
                          linecolor = color.black, height = 15, font = 'monospace', text = 'km', xoffset = 1,
                          line = False))

dragging = False  # whether the user is clicking and dragging
calibrating = False  # whether the user is drawing (or about to draw) the calibration stick
drawing_north = False  # whether the user is drawing (or about to draw) the north arrow

a1 = None  # the most recently drawn arrow

# the calibration stick
calib_stick = arrow(pos = vector(9.24, -5.78 + 0.7, 0), axis = vector(1.48, 0, 0), color = color.red,
                    shaftwidth = 0.1, headwidth = 0.35, headlength = 0.4)

# the north arrow
north_arrow = arrow(pos = vector(8.89, -6.2 + 0.7, 0), axis = vector(0, 1.5, 0),
                    color = vector(0.75, 0, 1), shaftwidth = 0.1, headwidth = 0.35,
                    headlength = 0.4)
```

```
383.1 km south
2350.8 km east
```

Fig. 2 Vector Simulation using a USA Map

Fig. 3 Robotic Earthquake Simulator

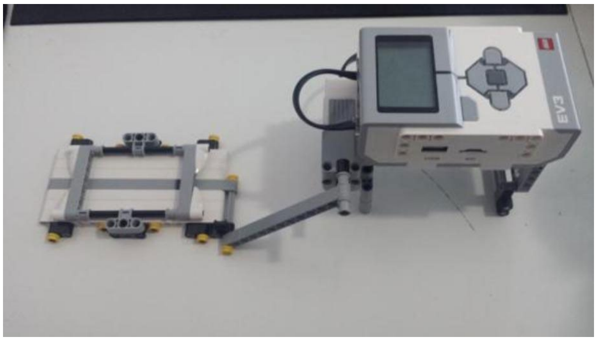
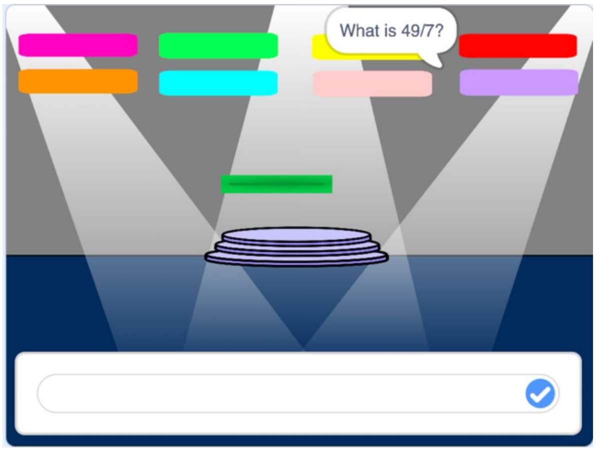


Fig. 4 Sample Project of a Math Pong Game



additional feedback from the instructors teaching the courses and modified the rubric accordingly. Our results report on the new data beginning Fall 2018.

Table 6 Frequency Distribution for CT Rubric Items from Fall 2018, Spring 2019, and Fall 2019 (n=125)

Criteria	No/Limited Proficiency (%)	Some Proficiency (%)	Proficiency (%)	High Proficiency (%)
Decomposition	0 (0%)	8 (6%)	38 (30%)	79 (63%)
Variables	1 (1%)	4 (3%)	46 (37%)	73 (59%)
Data	2 (2%)	4 (3%)	53 (42%)	66 (53%)
Algorithmic Thinking	1 (1%)	11 (9%)	27 (22%)	86 (69%)
Generalization	3 (2%)	4 (3%)	62 (50%)	56 (45%)

Table 7 Frequency Distribution for CT Rubric Items for Fall 2018 (n=45)

Criteria	No/Limited Proficiency (%)	Some Proficiency (%)	Proficiency (%)	High Proficiency (%)
Decomposition	0 (0%)	5 (11%)	25 (56%)	15 (33%)
Variables	1 (2%)	3 (7%)	26 (58%)	15 (33%)
Data	0 (0%)	2 (4%)	25 (56%)	18 (40%)
Algorithmic Thinking	1 (2%)	6 (13%)	20 (44%)	18 (40%)
Generalization	2 (4%)	4 (9%)	13 (29%)	26 (58%)

In order to assess CT proficiency in the students taking the courses, we examined the CT rubric scores completed by instructors in the four courses. In general, the results were high (Table 6). Over 90% of students exhibited proficiency or high proficiency in all five categories.

