Journal of Science Education and Technology (2022) 31:428–441
https://doi.org/10.1007/s10956-022-09965-0

Algorithmic Explanations: an Unplugged Instructional Approach to Integrate Science and Computational Thinking

Amanda Peel[1] · Troy D. Sadler[2] · Patricia Friedrichsen[3]

Accepted: 25 April 2022 / Published online: 5 May 2022

Abstract
Computing has become essential in modern-day problem-solving, making computational literacy necessary for practicing scientists and engineers. However, K–12 science education has not reflected this computational shift. Integrating computational thinking (CT) into core science courses is an avenue that can build computational literacies in all students. Integrating CT and science involves using computational tools and methods (including programming) to understand scientific phenomena and solve science-based problems. Integrating CT and science is gaining traction, but widespread implementation is still quite limited. Several barriers have limited the integration and implementation of CT in K–12 science education. Most teachers lack experience with computer science, computing, programming, and CT and therefore are ill-prepared to integrate CT into science courses, leading to low self-efficacy and low confidence in integrating CT. This theoretical paper introduces a novel instructional approach for integrating disciplinary science education with CT using unplugged (computer-free) activities. We have grounded our approach in common computational thinking in STEM frameworks but translate this work into an accessible pedagogical strategy. We begin with an overview and critique of current approaches that integrate CT and science. Next, we introduce the *Computational Thinking through Algorithmic Explanations* (CT-AE) instructional approach. We then explain how CT-AE is informed by constructionist writing-to-learn science theory. Based on a pilot implementation with student learning outcomes, we discuss connections to existing literature and future directions.

Keywords Computational thinking · Science education · Unplugged · Instructional approach

This theoretical paper introduces a novel instructional approach for integrating disciplinary science education with computational thinking (CT) using unplugged (computer-free) activities. We have grounded our approach in common computational thinking in STEM frameworks (e.g., Weintrop et al., 2016), but translate this work into an accessible pedagogical strategy. We begin with an overview and critique of current approaches that integrate CT and science. Next, we introduce the *Computational Thinking through Algorithmic Explanations* (CT-AE) instructional approach. We then explain how CT-AE is informed by constructionist writing-to-learn science theory. Based on a pilot implementation with student learning outcomes, we discuss connections to existing literature and future directions.

✉ Amanda Peel
amanda.peel@northwestern.edu

[1] Learning Sciences Department, Northwestern University, Evanston, IL, USA

[2] Culture, Curriculum and Teacher Education Department, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA

[3] Learning, Teaching, and Curriculum Department, University of Missouri, Columbia, MO, USA

CT and Science Integration

Computing has become essential in modern-day problem-solving, making computational literacy necessary for practicing scientists and engineers (diSessa, 2001). For example, the rapid development of COVID-19 vaccines was possible due to computational models that read the virus genetic code and predicted specific parts of the virus that would be best targets for immune response (Arnold, 2020). These targets were then used to create and test vaccines, a step that would have taken years without computational immunology. Similarly, computation has led to innovations in other STEM disciplines, such as the first image of a black hole (Event Horizon Telescope Collaboration, 2019) and the detection of gravitational waves (Abbott et al., 2016). However, K–12 science education has not reflected this computational shift. To address this, calls from around the world to integrate computing in compulsory schooling have emerged (Bocconi et al., 2016; Heintz et al., 2016). While some educational systems have begun offering computer science courses, these courses are rarely required, resulting in a small number of students building computational literacy (Bocconi et al., 2016; Heintz et al., 2016). Conversely, integrating CT into core science courses is an avenue that can build computational literacies in all students (Grover & Pea, 2013; Wilkerson-Jerde et al., 2015; Wing, 2006). Broadly, CT "is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (Wing, 2011, p. 1). Integrating CT and science involves using computational tools and methods (including programming) to understand scientific phenomena and solve science-based problems. Integrating CT and science is gaining traction, but widespread implementation is still quite limited.

In the USA, CT is included in the Next Generation Science Standards (NGSS) (NGSS Lead States, 2013), which have been adopted in varying forms in 44 states. The NGSS includes "using mathematics and computational thinking," as one of eight science and engineering practices that characterize scientific inquiry and promote learning about science and how science is conducted (National Research Council, 2012). NGSS standards and appendices aim to provide teachers with information about science practices; however, CT is not well defined and NGSS provides little guidance on how to achieve this integration and how to meaningfully scaffold its progression across grade bands. The standards provide performance expectations that describe what students should be able to do, and performance expectations containing CT are underrepresented compared to the other practices. Although the practice progressions in Appendix F of the NGSS include data analysis with digital tools and algorithms in middle school, there are no performance expectations that engage students in CT practices, making it unlikely that CT will be addressed in middle schools or elementary schools. Moreover, because CT is combined with mathematics, most of the associated performance expectations are connected only to mathematics. While NGSS-aligned teaching and learning has been widely implemented with other practices, CT integration has lagged.

To refine the application of computational tools and methods for science classrooms, Weintrop et al. (2016) defined CT in science and mathematics with a taxonomy of practices. This taxonomy is highly cited in the CT and science integration literature. The taxonomy presents science and mathematics specific CT practices that students should be familiar with by the end of their K–12 education. Taxa include data practices, modeling and simulation practices, computational problem-solving practices, and systems thinking practices. While these practices do align with science and mathematics learning, they focus on creating and using computational tools, resulting in computer- and programming-heavy implementations. For example, the modeling and simulation practices prescribe use of computational models to understand a concept and find and test solutions, and assessing, designing, and constructing computational models. This conceptual framing is a key step in supporting CT and science integration; however, translating the taxonomy practices into classroom instruction remains challenging because it requires computers and prior CT or computer science experience.

Several barriers have limited the integration and implementation of CT in K–12 science education. Most teachers lack experience with computer science, computing, programming, and CT and therefore are ill-prepared to integrate CT into science courses (Aljowaed & Alebaikan, 2018; Sands et al., 2018; Yadav et al., 2011, 2014; Wu et al., 2018). This inexperience leads to low self-efficacy and low confidence among teachers for integrating CT (Aljowaed et al., 2018; Rich et al., 2020). Many teachers believe CT is the use of programming and computing, which leads to discomfort with its integration into their science content (Peel et al., 2020; Yadav et al., 2014). A recent survey of 123 K–12 science teachers (Kite & Park, 2020) showed only 24% of the surveyed teachers had accurate conceptions of CT as conceptualized by Fraillon (2018). Teachers reported the following barriers to integrating CT and science: *I don't understand CT* (35.34%), *My students aren't academically prepared for CT-infused lessons* (15.26%), *I don't understand how CT aligns with my content* (14.46%), *I don't have access to the necessary technology* (11.65%), *I don't have room in my curriculum* (10.84%), *I'm not comfortable with technology* (5.62%), and *Lack of administration support* (2.81%). These reported barriers indicate that teachers need support and resources for CT and science integration that help them understand CT, align with their students' preparation, aligns with their content and existing curriculum, and does not require technology.

