International Society of the Learning Sciences

The Role of Science Proficiency in Students' Engagement with Computational Modeling Practices

Satabdi Basu, SRI International, satabdi.basu@sri.com
Arif Rachmatullah, SRI International, arif.rachmatullah@sri.com
Shruti Jain, Vanderbilt University, shruti.jain@vanderbilt.edu
Keun-Woo Lee, Digital Promise, klee@digitalpromise.org
Kevin W. McElhaney, Digital Promise, kmcelhaney@digitalpromise.org

Abstract: Computational modeling of scientific systems is a powerful approach for fostering science and computational thinking (CT) proficiencies. However, the role of programming activities for this synergistic learning remains unclear. This paper examines alternative ways to engage with computational models (CM) beyond programming. Students participated in an integrated Science, Engineering, and Computational Modeling unit through one of three distinct instructional versions: Construct a CM, Interpret-and-Evaluate a CM, and Explore-and-Evaluate a simulation. Analyzing 188 student responses to a science+CT embedded assessment task, we investigate how science proficiency and instructional versions related to pseudocode interpretation and debugging performances. We found that students in the Explore-and-Evaluate a simulation outperformed students in the programming-based versions on the CT assessment items. Additionally, science proficiency strongly predicted students' CT performance, unlike prior programming experience. These results highlight the promise of diverse approaches for fostering CT practices with implications for STEM+C instruction and assessment design.

Introduction

Over the past decade, numerous science, technology, engineering, and mathematics (STEM) education initiatives and reforms have aimed to prepare K-12 students with the computational knowledge and skills needed to engage in authentic STEM practices (National Research Council [NRC], 2012). For example, the Next Generation Science Standards (NGSS, 2013) in the U.S. include "using mathematics and computational thinking" as one of eight science and engineering practices, emphasizing that "classroom instruction should enhance all of science through the use of quality mathematical and computational thinking" (NGSS Lead States, 2013). Computational modeling is a professional STEM practice that is widely recognized as an effective means for K-12 learners to forge deep connections between computational thinking (CT) and science and engineering disciplines (McElhaney et al., 2020). A computational model (CM) is a mathematical representation of a system created using a formal notation (e.g., a programming language) so that the system behavior can be studied using computer simulations (Melnik, 2015). The collective practices of developing, testing, modifying, assessing, and using science-based CMs (NRC, 2012)—can integrate CT concepts and practices with science and engineering learning.

Designing effective instruction for computational modeling in integrated science-CT contexts and training teachers to implement such instructional materials in their classrooms requires a nuanced understanding of student learning and challenges in such settings and relations between the different intersecting disciplinary practices. However, there is currently limited knowledge of how students learn at the intersection of science and CT when they engage in computational modeling. In fact, several studies engage students in computational modeling by having them develop CMs of scientific systems in order to synergistically learn science and CT (e.g., Sengupta et al., 2013; Lee et al., 2024). But, there is little evidence on whether and how engaging with CMs without creating them influences student learning. The benefits of developing models are well established in the science education literature (Schwarz et al., 2009; Windschitl et al., 2008), but it is unclear whether the benefits extend to developing science-based CMs, especially for students who are new to programming. Understanding what aspects of student engagement with science-based CMs reflect students' science proficiency versus CT proficiency can help scaffold student instruction in such integrated settings.

In this paper, we present three versions of a curriculum unit that integrates earth science, engineering and CT for 5th and 6th grade students. Each version entails the use of different computational modeling practices: (1) CM construction, (2) evaluating CMs (instead of creating them), and (2) evaluating simulations (without being able to view the simulation code). We describe a classroom study using these three versions and report on an assessment task scenario with multiple items that was embedded in the curriculum for all three versions. The task measured students' understanding of science concepts and their ability to engage in the CT practices of code interpretation and debugging using pseudocode, and elicited students' proficiencies and challenges at the intersection of science and CT. We use student responses to this assessment task scenario to answer our research

question: *What are the relationships among students' prior programming experience, science proficiency, code interpretation ability, and debugging skills in an unplugged science+CT task involving a science-based CM?* This work has the potential to advance the field of integrated STEM and computing education research in K-12 by providing evidence for multiple instructional approaches to synergistic science and CT learning and contributing to the limited research on student learning at the intersection of science and CT.