While not all of our instructors made strict use of the CT scoring to calculate students' grades for the assignments, the rubric helped them measure the CT strengths and weaknesses of their students in the given assignment. Similar to Bloom's Taxonomy where students master lower levels before going on to the higher levels, our rubric's higher levels require prerequisite knowledge from the lower levels. For example, there were eight students who had only some proficiency in decomposition (a lower level in the taxonomy) and these students generally performed poorly in the higher levels (mean total rubric score for these students was 12 out of 20 compared to a mean of 17 for all students). In fact, all but one student who ranked in the lowest category (no/limited proficiency) on the other rubric items were from these eight students. If students were not able to proficiently decompose a problem into its components, they would be very unlikely to complete the higher levels of the rubric successfully.

Furthermore, we noticed significant improvement in CT scores for students from Fall 2018 (our first semester implementing the rubric) compared to later semesters. Using the rubric enabled instructors to notice where their students were lacking and address that in subsequent semesters. A Kruskal-Wallis non-parametric test showed significant improvement in total CT scores χ^2 (df=2)=14.04 (p=.0009) from Fall 2018 (mean=16 out of 20) to Spring 2019 (mean=18 out of 20) and Fall 2019 (mean=18 out of 20). When examining the Fall 2018 data separately (Table 7), many of the lowest proficiency scores were from that semester.

Table 8 Correlation between CT Rubric Items

	Decomposition	Variables	Data	Algorithmic Thinking	Generalization
Decomposition	1	0.707*	0.428*	0.738*	0.142
Variables		1	0.412*	0.582*	-0.003
Data			1	0.418*	0.355*
Algorithmic Thinking				1	0.092
Generalization					1

*p<.0001

Spearman's correlation was also computed to assess the relationship between each of the rubric items. Table 8 shows a significant and positive correlation between most of our CT criteria. In most cases, students' scores in one CT criteria, correlated with the other CT areas. The exception was generalization, which was not significantly correlated in most cases.

4.2 Analysis of surveys

We conducted pre- and post-surveys for all of the students in the four courses for three semesters. Students in multiple courses in a given semester only completed the survey once. A total of 80 students (46 female and 34 male) completed both pre- and post-semester surveys (28 in Fall 2018, 30 in Spring 2019, and 21 in Fall 2019).

Using a 5-point Likert scale (1 = Strongly Disagree; 5 = Strongly Agree), we measured students' perception of their CT skills through items derived from our CT rubric (Table 9). Since our data was non-parametric, we used a two-tailed Wilcoxon signed-rank test, with $\alpha = 0.05$, in order to determine whether there were positive or negative changes. We found that students' overall perception of their CT skills increased from the beginning to end of the semester. In addition, our survey results revealed significant gains for students in all individual CT components. Participants felt they had gained in breaking down complex problems (decomposition), manipulating variables (variables), creating and modifying computer code (algorithmic thinking), analyzing program output and data (data), and applying what they learned to other problems/disciplines (generalization). Although our data showed significant increases in these items, it is important to examine whether that was practically significant. While generalization had a medium, or moderate, effect size according to Cohen's classification of effect sizes which is 0.10 - < 0.3 (small effect), 0.30 - < 0.5 (medium effect) and >=0.5 (large effect) (Cohen, 1988, 1992), there were high effect sizes for all other CT items. A power analysis post hoc using G*Power 3.1 (Faul et al., 2009) showed that for our medium effects size (0.44), a sample size of 80, and $\alpha = 0.05$, the power is 0.9668, and for the highest effect size of 0.78 power was 0.9999991.