The CT integration approach presented in this paper is one possible way to help teachers reduce these barriers. We argue the unplugged approach not only eliminates technology and teachers' technological discomfort, but also provides an entry point for student learning and engagement in CT that may better align with students' academic preparation. We also argue that implementing the unplugged approach is easier for teachers when compared to programming and technology CT activities because teachers do not have to learn programming or a new technology. Rather, teachers can learn about CT imbedded in their content and curricula. As with any innovation for teaching, adopting the approach advocated in this paper will require that teachers consider new ideas and integrate new practices within their repertoires which will require some additional training and access to resources. However, this approach helps to meet many science teachers where they are and presents substantially fewer barriers for the integration of CT in their classes.

Most emerging CT integration efforts, including the Weintrop et al. (2016) taxonomy, involve the use of computers (Hsu et al., 2018; Lockwood & Mooney, 2017). Disparities in access to computers among schools with different resources hinder the implementation of CT lessons and units. Gaps in access to computers and computing resources are still widespread in school systems across the US, and while many schools have computer labs or laptop carts, most science classrooms are still not outfitted with computers for all students (Hohlfeld et al., 2017). To reduce the technological barrier to CT integration, more approaches should support CT engagement without the use of computers.

There is a clear need for instructional approaches and examples that integrate CT and science (Li et al., 2020). These approaches and examples need to address barriers to CT integration. This paper draws on conceptual and theoretical frameworks to present an instructional approach to integrate unplugged, or computer free, CT and science. We take the position that unplugged CT can lessen barriers to CT integration and is an easier and more accessible entry point for teachers and students who are intimidated by computing. Moreover, we argue unplugged CT can support science learning in new ways. This paper will introduce the instructional approach and results of an early implementation.

CT-AE

We propose *Computational Thinking through Algorithmic Explanations* (CT-AE) as an instructional approach that engages students in sense-making about science processes through the creation of unplugged algorithmic explanations. Teachers can use CT-AE in their classrooms to synergistically engage students in science and CT as students use CT to make sense of and learn complex scientific processes. Central to CT-AE is student creation of hand-written algorithms, or sequences of steps, that explain a scientific process as they investigate and learn about the process. For example, students can make sense of protein synthesis by creating algorithms that explain transcription, translation, and protein folding processes. CT-AE utilizes algorithm creation as an explanation of a scientific phenomenon, thus shifting CT from technology-dependent applications to science learning and sense-making applications. We argue the combination of sense-making about science through CT makes the integration of CT into science classrooms more approachable to teachers.

CT-AE foregrounds two dimensions of CT: algorithm concepts and CT practices. This approach allows students to learn how to use algorithm concepts and employ CT practices. Algorithm concepts (Table 1) include branching (i.e., if/then/else conditional statements), iteration (i.e., loops and repeating steps), methods (i.e., encapsulated sequence of steps), and variables (i.e., values that can change). These concepts are commonly referenced in CT literature (Peel et al., 2015; Peel & Friedrichsen, 2018; Brennan & Resnick, 2012; Grover et al., 2019; Lye & Koh, 2014). CT practices are actions, or things students do during algorithm creation and other computational activities, whereas concepts are represented as written aspects of an algorithm. Figure 1 presents an example algorithmic explanation of protein translation.

CT practices are sequencing steps, abstracting information, generalizing, recognizing patterns, decomposing processes, and evaluating algorithms (Table 2). When students create algorithmic explanations, they are engaging in CT practices by using algorithm concepts, so correctly using algorithm concepts indicates students can engage in CT practices. For example, using a *branching* statement (algorithm concept) involves *sequencing* events and steps (CT practice) based on a specific condition: if the condition is true, then something happens, else something different happens. Using *iteration* (concept) requires *evaluating* (practice) a sequence of steps, *recognizing patterns* (practice), and identifying where an algorithm can be simplified with a loop (concept). Writing a *method* (concept) to be used in many algorithms requires students to *decompose* (practice) the science process, *generalize* (practice) the steps for use across contexts, and *abstract*

Table 1 CT-AE algorithm concepts

Algorithm concept	Example
Branching—choosing a path, If/Then/Else conditional statements	If organism has the favorable trait, then it is more likely to survive
Iteration—repeating a sequence of steps until a condition is met	Repeat the process of natural selection for every new selection pressure
Method—an encapsulated sequence of steps used in multiple processes	The process of reproduction is a method that can be called in several biological processes
Variable—a value that can change	The selection pressure is a variable, and its value is set to the specific pressure being explained (e.g., drought, hunting, antibiotics)

```
1. If 5’ cap is recognized, then ribosome assembles and attaches, or
   ribosome doesn’t assemble
2. Ribosome reads mRNA without adding tRNA until it reaches start
   codon AUG
3. *
4. Ribosome detaches, polypeptide chain detaches and folds

*
1. tRNA enters through A site according to “rule 1”
2. Drops off amino acid at P site
3. tRNA leaves E site
4. Repeat * unit reaches stop codon

Rule 1
Codon = codon that is in A site
Anticodon = anticodon on tRNA in A site
If the anticodon and codon are complimentary, then the tRNA carrying
the amino acid will enter.
```

Fig. 1 Example of a translation algorithmic explanation

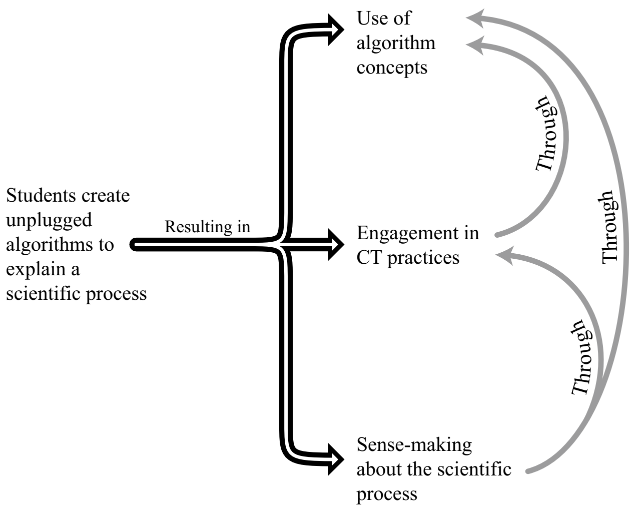

Students create unplugged algorithms to explain a scientific process
Resulting in
Use of algorithm concepts
Engagement in CT practices
Sense-making about the scientific process
Through
Through
Through

Fig. 2 CT-AE instructional approach

(practice) important information. Using *variables* (concept) in an algorithm requires students to *evaluate* (practice) their algorithm and identify where variables can be used to make their algorithm more efficient, or to *generalize* (practice) their algorithm for use in multiple contexts.

The CT literature typically positions the following as CT practices: abstraction, generalization, decomposition, algorithmic thinking, and debugging (Angeli, et al., 2016; Bocconi et al., 2016; Caeli & Yadav, 2020; Csizmadia & Boulton, 2017; Csizmadia et al., 2015; Kalelioglu et al., 2016). We position algorithmic thinking as a higher leverage practice than the others listed here because it requires students to engage in other CT practices: abstraction, generalization, decomposition, evaluation, sequencing, and pattern recognition. Using algorithmic design as the overarching practice allows students to engage with both dimensions of CT-AE at once (Fig. 2). While algorithms are typically defined as “a method to solve a problem that consists of exactly defined instructions” (Futschek, 2006), CT-AE shifts the purpose of algorithms and defines algorithms as explanations of science processes.

In presenting the CT-AE approach, we are not arguing that algorithm creation is the only CT students should engage in. Rather, CT-AE provides an approach that focuses on one aspect of CT, algorithm practices, as a starting point for meaningful integration of CT into science learning opportunities. The approach is an easier introduction to CT for many science teachers and students, particularly those with limited experience and/or access to technology. Furthermore, the approach can provide a foundation for subsequent engagement in other CT practices that require technology such as programming, computational modeling, and computational data practices. Framing scientific explanations with algorithm structure and CT practices will require new classroom practices and instruction, but we argue this approach leverages existing logic and reframes it explicitly as computational. For example, people commonly use “if” statements in everyday language to make decisions, but when students write an algorithmic explanation with if/then/else statements, the colloquial phrase is reframed as computational logic that provides structure (branching) to the explanation. We also recognize the approach is best

Table 2 CT-AE practices

CT practices	Example of connections to algorithm concepts
Sequencing events and steps	Process of writing an algorithm; writing a branching statement to sequence events based on a condition
Abstracting information	Simplifying information with an *iteration* or *method*
Generalizing steps	Using *variables* to make an algorithm useful in multiple contexts
Recognizing patterns	Identifying *methods* that are present in multiple algorithms; identifying similar concepts between algorithms that can be assigned as *variables*; identifying repeating steps where an *iteration* can be used
Decomposing complex processes	Breaking the process into steps of an algorithm and *methods* that can be written
Evaluating algorithms—efficiency, correctness, testing, debugging	Identifying when an *iteration*, *branch*, *method*, or *variable* can be used to simplify the algorithm or make the algorithm clearer and more correct

suited for scientific processes and may not be as effective for explaining scientific structures, mechanisms, or other states.

Conceptual and Theoretical Framing

Several resources informed development of CT-AE as an instructional approach. The following sections describe a CT conceptual framework and the theoretical foundations that frame CT-AE.

CT Conceptual Framework

A revised version of the CT in science and mathematics taxonomy (Weintrop et al., 2016) has been released, and it encompasses a wider range of CT in science and mathematics practices (Peel et al., 2021a). The newer taxonomy is divided into six categories of CT in science and mathematics practices. This revised taxonomy includes new practice categories of algorithms, programming, and computational visualization in addition to computational modeling and simulation, computational data, and computational problem-solving. CT-AE translates this conceptual CT framework into an actionable instructional framework that can be used in classrooms without computer access by focusing on algorithm practices as a subset of CT in science and mathematics practices. Focusing on one category of CT practices allows CT-AE to translate the CT in science and mathematics taxonomy into classroom practice, providing an innovative approach to CT integration. There are five algorithm subpractices in the new taxonomy: (1) using an algorithm to solve a problem or understand a phenomenon, (2) selecting an appropriate algorithm to solve a problem, (3) assessing algorithms, (4) modifying an algorithm to better address a problem, and (5) designing and constructing algorithms (Peel et al., 2021a).

The CT-AE instructional approach engages students in designing and constructing algorithms (#5) as they create hand-written algorithmic explanations of science processes. Once created, algorithmic explanations can be leveraged to engage students in other algorithm practices. Students can iteratively revise their algorithms as they engage in classroom investigations, which allows them to assess (#3) and modify algorithms (#4). As students gain more evidence, they can assess their algorithms for accuracy and clarity, which leads to modifying the algorithm to better explain the science process. Similarly, as students become more familiar with algorithm concepts, they can assess their algorithms in terms of correct and efficient use of algorithm concepts and how algorithm concepts can be better used to explain the science process. The algorithmic explanations can then be utilized to solve problems related to the science process or to explain the process in new contexts.

We argue the unplugged creation of algorithms can also enhance programming and modeling practices described in the revised taxonomy (Peel et al., 2021a). Programming is included as one of the practices, and engaging students in unplugged CT can support algorithm logic and creation, which is essential to programming. If students do not understand the function and importance of algorithm concepts, such as loops, variables, conditionals, and procedures, they will struggle to program correctly and efficiently. We argue that supporting these competencies in students' common language will help them apply algorithm logic and creation across programming languages and in their everyday lives. Programming practices can then be applied to modify and create computational artifacts, which connects to each of the remaining taxonomy practices.

CT-AE aligns with computational modeling and simulation practices described in the revised taxonomy, specifically designing and constructing computational models. Modeling is central to scientific inquiry through mental modeling and conceptual modeling which provide explanatory and predictive power for understanding science phenomena and testing hypotheses (National Research Council, 2012). Modeling has representational and epistemic affordances, which can be leveraged to engage students in sense-making about science phenomena (National Research Council, 2012; Wilensky & Reisman, 2006). Computational modeling has supported science learning and CT learning (Arastoopour Irgens et al., 2020; Aslan et al., 2020; Buckley et al., 2004; Klopfer, 2003; Schwarz et al., 2007; White & Frederiksen, 1998; Wilkerson-Jerde e t al., 2015). In addition to using models, the modification and creation of scientific models support deeper learning of the science phenomenon (Sengupta et al., 2013; Wilensky & Reisman, 2006). However, the creation of computational models can be complex and time consuming, and many teachers are uncomfortable with and unprepared to integrate computational model creation.

We propose that CT-AE can bridge conceptual models and computational models by providing an intermediate representational and epistemological step between hand-drawn model explanations and computational models. Hand-written algorithmic explanations provide an opportunity to translate drawn images and mental representations into written sequenced steps. The algorithm concepts provide a structure for writing by supporting the recognition of patterns, cause and effect relationships, and sequencing of events. Once students and teachers are comfortable with unplugged CT, they can use these competencies to scaffold programming and creating computational models in varying languages and environments. It is difficult to create computational models, even within modeling environments designed to support novice programmers, such as NetLogo (Wilensky, 1999, 2001). In addition to programming issues (e.g., Spohrer, 1989), science phenomena are often complex.

For example, to create a computational model of natural selection, students must reason about and represent other ecological phenomena, such as reproduction, connections between multiple organisms and their environment, aging, and death (Wilensky & Resnick, 1999). Students often struggle to understand the phenomena leading to challenges in attempts to represent them computationally (Forrester, 1994; Wilensky, 2001).

For example, students' challenges with creating and testing computational models have been categorized into four categories: domain knowledge challenges, modeling and simulation challenges, agent-based thinking challenges, and programming challenges (Basu et al., 2016). Students struggled with missing and incorrect science conceptions about the phenomenon they were modeling. Challenges with modeling and simulation included the following: "(1) challenges in identifying relevant entities and their interactions; (2) challenges in choosing correct preconditions; (3) systematicity challenges; (4) challenges with specifying model parameters and component behaviors; and (5) model verification challenges" (Basu et al., 2016, p19). Students also showed issues with agent-based modeling, such as struggling with how to represent individual behaviors and connecting individual actions to aggregate trends and behaviors. Programming challenges included:

> (1) challenges in understanding the semantics of domain-specific primitives; (2) challenges in using computational primitives like variables, conditionals, nesting, and loops to build programs (i.e., behaviors); (3) procedurality challenges; (4) modularity challenges; (5) code reuse challenges; and (6) debugging challenges. (Basu et al., 2016, p20)

Without explicitly considering components, interactions, sequences, and algorithmic logic, students may struggle to create computational models. Thus, engaging in algorithmic explanations may help students succeed in computational model creation. To scaffold computational modeling practices with algorithmic explanations, students can begin by using a computational model to explore a science process. Students then can write an algorithmic explanation of the model with the following question: what rules do the individuals follow as the model runs? Next students can look at the model's code and identify algorithm concepts, such as methods and branching. This connection builds on students' understandings of algorithm concepts to help them read and understand the model's code. Reading and understanding code can then be expanded into programming and model creation. Students can modify pieces of code and see how it impacts the model. Students can write new pieces of code (e.g., add a variable or branching statement). Modifying computational models allows students to better represent the science process and their understandings of that process. This transition and expansion of CT practices aided by algorithmic explanations shift students from consumers of computational models to creators of computational models.