Our survey also asked students how they would incorporate CT into their future teaching. Their answers demonstrated that they were planning on integrating CT into their future courses. Some examples include:

"This could be great for when I am trying to teach a complicated topic and I can break it up into much smaller parts for the students to understand it." **(Decomposition)**.

"having students create models, and have them predict outcomes based on changing certain variables." **(Algorithmic Thinking; Data; Variables)**.

Table 9 Pre- and Post-test Survey Statements (n=80[a])

Category	Statement	*Mean (S. D.)*		Wilcoxon signed-rank test	Effect Size
		Pre-test	**Post-test**		
General CT					
CT	I am confident in my ability to use computational thinking to understand or analyze problems.	3.41 (1.10)	4.11 (0.88)	V=162***	0.66
Specific CT Category					
Decomposition	I am able to break a complex problem into smaller, more manageable parts or components so that it can be solved using a computer.	3.49 (1.06)	4.18 (0.83)	V=218***	0.62
Variables	I am able to manipulate a system's variables or components to achieve a desired result.	3.48 (1.13)	4.06 (0.88)	V=168.5**	0.57
Data	I can analyze or interpret a program's output or data.	2.93 (1.17)	3.93 (0.96)	V=218***	0.70
Algorithmic Thinking	I am able to modify existing computer code to complete small tasks in subject areas I am familiar with.	2.74 (1.25)	3.88 (0.95)	V=195***	0.72
	I am able to create computer code to complete small tasks in subject areas I am familiar with.	2.67 (1.17)	3.95 (0.97)	V=121.5***	0.78
Generalization	I am able to apply what I've learned to another problem or discipline.	3.89 (0.89)	4.19 (0.78)	V=239*	0.44

*p<.01; **p<.001; ***p<.0001; [a]Due to student omissions in some pre- or post-survey items n=77 for variables and n=79 for creating code and overall confidence in CT.

"A couple of Scratch programs can be used when I teach my students certain formulas like the Pythagorean Theorem." **(Generalization)**.

"I will definitely incorporate Scratch in my future teaching. You are able to use Scratch for basically any subject. I would have students create games on certain things we are learning in class or I will create a game for them to play and then leave a part for them to remix or debug. Using Scratch for any core subject allows students to understand that computational thinking can be applied to anything." **(Algorithmic Thinking; Generalization)**.

5 Discussion

Our aim was to create a single CT assessment tool that could be used by instructors in multiple courses who use different programming environments. We created a CT rubric based on components derived from the literature (Brennan & Resnick, 2012; Csizmadia et al., 2015; ISTE and CSTA, 2011; Rose et al., 2017; Weintrop et al., 2016) and integrated it with higher orders of Bloom's Taxonomy (Anderson & Krathwohl, 2001; Bloom et al., 1956). Using Bloom's Taxonomy enabled us to define our CT items in terms of how students learn, focusing on students' ability to understand, apply, analyze, create, and evaluate. We implemented our CT rubric in four different courses in a pre-service education program.

Many researchers argue that multiple forms of assessment are ideal (Boe et al., 2013; Brennan & Resnick, 2012; Grover et al., 2014). Therefore, in addition to using our rubric to assess CT skills in our students, we also surveyed the students to measure the self-efficacy of students' CT skills. Similar to Weese and Feldhausen (2017) who created a self-efficacy survey to gather students' perceptions on their CT abilities, our CT survey measures students self-efficacy in CT and maps to each of our five CT rubric items. Our survey results showed that students felt they had significant gains in CT by the end of the semester. This is consistent with CT rubric scores which showed a significant percentage of students in the proficiency/high proficiency range for each CT item.

We also found that our students were positive about their experiences in our updated courses and had many ideas on how to integrate CT into their future classrooms.

In our post-survey, one student who took the Computer Science for All course wrote *"This semester has made me grow in computational thinking more than ever. Continuing what I learned is the key going forward. Everything else is on me, the school did its job."* In our technological society, having computing incorporated into teacher education programs is vital for the future generations of children to have the opportunity to see and use coding and CT.

5.1 Implications and Lessons learned

Below we discuss implications and lessons we learned.

5.1.1 Having a rubric led to better assignments

Similar to Crowe et al. (2008, p. 373), who found that their rubric was leading them "to ask and write better questions," using the CT rubric helped improve CT assignments provided to our students. For example, during the implementation of CT in our pilot year, the geometry CT module consisted of students creating shapes with Scratch. However, the module did not consider a key CT criterion, generalization. Students were not shown how what they learned could be generalized to other real-world problems or disciplines. After the creation of the rubric, we added a real-world component in which students could see how learning how to draw shapes in Scratch can be taken to the next level through exploring patterns found in a different discipline, art. This implementation was superior, as students learned not only how to integrate geometry and Scratch, but also how the CT skills they learned could be applied in more than one discipline. In the post-survey, one student wrote that he might incorporate CT into his own future class by doing a similar project in his classroom.

Furthermore, in order to determine whether students would be able to generalize what they learned to another discipline or real-world problem, students in the Geometry course also had to submit answers to questions such as:

1) How would you apply this to other disciplines?
2) What other discipline could you incorporate into this project (besides art)?
3) Are there ways to connect this project to other courses you have taken or what you plan to take in the future?

4) How would you incorporate this project or ideas from this project in your future classroom?

Taking CT one step further by not only using it, but evaluating how it could be used in the future is now included as an important part of the module.

In the Physics course, students used VPython simulations as part of our agenda to incorporate CT into the curriculum. However, after the creation of the rubric, the assignment was modified so that students were also required to make modifications to the Python code, thereby improving algorithmic thinking skills. In their post-survey responses on how they would integrate CT into their own classrooms, students now reported including coding and more hands-on activities. Furthermore, to provide students with more coding experience in a text-based platform, we implemented additional assignments. In one, students are given a VPython simulation, which drew vectors on the screen, displaying the sum of the vectors, but now they needed to include the code to add the vectors together. This provided them with more experience using loops and variables, as well as an opportunity to better understand existing code and how to contribute to it in a more significant way.

5.1.2 More than one data source to determine rubric scores

We found that the programming projects were not enough to determine rubric scores. In addition to students' submitting their artifacts, an important factor was the inclusion of additional methods for assessment. To help determine the rubric scores, many instructors included individual assignment questions asking students to submit answers pertaining to each of the five rubric components in the module. (See Appendix A for an example from the Computer Science course.) Since most of the rubric modules were group work, this allowed instructors to determine individual student scores for each of the five rubric items.

Another method used by some of the instructors in their scoring included relying on classroom observation and their notes on class activities, particularly when grading the algorithmic thinking item. For example, in Science Methods and Physics, although student programs and output from all groups were similar, instructors also based some of their calculations on class observation and video when determining students' algorithmic thinking scores. Instructors determined in class whether students were able to break the problem into small steps and use algorithmic thinking as they were doing the exercises.

Lastly, while many of the rubric items were able to be determined based on project artifacts, one area that most relied on other sources was *generalization*, which is not displayable directly by code, but mostly rated either through the individual assignment questions or through classroom observation/discussion. For example, to examine generalization in Computer Science for All (Appendix A), question 5 asked students how they could use what they learned to solve a different problem.