For example, if students are learning about predator and prey relationships, they can use the "Wolf Sheep Predation" NetLogo computational model (Wilensky, 1997). Through using the model, students can see how population trends are affected by the amount of sheep, wolves, and grass over time. Students can write algorithmic explanations (what steps the organisms follow when the model runs) for each of the organisms in the model: sheep, wolves, and grass. Then students can look at the model code and identify algorithm concepts, like branching: if a sheep and a wolf are on the same square, then the wolf eats the sheep. Students can then try changing aspects of the model, like how much the sheep and wolves move with every step. They could add code so that the wolves move towards a sheep if they see one, rather than moving randomly. This is just one example of how algorithmic explanations can connect to and scaffold computational modeling practices. We argue the prior knowledge of algorithm concepts provides a structure that makes the code more readable and understandable to students and can scaffold code modification and creation of new code.

Theoretical Foundations

While many have suggested benefits to incorporating unplugged CT approaches, little work has been done to ground these approaches in learning theories (Huang & Looi, 2021). This section will describe the theoretical basis for CT-AE: writing-to-learn as a form of constructionism.

Constructionism, like constructivism, involves building knowledge through experiences. Students do not absorb information told to them but are active constructors of their own knowledge as "they make tentative interpretations of experiences and go on to elaborate and test those predictions" (Perkins, 1991, p. 20). Constructionism adds the idea that learning happens best when students create public products "that can be shown, discussed, examined, probed, and admired" (Papert, 1993, p. 142). As such, learning experiences should be designed to engage students in the construction of meaning through the creation of artifacts that are publicly shareable. Students learn through constructing an artifact and from interpreting an artifact (Papert, 1980). The co-occurrence and cycling of interpretation and representation result in the co-construction of knowledge and an artifact. While constructionism has guided many computational and programming learning experiences (e.g., Harel & Papert, 1990; Kahn, 1999; Papert, 1980; Wilensky & Reisman, 2006), pencil-and-paper constructions utilize the same learning approach (Papert & Harel, 1991). CT-AE follows this logic as students create algorithmic explanations, thus engaging in learning-by-making.

CT-AE focuses the constructionist idea of *learning-by-making* by employing writing as the medium for *making*. More specifically, CT-AE leverages algorithm creation as a genre of writing. The writing-to-learn science theory suggests that science knowledge and understanding is constructed through the process of writing (Yore et al., 2003). Students use inquiry experiences to collect evidence that is used to construct explanations and understandings about the science phenomenon. Consistent with constructionist theory, language does not only describe "what the scientist does but it actually helps determine it. The relationship of the scientific paradigm and its language is a reciprocal one: language shapes the paradigm, and the paradigm shapes the language." (Locke, 1992, p. 33). Writing-to-learn not only supports learning about the science, but it also supports learning of writing, itself, allowing students to make sense of science and to effectively communicate it (Yore et al., 2003).

We argue that creating algorithmic explanations is a genre of writing and thinking, and it connects to several of the writing-to-learn genres. Gallagher et al. (1993) argue that writing-to-learn approaches should include several genres of writing, including narrative, descriptive, explanation, instruction, and argumentation. Algorithmic writing clearly connects to the explanation and instruction genres.

> Explanation involves sequencing events in cause–effect relationships. Instruction involves ordering a sequence of procedures to specify directions, such as a manual, experiment or recipe, and can effectively utilize a series of steps in which the sequence is established by tested science and safety. (Yore et al., 2003, p 700)

Students engage in sequential and cause-effect reasoning through writing algorithmic explanations. Unplugged algorithm creation as a form of writing not only allows students to find meaning and effectively communicate with other people (a publicly shareable artifact), but also supports the development of foundational algorithm skills that can be applied to computational tools and systems. Just as writing is central to learning and engaging in science, CT has become central to scientific inquiry. As such, combining writing-to-learn approaches with algorithmic explanations can support constructionist learning of essential science practices and ideas.

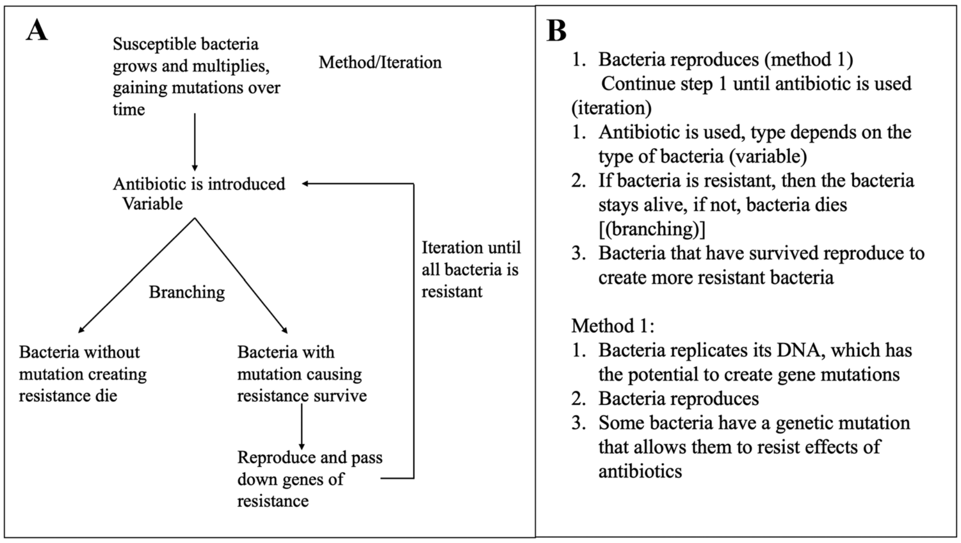
A key component of constructionism is allowing students to make products with varying approaches that match their way of thinking and constructing (Papert & Harel, 1991). Students are limited with the ideas they can employ within programming environments because these environments require specific and correct syntax and prior experience with the language and coding environment. Conversely, CT-AE leverages everyday language to allow students freedom to construct in a common medium (i.e., words) using logic based in computer science that provides meaningful structure to their construction. It is important to consider the affordances and limitations of the construction medium. If creating and sharing a public artifact are key to constructionism, some media impact the shareability of the product. For example, if students are creating a computer program, they can only share it with others who know the programming language. Using an unplugged approach with everyday language alleviates these issues and results in a product that can be publicly shared with everyone.

CT-AE in Practice

The purpose of this manuscript is to elaborate, describe, and justify the CT-AE framework and approach. In the following section, we cite evidence collected across two studies that support the use of the CT-AE approach. Here, we provide an example of CT-AE in use through a sample unit that incorporated multiple algorithm creation activities to support students' learning about the process of natural selection (Peel et al., 2021b). To begin the unit, students' prior knowledge was assessed with an initial algorithm creation task where students were prompted to write the steps of natural selection without any instruction. Next, students were introduced to algorithm concepts and practices by explaining a familiar, everyday process—getting ready in the morning (Peel & Friedrichsen, 2018). They wrote process steps and discussed them with peers. The teacher then led a discussion of algorithm concepts.

Next, students applied algorithm concepts and practices to make sense of natural selection. Students did a lab investigation of the development of antibiotic resistance in bacteria (Williams et al., 2018) and used their results as evidence to explain natural selection with an algorithm. Students also investigated horn size change in a big horned sheep population (Adapted from Chinn & Duncan, 2014) and flowering time change in a field mustard population. After each investigation, students created algorithmic explanations of the process using their class exploration as evidence to support their explanations (a sample student algorithm is presented in Fig. 3).

To help students see similarities of the natural selection process between the three contexts, students were tasked with creating generalized algorithmic explanations of natural selection (Fig. 4). Students examined their bacteria, sheep, and mustard algorithms for similarities and used variables and methods to replace similarities between contexts to create generalized algorithms that could be used to explain most natural selection scenarios. This process of using algorithmic explanations of natural selection across contexts engaged students in CT practices and algorithm concepts while helping them understand the intricacies of the natural selection process. See Peel et al. (2021b) for full unit details and materials.


A
Susceptible bacteria grows and multiplies, gaining mutations over time
Method/Iteration
Antibiotic is introduced Variable
Iteration until all bacteria is resistant
Branching
Bacteria without mutation creating resistance die
Bacteria with mutation causing resistance survive
Reproduce and pass down genes of resistance
B

1. Bacteria reproduces (method 1)
   Continue step 1 until antibiotic is used

(iteration)

1. Antibiotic is used, type depends on the type of bacteria (variable)
2. If bacteria is resistant, then the bacteria stays alive, if not, bacteria dies [(branching)]
3. Bacteria that have survived reproduce to create more resistant bacteria

Method 1:

1. Bacteria replicates its DNA, which has the potential to create gene mutations
2. Bacteria reproduces
3. Some bacteria have a genetic mutation that allows them to resist effects of antibiotics

Fig. 3 Example antibiotic resistance algorithmic explanations

CT-AE Results

The natural selection implementation described in the previous section has been successfully implemented in several secondary biology classrooms (Peel et al., 2019a, b, 2021b). Empirical investigations of student learning outcomes from this implementation indicate the CT-AE approach supported the development of natural selection understandings and CT competencies. Student learning outcomes from these published studies are summarized in the following sections to provide evidence that supports the CT-AE approach.

Science Learning

The unit described above was implemented and data were collected from 113 10th grade biology students (Peel et al., 2019b). Results from this study indicate the CT-AE approach supported student learning of natural selection. Science learning was measured via students' hand-written algorithmic explanations of natural selection (e.g., Fig. 4). Students' pre- and post-unit algorithmic explanations were analyzed for correct usage of seven natural selection factors: mutation, favorable trait, initial variation, selective pressure, differential survival, reproduction, and population shift. Algorithms were evaluated for the correct sequence of steps, specifically analyzing if the algorithms contained all steps and if the steps were in the correct order. Natural selection misconceptions were identified and categorized in both pre- and post-unit algorithms.

1. Population = all trait A
2. Pop Size = large
3. Random mutation occurs during reproduction, introducing trait B
4. Individuals with traits A and B reproduce
5. Population = some with trait A and some with trait B
6. Pop Size = large
7. Selection pressure is introduced
8. Favorable trait = trait B
9. If individual has trait A, then they die
   a. Else, they survive
10. Population = all trait B and pop size = small
11. If individual survives, then they reproduce
12. Population = all trait B and pop size = large

Fig. 4 Example natural selection algorithmic explanation

Each natural selection factor was used significantly more in post-unit algorithmic explanations compared to pre-unit algorithmic explanations. The mean natural selection sequencing score significantly increased from pre- to post-unit algorithmic explanations with a large effect size. Needs-based misconceptions (populations change because they need to survive) and individual change misconceptions (individual organisms change their traits) significantly decreased with a large effect size after the unit. Relationships between natural selection factors and algorithm concepts in students' post-unit algorithms showed the synergies between algorithmic explanations and natural selection. Students used branching to explain differential survival; iteration to show reproduction repetition and repetition of the whole natural selection process; and variables to generalize selection pressure, favorable trait, population, mutation, and organism. These results indicate the CT-AE approach used to integrate unplugged CT and natural selection supported student content learning.

CT Learning

Another empirical investigation from the same implementation has shown CT learning increases after the CT-AE unit

(Peel et al., 2019a). This work is an analysis of the six natural selection algorithmic explanations created before, during, and after the unit. Algorithm concepts (branching, iteration, method, and variable) were analyzed for explicit use or implicit use (i.e., algorithm concept was labeled or not labeled) and single use or multiple use (i.e., one use of the algorithm concept, vs more than one use of the algorithm concept). A comparison of pre- to post-unit algorithmic explanations showed a significant increase in each algorithm concept with large or moderate effect sizes. This indicates the CT-AE approach supported the development of CT skills because students could use more algorithm concepts in their algorithmic explanations after the unit.

When comparing algorithm concept usage in post-unit algorithms, variables were used the most, followed by branching and iteration, and methods were used the least. This suggests students may have struggled to incorporate methods more than other algorithm concepts. To examine trends in the development of students' conceptions of algorithm concepts over the course of the unit, each algorithm concept was analyzed within the six algorithms created during the unit. Each algorithm concept usage increased from pre-unit algorithms (algorithm 1) to algorithm 2, and from algorithm 2 to algorithm 5. Scores remained high in students' post-unit algorithms (algorithm 6). The steady increase of algorithm concept usage throughout the unit indicates students gradually developed CT competencies. After the CT-AE natural selection unit, students came to understand CT as a tool used to explain and understand science processes. Students were also able to suggest other biological and everyday processes in which algorithmic explanations could be used. These results indicate the CT-AE approach supported student development of CT competencies and understandings through the creation of unplugged algorithmic explanations of natural selection.

Discussion

In an extensive review of unplugged CT literature, Huang and Looi (2021) call for rigorous empirical investigations of the benefits of unplugged CT approaches. Specifically, they propose questions for future research. We believe the CT-AE instructional approach can directly address these questions.

1. What theories effectively describe how unplugged approaches foster learning CT?
2. How do unplugged approaches that separate CT from coding influence how we define, teach, and measure CT?
3. How do we assess CT in unplugged activities without using code representations?
4. How are unplugged activities facilitated, especially in K–12 classrooms?
5. How do we integrate unplugged activities into existing school subjects? (Huang & Looi, 2021, p. 17)

The unplugged instructional approach described in this paper is grounded in theory to support student learning of CT and science content (#1, #5). The approach decouples CT and programming and provides new avenues for exploring CT integration and assessment (#2). This paper describes a curricular example of the unplugged approach with student work, which can inform the approach's facilitation in the classroom and the assessment of CT with this approach (#3, #4, #5).