5.1.3 Integrating CT for their own students

Overall, the integration of CT into the courses was positively received by the students. While our students were eager to use CT in the future, in some cases they may have misunderstood the goal by thinking they should use programming or CT to teach a concept rather than having their students use it. For example, as shown in the quote in Sect. 4.2, when asked about how they would incorporate CT into their future teaching, one student wrote "This could be great for when I am trying to teach a complicated topic and I can break it up into much smaller parts for the students to understand it." Rather than describing how their future students would decompose a problem into its parts, this student described breaking up a problem for her students. Therefore, it is important to ensure that future teachers understand the real motivation is for providing opportunities for their students to use CT themselves.

5.2 Limitations and future work

While the CT rubric was implemented in four different courses, it was only used at one university, and two out of four of the courses had small class sizes. Nevertheless, the four instructors found the rubric helpful and it guided them in creating assignments that covered the key CT components. While some instructors used additional factors to complete the rubric, such as classroom observation and answers to questions, having the rubric helped guide assignment questions and ensure students were able to acquire the CT skills on the rubric. Future research can compare our students' CT skills at the beginning of the program to the end of the program to determine gains in CT not only for one semester but as a program.

6 Conclusion

Incorporating a CT module into a course for future teachers to improve their CT skills is adequate, but not sufficient. Pre-service teacher programs can incorporate CT into multiple courses to be really successful in training future teachers with CT. We modified our pre-service STEM curriculum for future elementary and middle school teachers to include CT and coding in content courses, the Science Methods course, and a Computer Science course for educators. We created a rubric, based on Bloom's Taxonomy, to assess CT and used it in four courses successfully. Using the rubric enabled us to learn how our students' CT skills were growing and where there was room for improvement, such as having more hands-on coding opportunities. The creation of a CT rubric can allow instructors in different courses and disciplines to have a single systematic method for assessment of CT.

7 Appendix A

Sample Individual Project Questions from the Computer Science Course.

1) What was the general problem or topic you were addressing? What are the key components of that problem? What is the significance of the key component(s)? Describe the code that corresponds to each of the key components?
2) What variables did you use in your project? Explain how those variables were used and why it was necessary to use variables. How could you modify your project by changing the variables?
3) Explain the output of your program. What is the significance of the results of your code? What are the limitations of your code or the data you collected?
4) Describe the part of the code that you created? (Not your team member's portion of code). What component in the overall code does the part that you created solve or address? How did you construct the code? What logical operators did you use to make your code efficient?
5) Your project used Scratch or Robotics, do you have ideas for a project using the same tool you chose but to address a different real world problem and/or a different discipline. How does this new problem relate or compare to the original problem? How would you justify the results and the importance of using this tool?

Acknowledgements We would like to thank Jennifer Slate and Durene Wheeler for their contributions to the creation of the CT rubric. This material is based upon work supported by the National Science Foundation under Grant No. DRL-1640041. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Conflict of interest None.

Data Availability The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

References

Adler, R. F., & Kim, H. (2018). Enhancing future K-8 teachers' computational thinking skills through modeling and simulations. *Education and Information Technologies*, *23*(4), https://doi.org/10.1007/s10639-017-9675-1

Aho, A. V. (2012). Computation and computational thinking. *Computer Journal*, *55*(7), 832–835. https://doi.org/10.1093/comjnl/bxs074

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, *15*(2), 83–102.

Alves, N. D. C., Von Wangenheim, C. G., & 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.

Anderson, L. W., & Krathwohl, D. R. (2001). *A taxonomy for learning, teaching, and assessing: a revision of Bloom's taxonomy of educational objectives*. Longman.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking Curriculum Framework: implications for teacher knowledge. *Educational Technology & Society*, *19*(3), 47–57.

Bagley, S., & Rabin, J. M. (2016). Students' Use of Computational thinking in Linear Algebra. *International Journal of Research in Undergraduate Mathematics Education*, *2*(1), 83–104. https://doi.org/10.1007/s40753-015-0022-x

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

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*, *0*(0), 1–30. https://doi.org/10.1080/08993408.2020.1866939

Becker, K. (2003). Grading programming assignments using rubrics. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, 253.