Potential Outcomes

We propose that the integration of CT-AE in science classes has several benefits for students and teachers (Table 3). The following sections describe CT-AE's potential outcomes with support from existing literature.

Science Content Knowledge and Computational Literacy Increases

The process of creating unplugged algorithmic explanations supports the simultaneous development of science ideas and algorithmic logic and practices. Our pilot implementation has shown increases in natural selection conceptions and CT understanding (Peel et al., 2019a, b, 2021b). While positive science (Mensan et al., 2020) and CT learning outcomes (Brackmann et al., 2017; Delal & Oner, 2020; Looi et al., 2018; Rodriguez et al., 2017; Tsarava et al., 2019) have been achieved with unplugged approaches, CT-AE supports synergistic science and CT learning, which has the potential to contribute to the development of a STEM workforce with computing competencies.

Table 3 CT-AE potential outcomes

Intervention	Potential direct outcomes	Potential downstream outcomes
• CT concepts and practices taught through unplugged algorithmic explanations of science processes • CT-AE implemented in core science courses	• Science content knowledge increases • Computational literacy increases • All students learn computer science concepts • Scaffolds other CT practices • Implementation barrier decreases	• Better programmers • More people interested in computing and computing-related classes and careers • STEM workforce with computing competencies

Broadening Computational Participation

The implementation of CT-AE in core science classes also broadens computational literacy education to include all students since only a small percentage of students elect to enroll in computer science courses (Nager & Atkinson, 2016). CT-AE is designed to leverage unplugged learning in science contexts, which can be beneficial for student engagement and interest in learning with CT, especially for student populations that have been historically marginalized. Women struggle with confidence regarding computers (Beyer et al., 2003; Bock et al., 2013) and find computer science environments "chilly," which makes them feel like they do not belong (Master et al., 2016; Walton et al., 2015). Hispanic and Black students are also underrepresented populations in computer science and cite lack of experience with computing and viewing computers as difficult to use and understand as reasons for not pursuing computer science (Bock et al., 2013; Buzzetto-More et al., 2010).

In addition to engaging more students in computing through its integration with science, we argue the unplugged approach provides an easier and less intimidating entry point to CT because it leverages students' everyday language and makes explicit connections between intuitive thinking and algorithm concepts. Once the foundation is set, these ideas can provide an on-ramp to plugged-in CT. Understanding algorithmic logic before plugging in should help students feel more comfortable, confident, and prepared to program. Increased comfort, interest, and engagement in CT lessons may facilitate long term interest in STEMC careers.

Other studies have shown unplugged and programming-free approaches support the development of interest and active engagement in CT in young women and students from minorities that are typically underrepresented in STEM and computing careers. Brady et al. (2017) showed that participatory simulations, or unplugged simulations, where students act out phenomena following computational rules, supports young women's interest and engagement in CT. Sabitzer and Pasterk (2014) used an unplugged approach to incorporate CT into science where students showed increased interest in computer science and knowledge about informatics. Del Olmo-Muñoz et al. (2020) unplugged vs plugged comparison study found that girls and boys learned CT equally well. However, unplugged lessons led to higher motivation in girls when compared to plugged approaches.

Scaffolds Other CT Practices: Computational Modeling and Programming

Unplugged approaches can scaffold plugged approaches and other CT practices described in the CT in science and mathematics taxonomy: computational modeling, programming, computational visualizations, computational data, and computational problem-solving practices (Peel et al., 2021; Weintrop et al., 2016). We argue that learning unplugged approaches first helps students understand CT in deeper ways, which should allow students to apply CT more successfully in other contexts, including programming. Caeli and Yadav (2020) argue that understanding algorithms and algorithm design is key to the computational problem-solving approach, and programming languages can hinder understanding algorithms and algorithm design due to complex syntax. To address this challenge, they suggest that "in order for learners to conceptually understand computer science ideas and practices, we need to add or even begin with unplugged approaches" (Caeli & Yadav, 2020, p. 31). CT-AE can build computational modeling practices (Peel et al., 2021) by scaffolding block-based and text-based programming, which can both be used to modify and create computational models. Thus, CT-AE supports multiple computational modeling practices: using, modifying, and creating computational models (Peel et al., 2021). Algorithmic explanations can be important steps between conceptual models and computational models because they provide essential scaffolding that connects common language and computational language.

The CT-AE instructional approach aligns with other unplugged literature showing that unplugged CT lessons can support programming skills and learning (Grover et al., 2019; Hermans & Aivaloglou, 2017). While these articles used unplugged lessons that supported programming, they were not situated in science contexts. The clear next step for unplugged education is to reach a wider range of students, which can be done by incorporating unplugged CT into core science courses. However, disconnected and decontextualized unplugged CT approaches will not integrate well into science courses because they need to be meaningfully integrated with science content. In contrast to decontextualized unplugged CT approaches, CT-AE is anchored in science and leverages unplugged CT to support science learning.

Implementation Barrier Decreases

CT-AE has potential to help alleviate many of the barriers to CT and science integration. Just as the programming threshold has been lowered through block-based programming environments (Repenning et al., 2010), unplugged CT approaches can continue to lower the threshold and time commitment while still providing substantive learning experiences. Additionally, leveraging algorithms as a medium for constructionist learning shifts CT from problem-solving to sense-making, thus refocusing CT integration on synergistic science and CT learning.

Future Work

Empirical investigations of the pilot CT-AE unit show promising and productive student learning outcomes related to science content and CT skills. Future analysis will include investigations of student discourse and interactions during the generalized natural selection lesson. Data indicated the natural selection lesson supported student learning of natural selection and algorithm concepts. As such, understanding how this lesson engaged students in sense-making may shed light on how the CT-AE approach impacts students' understandings. Next steps for the CT-AE approach are to integrate unplugged CT with more science processes to broaden implementation and test the approach's effectiveness in other science disciplines taught in K–12 schools. The natural selection process has key connections to algorithm concepts, which may have facilitated the integration. For example, differential survival is often explained through branching (CT concept) as follows: if the organism has the favorable trait, then they are more likely to survive. Other processes with CT synergies need to be identified and investigated. Another avenue for integration that needs to be explored is CT-AE integrations and applications in middle school and elementary school settings. While the creation of full algorithmic explanations may be too complex for young learners, aspects of CT-AE can be leveraged to support science learning and CT development. For example, branching can be introduced and used to make sense of cause and effect relationships in ecosystems, or iteration can be used to support learning about life cycles, the water cycle, or the carbon cycle. Future work with CT-AE can be done to expand the approach to wider grade ranges in order to support systemic development of CT competencies.

To foster CT integration, the CT-AE approach needs to be introduced to teachers. Teachers are key actuators of curricular change, and it will be important to engage them in designing and implementing CT-AE curricula. Investigations of teacher practice, comfort, confidence, self-efficacy, and professional development related to CT-AE will be important for understanding the effectiveness of the CT-AE approach in broader contexts. We propose the unplugged approach helps teachers implement CT integration, but this needs to be empirically investigated in varying contexts.

We have argued that unplugged CT approaches support student learning in several ways. First, it helps students make sense of science content. Second, it supports the development of fundamental CT skills. The above sections show evidence of both learning outcomes. However, the next key step in CT integration in science is to connect these fundamental CT skills to computing. It will be important for students to build on these skills to engage in authentic scientific inquiry CT practices. The connection between unplugged and plugged approaches has not been empirically investigated or substantiated (Huang & Looi, 2021). Studies referenced above show that unplugged lessons support programming knowledge and student engagement. However, these studies do not take place in integrated science learning environments. Future research needs to focus on how unplugged approaches, such as CT-AE, support and scaffold engagement in computational scientific inquiry.

Acknowledgements This work would not be possible without the contributions and support of Kerri Graham, Dr. Andrew Kinslow, Kaitlin EuDaly, Eric Hayes, and Dr. Michael Horn.

Declarations

Ethics Approval and Consent to Participate The work presented in this paper received approval by the appropriate ethics committee for research involving humans and informed consent for human participants was followed.

Conflict of Interest The authors declare no competing interests.

References

Abbott, B. P., Abbott, R., Abbott, T. D., Abernathy, M. R., Acernese, F., Ackley, K., ... & Cavalieri, R. (2016). Observation of gravitational waves from a binary black hole merger. *Physical Review Letters*, *116*(6), 061102.

Aljowaed, M., & Alebaikan, R. A. (2018). Training needs for computer teachers to use and teach computational thinking skills. *International Journal for Research in Education, 42*(3), 237–284.

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. *Journal of Educational Technology & Society, 19*(3), 47.

Arastoopour Irgens, G., Dabholkar, S., Bain, C., Woods, P., Hall, K., Swanson, H., Honr, M., & Wilensky, U. (2020). Modeling and measuring high school students' computational thinking practices in science. *Journal of Science Education and Technology, 29*(1), 137–161.

Arnold, C. (2020). How computational immunology changed the face of COVID-19 vaccine development. *Nature Medicine*.

Aslan, U., LaGrassa, N., Horn, M., & Wilensky, U. (2020c). *Putting the taxonomy into practice: Investigating students' learning of chemistry with integrated computational thinking activities.* Paper presented at the American Education Research Association (AERA) 2020 Annual Meeting.

Basu, S., Biswas, G., Sengupta, P., Dickes, A., Kinnebrew, J. S., & Clark, D. (2016). Identifying middle school students' challenges in computational thinking-based science learning. *Research and Practice in Technology Enhanced Learning, 11*(1), 1–35.

Beyer, S., Rynes, K., Perrault, J., Hay, K., & Haller, S. (2003). Gender differences in computer science students. In *Proceedings of the 34th SIGCSE technical symposium on Computer Science Education* (pp. 49–53).

Bock, S. J., Taylor, L. J., Phillips, Z. E., & Sun, W. (2013). Women and minorities in computer science majors: Results on barriers from interviews and a survey. *Issues in Information Systems, 14*(1), 143–152.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). Developing computational thinking in compulsory education-Implications for policy and practice. *Joint Research Centre (Seville site), No. JRC104188*.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017, November). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 65–72).

Brady, C., Orton, K., Weintrop, D., Anton, G., Rodriguez, S., & Wilensky, U. (2017). All roads lead to computing: Making, participatory simulations, and social computing as pathways to computer science. *IEEE Transactions on Education, 60*(99), 1–8.

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking.* Paper presented at the Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada.

Buckley, B. C., Gobert, J. D., Kindfield, A., Horwitz, P., Tinker, R., Gerlits, B., Wilensky, U., Dede, C., & Willett, J. (2004). Model-based teaching and learning with BioLogica™: What do they learn? How do they learn? How do we know? *Journal of Science Education and Technology, 13*(1), 23–41.

Buzzetto-More, N. A., Ukoha, O., & Rustagi, N. (2010). Unlocking the barriers to women and minorities in computer science and information systems studies: Results from a multi-methodolical study conducted at two minority serving institutions. *Journal of Information Technology Education: Research, 9*(1), 115–131.

Caeli, E. N., & Yadav, A. (2020). Unplugged approaches to computational thinking: A historical perspective. *TechTrends, 64*(1), 29–36.

Chinn, C. & Duncan, R.G. (2014). *Promoting Reasoning and conceptual change in science (PRACCIS) curriculum: Evolution*. Retrieved from https://sites.google.com/a/gse.rutgers.edu/praccis-promoting-reasoning-and-conceptual-change-in-science/home

Csizmadia, A., & Boulton, H. (2017). *Computational thinking–Back to the future.* Paper presented at the Conference Proceedings. The Future of Education.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). Computational thinking-A guide for teachers. Retrieved from Computing at School website: https://community.computingatschool.org.uk/resources/2324

Delal, H., & Oner, D. (2020). Developing middle school students' computational thinking skills using unplugged computing activities. *Informatics in Education, 19*(1), 1–13.

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/10.1016/j.compedu.2020.103832

DiSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy*. Mit Press.

Event Horizon Telescope Collaboration. (2019). First M87 event horizon telescope results. I. The shadow of the supermassive black hole. arXiv preprint arXiv:1906.11238

Forrester, J. W. (1994). System dynamics, systems thinking, and soft OR. *System Dynamics Review, 10*(2–3), 245–256.

Fraillon, J., Schulz, W., Duckworth, D., & Ainley, J. (2018). ICILS 2018 Assessment Framework. Amsterdam: IEA. Manuskript in Vorbereitung.

Futschek, G. (2006, November). Algorithmic thinking: The key for understanding computer science. In *International conference on informatics in secondary schools-Evolution and perspectives* (pp. 159–168). Springer, Berlin, Heidelberg.

Gallagher, M., Knapp, P., & Noble, G. (1993). Genre in practice. In B. Cope & M. Kalatzis (Eds.), *The power of literacy: A genre approach to teaching writing* (pp. 179–202). University of Pittsburgh Press.

Grover, S., Jackiw, N., & Lundh, P. (2019). Concepts before coding: Non-programming interactives to advance learning of introductory programming concepts in middle school. *Computer Science Education, 29*(2–3), 106–135.

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

Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments, 1*(1), 1–32.

Heintz, F., Mannila, L., & Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K–12 education.

Hermans, F., & Aivaloglou, E. (2017, November). To scratch or not to scratch? A controlled experiment comparing plugged first and unplugged first programming lessons. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 49–56).

Hohlfeld, T. N., Ritzhaupt, A. D., Dawson, K., & Wilson, M. L. (2017). An examination of seven years of technology integration in Florida schools: Through the lens of the levels of digital divide in Schools. *Computers & Education, 113*, 135–161.

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

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

Kahn, K. (1999). From prolog to zelda to toontalk. Proceedings of International Conference on Logic Programming

Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing, 4*(3), 583.

Kite, V., & Park, S. (2020). Secondary science teachers' conceptualizations of computational thinking and perceived barriers to CT/content integration. Prepared for the 2020 annual meeting of the National Association for Research in Science Teaching and teacher education (NARST).

Klopfer, E. (2003). Technologies to support the creation of complex systems models—Using StarLogo software with students. *Bio Systems, 71*(1), 111–122.