Bloom, B. S., Englehart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). *Taxonomy of Educational Objectives, Handbook I: the cognitive domain*. David McKay Company.

Boe, B., Hill, C., Len, M., Dreschler, G., Conrad, P., & Franklin, D. (2013). Hairball: lint-inspired static analysis of scratch projects. In *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 215–220). ACM. https://doi.org/10.1145/2445196.2445265

Bort, H., & Brylow, D. (2013). CS4Impact: Measuring Computational Thinking Concepts Present in CS4HS Participant Lesson Plans. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 427–432. https://doi.org/10.1145/2445196.2445323

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *American Education Researcher Association*. https://dam-prod.media.mit.edu/x/files/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Cateté, V., Snider, E., & Barnes, T. (2016). Developing a Rubric for a Creative CS Principles Lab. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, 290–295. https://doi.org/10.1145/2899415.2899449

Cetin, I., & Andrews-Larson, C. (2016). Learning sorting algorithms through visualization construction. *Computer Science Education*, *26*(1), 27–43. https://doi.org/10.1080/08993408.2016.1160664

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, *109*, 162–175. https://doi.org/10.1016/j.compedu.2017.03.001

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences*. Routledge.

Cohen, J. (1992). A power primer. *Psychological Bulletin*, *112*(1), 155–159.

Crowe, A., Dirks, C., & Wenderoth, M. P. (2008). Biology in Bloom: implementing Bloom's taxonomy to Enhance Student Learning in Biology. *CBE-Life Sciences Education*, *7*(4), 368–381.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational thinking: a guide for teachers*.

Dagiene, V., & Futschek, G. (2008). Bebras International Contest on Informatics and Computer literacy: Criteria for Good Tasks. In R. T. Mittermeir, & M. M. Sysło (Eds.), *Informatics Education - supporting computational thinking* (pp. 19–30). Berlin Heidelberg: Springer.

Docktor, J. L., Dornfeld, J., Frodermann, E., Heller, K., Hsu, L., Jackson, K., Mason, A., Ryan, X., Q., & Yang, J. (2016). Assessing student written problem solutions: A problem-solving rubric with application to introductory physics. *Physical Review Physics Education Research*, *12*. https://doi.org/10.1103/PhysRevPhysEducRes.12.010130

Faul, F., Erdfelder, E., Buchner, A., & Lang, A. G. (2009). Statistical power analyses using G*Power 3.1: tests for correlation and regression analyses. *Behavior Research Methods*, *41*, 1149–1160.

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013). Computational thinking in educational activities: an evaluation of the educational game light-bot. *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, 10–15. https://doi.org/10.1145/2462476.2466518

Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 267–272. https://doi.org/10.1145/3017680.3017723

Grover, S., Basu, S., & Schank, P. (2018). What We Can Learn About Student Learning From Open-Ended Programming Projects in Middle School Computer Science. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 999–1004. https://doi.org/10.1145/3159450.3159522

Grover, S., Cooper, S., & Pea, R. (2014). Assessing computational learning in K-12. *Proceedings of the 2014 Conference on Innovation Technology in Computer Science Education*, 57–62. https://doi.org/10.1145/2591708.2591713

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

ISTE and CSTA (2011). *Operational definition of computational thinking for K-12 Education*. http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf?sfvrsn=2

Jaipal-Jamani, K., & Angeli, C. (2017). Effect of Robotics on Elementary Preservice Teachers' Self-Efficacy, Science Learning, and computational thinking. *Journal of Science Education and Technology*, *26*(2), 175–192. https://doi.org/10.1007/s10956-016-9663-z

Kalelioglu, F., Gulbahar, Y., & Kukul, V. (2016). A Framework for Computational thinking based on a systematic Research Review. *Baltic Journal of Modern Computing*, *4*, 583–596.