Li, Y., Schoenfeld, A. H., diSessa, A. A., Graesser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2020). On computational thinking and STEM education. *Journal for STEM Education Research*. https://doi.org/10.1007/s41979-020-00044-w

Locke, D. (1992). *Science as writing*. Yale University Press.

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

Looi, C. K., How, M. L., Longkai, W., Seow, P., & Liu, L. (2018). Analysis of linkages between an unplugged activity and the development of computational thinking. *Computer Science Education, 28*(3), 255–279.

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

Master, A., Cheryan, S., & Meltzoff, A. N. (2016). Computing whether she belongs: Stereotypes undermine girls' interest and sense of belonging in computer science. *Journal of Educational Psychology, 108*(3), 424.

Mensan, T., Osman, K., & Majid, N. A. A. (2020). Development and validation of unplugged activity of computational thinking in science module to integrate computational thinking in primary science education. *Science Education International, 31*(2), 142–149.

Nager, A., & Atkinson, R. D. (2016). *The case for improving U.S. computer science education* (SSRN Scholarly Paper No. ID 3066335). Rochester, NY: Social Science Research Network. Retrieved from https://papers.ssrn.com/abstract=3066335

National Research Council. (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. National Academies Press.

NGSS Lead States (2013). *Next generation science standards. For states, by states*. Washington, DC: The National Academies Press.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas.

Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. BasicBooks, 10 East 53rd St., New York, NY 10022–5299.

Papert, S., & Harel, I. (1991). *Situating Constructionism Constructionism, 36*(2), 1–11.

Peel, A., Dabholkar, S., Anton, G., Wu, S., Wilensky, U., & Horn, M. (2020). A case study of teacher professional growth through co-design and implementation of computationally enriched biology units. In Gresalfi, M. and Horn, I. S. (Eds.), The Interdisciplinarity of the Learning Sciences, 14th International Conference of the Learning Sciences (ICLS) 2020, Volume 4 (pp. 1950-1957). Nashville, Tennessee: International Society of the Learning Sciences. https://repository.isls.org//handle/1/6478

Peel, A., Dabholkar, S., Wu, S., Horn, M.S., Wilensky, U. (2021a). An evolving definition of computational thinking in science and mathematics classrooms. *Proceedings of the 5th APSCE International Computational Thinking and STEM in Education Conference 2021*, (pp. 119-122).

Peel, A., & Friedrichsen, P. (2018). Algorithms, abstractions, and iterations: Teaching computational thinking using protein synthesis translation. *The American Biology Teacher*, *80*(1), 21-28.

Peel, A., Fulton, J., & Pontelli, E. (2015). DISSECT: An experiment in infusing computational thinking in a sixth grade classroom. *Paper presented at the Frontiers in Education Conference (FIE)*, 2015. 32614 2015. IEEE.

Peel, A., Sadler, T. D., & Friedrichsen, P. J. (2019a). Learning Computational Thinking Through Unplugged Algorithmic Explanations of Natural Selection. In Developing computational thinking competencies and natural selection understanding through unplugged algorithmic explanations. [Doctoral Dissertation, University of Missouri].

Peel, A., Sadler, T. D., & Friedrichsen, P. J. (2019b). Learning Natural Selection Through Computational Thinking: Unplugged Design of Algorithmic Explanations. *Journal of Research in Science Teaching*.

Peel, A., Sadler, T. D., & Friedrichsen, P. J. (2021b). Using unplugged computational thinking to scaffold natural selection learning. *The American Biology Teacher*, *83*(2), 112-117.

Perkins, D. N. (1991). Technology meets constructivism: Do they make a marriage? *Educational Technology, 31*(5), 18–23.

Repenning, A., Webb, D., & Ioannidou, A. (2010, March). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 265–269).

Rich, P. J., Larsen, R. A., & Mason, S. L. (2020). Measuring teacher beliefs about coding and computational thinking. *Journal of Research on Technology in Education*, 1–21.

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017, March). Assessing computational thinking in CS unplugged activities. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 501–506).

Sabitzer, B., & Pasterk, S. (2014). *Cool informatics: A new approach to computer science and cross-curricular learning.* Paper presented at the Proceedings of the European Conference on Technology in the Classroom 2014, Brighton, United Kingdom.

Sands, P., Yadav, A., & Good, J. (2018). Computational thinking in K-12: In-service teacher perceptions of computational thinking. In Computational Thinking in the STEM Disciplines (pp. 151–164). Springer, Cham.

Schwarz, C. V., Meyer, J., & Sharma, A. (2007). Technology, pedagogy, and epistemology: Opportunities and challenges of using computer modeling and simulation tools in elementary science methods. *Journal of Science Teacher Education, 18*(2), 243–269.

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

Spohrer, J. C. (1989). Marcel: a generate-test-and-debug (gtd) impasse/repair model of student programmers.

Tsarava, K., Leifheit, L., Ninaus, M., Román-González, M., Butz, M. V., Golle, J., Trautwein, U., & Moeller, K. (2019, October). Cognitive correlates of computational thinking: Evaluation of a blended unplugged/plugged-in course. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education* (pp. 1–9).

Walton, G. M., Logel, C., Peach, J. M., Spencer, S. J., & Zanna, M. P. (2015). Two brief interventions to mitigate a "chilly climate" transform women's experience, relationships, and achievement in engineering. *Journal of Educational Psychology, 107*(2), 468.

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.

White, B. Y., & Frederiksen, J. R. (1998). Inquiry, modeling, and metacognition: Making science accessible to all students. *Cognition and Instruction, 16*(1), 3–118.

Wilensky, U. (1997). NetLogo Wolf Sheep Predation model. http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Wilensky, U. (1999). NetLogo. Northwestern University, Evanston, IL: Center for Connected Learning and Computer-Based Modeling. http://ccl.northwestern.edu/netlogo/

Wilensky, U. (2001) *Modeling nature's emergent patterns with multi-agent languages*. Proceedings of EuroLogo 2001. Linz, Austria

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theories—an embodied modeling approach. *Cognition and Instruction, 24*(2), 171–209.

Wilensky, U., & Resnick, M. (1999). Thinking in levels: A dynamic systems perspective to making sense of the world. Journal of Science Education and Technology, 8(1).

Wilkerson-Jerde, M., Wagh, A., & Wilensky, U. R. I. (2015). Balancing curricular and pedagogical needs in computational construction kits: Lessons from the DeltaTick Project. *Science Education, 99*(3), 465–499.

Williams, M. A., Friedrichsen, P. J., D. Sadler, T., & Brown, P. J. (2018). Modeling the emergence of antibiotic resistance in bacterial populations. *The American Biology Teacher*, *80*(3), 214-220.

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

Wing, J. M. (2011). Computational thinking. In *VL/HCC* (p. 3).

Wu, L., Looi, C. -K., Liu, L., & How, M. -L. (2018). Understanding and developing in-service teachers' perceptions towards teaching in computational thinking: Two studies. *Proceedings of the 26th International Conference on Computers in Education., Philippines: Asia-Pacific Society for Computers in Education*.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE), 14*(1), 5.

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

Yore, L., Bisanz, G. L., & Hand, B. M. (2003). Examining the literacy component of science literacy: 25 years of language arts and science research. *International Journal of Science Education, 25*(6), 689–725.