Kim, C., Yuan, J., Vasconcelos, L., Shin, M., & Hill, R. B. (2018). Debugging during block-based programming. *Instructional Science*, *46*(5), 767–787. https://doi.org/10.1007/s11251-018-9453-5

Lister, R., & Leaney, J. (2003). Introductory programming, criterion-referencing, and bloom. *ACM Sigcse Bulletin*, *35*, 143–147. https://doi.org/10.1145/792548.611954

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2010). Learning computer science concepts with scratch. In *Proceedings of the Sixth international workshop on Computing education research* (pp. 69–76). ACM. https://doi.org/10.1145/1839594.1839607

Moreno-León, J., & Robles, G. (2015). Dr. Scratch: A Web Tool to Automatically Evaluate Scratch Projects. *Proceedings of the Workshop in Primary and Secondary Computing Education*, 132–133. https://doi.org/10.1145/2818314.2818338

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education* (pp. 75–79). ACM. https://doi.org/10.1145/971300.971328

Ornek, F. (2008). Models in Science Education: applications of Models in Learning and Teaching Science. *International Journal of Environmental & Science Education*, *3*(2), 35–45.

Repenning, A., Basawapatna, A., & Escherle, N. (2016). Computational thinking tools. *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 218–222.

Rochford, L., & Borchert, P. S. (2011). Assessing higher level learning: developing rubrics for case analysis. *Journal of Education for Business*, *86*(5), 258–265.

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior*, *72*, 678–691. https://doi.org/10.1016/J.CHB.2016.08.047

Rose, S., Habgood, J., & Jay, T. (2017). An exploration of the role of Visual Programming Tools in the development of Young Children's computational thinking. *Electronic Journal of E-Learning*, *15*(4), 297–309.

Seiter, L., & Foreman, B. (2013). Modeling the Learning Progressions of Computational Thinking of Primary Grade Students. *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, 59–66. https://doi.org/10.1145/2493394.2493403

Selby, C. C. (2015). Relationships: computational thinking, pedagogy of programming, and Bloom's Taxonomy. *Proceedings of the Workshop in Primary and Secondary Computing Education*, 80–87. https://doi.org/10.1145/2818314.2818315

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

Sherman, M., & Martin, F. (2015). The Assessment of Mobile Computational thinking. *J Comput Sci Coll*, *30*(6), 53–59.

Snow, E., Rutstein, D., Bienkowski, M., & Xu, Y. (2017). Principled Assessment of Student Learning in High School Computer Science. *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 209–216. https://doi.org/10.1145/3105726.3106186

Sykora, C. (2021). *Computational Thinking for All*. ISTE. https://www.iste.org/explore/computational-thinking/computational-thinking-all

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

Von Wangenheim, C. G., Hauck, J. C. R., Demetrio, M. F., Pelle, R., da Cruz Alves, N., Barbosa, H., & Azevedo, L. F. (2018). CodeMaster–Automatic Assessment and Grading of App Inventor and snap! Programs. *Informatics in Education*, *17*(1), 117–150.

Weese, J., & Feldhausen, R. (2017, June 1). STEM Outreach: Assessing Computational Thinking and Problem Solving. *2017 American Society for Engineering Education Annual Conference & Exposition (ASEE)*.

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

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 215–220). ACM. https://doi.org/10.1145/2157136.2157200

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical Physical and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Yadav, A., Krist, C., Good, J., & Caeli, E. N. (2018). Computational thinking in elementary classrooms: measuring teacher understanding of computational ideas for teaching science. *Computer Science Education*, *28*(4), 371–400. https://doi.org/10.1080/08993408.2018.1560550

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in Elementary and secondary teacher education. *ACM Transactions on Computing Education*, *14*(1), 1–16. https://doi.org/10.1145/2576872

Yadav, A., Stephenson, C., & Hong, H. (2017). Computational thinking for Teacher Education. *Communications of the ACM*, *60*(4), 55–62.