Theoretical and empirical perspectives

Integrating STEM and CT using computational modeling

CT in the context of computer science and non-STEM programming activities can differ from CT applied to STEM learning. CT in the context of STEM learning can be defined using a taxonomy of four practices–namely data practices, modeling and simulation practices, computational problem-solving practices, and systems thinking practices, each of which comprises several components (Weintrop et al., 2016). For example, the modeling and simulation practice includes designing, constructing, and assessing CMs and using CMs to understand a concept and find and test solutions, while the computational problem-solving practice includes preparing problems for computational abstractions, creating computational abstractions, choosing effective computational tools, programming, developing modular computational solutions, troubleshooting and debugging, and assessing different solutions to a problem.

CMs are widely recognized as an effective means for learners to forge deep connections between CT and STEM disciplines. CMs provide explicit mechanisms for constructing and visualizing scientific phenomena and reasoning about scientific processes as a sequence of step-by-step changes, as opposed to creating aggregated mathematical models (e.g., Sengupta et al., 2013; Hutchins et al., 2020). Complex scientific systems become easier to understand when the system structure and behaviors are explained computationally, while programming also becomes easier to learn when anchored in real-world problem contexts (Sengupta et al., 2013). Past research on synergistic STEM and computing learning via computational modeling has primarily involved students learning to program CMs representing physical phenomena and has shown to deepen students' science learning (Sengupta et al., 2013; Basu et al., 2017; Waterman et al., 2020; Rachmatullah & Wiebe, 2022). Alongside documenting the benefits of computational modeling, research has also identified the significant difficulties students may have with developing scientific CMs. Students may struggle with both understanding the science concepts underlying CMs and generalizing scientific processes by representing system variable relationships using computational expressions (Basu et al., 2016; Hutchins et al., 2020). These findings indicate a need to scaffold students' computational modeling experience as they grapple with the multi-fold complexities of scientific knowledge, modeling practices, and computational representations.

While CM development has been the most common way to integrate CT with STEM disciplines, a few alternate approaches have been studied. For instance, in 'decoding' activities (Lee et al., 2024; Rabinowitz et al., 2023), students elucidate connections between the provided code and the represented scientific processes, thus fostering a deeper understanding of both the code and the underlying science. Such approaches report positive impacts on CT learning, but their effects on science learning are still unclear and their comparison with traditional CM development approaches remain underexplored.

Our alternate approaches to developing CMs involve evaluating existing CMs and existing simulations without looking at the code. According to the CT taxonomy for math and science (Weintrop et al., 2016), evaluation or assessment of CMs is an important CT practice that involves determining which aspects of a phenomenon are correctly modeled in a CM and which aspects are incorrectly modeled or left out. Weintrop et al. (2016) also assert that engagement in CT practices can include computer simulations. In our activities involving computer simulations, the code driving the simulations remains hidden, allowing students to focus solely on the scientific phenomena. Research demonstrates the benefits of learning using simulations such as student conceptual understanding, inquiry skills, knowledge acquisition, and overall satisfaction (e.g., Xie et al., 2018). However, the relative affordances of evaluating CMs or simulations relative to using or constructing them have not yet been investigated. Understanding the unique affordances of different computational modeling practices for students' science, engineering, and CT learning is critical to enabling educators to implement integrated STEM and computing instruction in ways that meet the needs of specific classroom contexts.

Science-based computational modeling versus domain-general programming

Science-based computational modeling can differ from programming in domain-general contexts in important ways. For example, creating a science-based CM requires both programming knowledge and science content knowledge. In domain-general programming, the Use-Modify-Create (UMC) progression can engage students in increasingly deep interactions with CT practices, with the ultimate goal of developing a functional program (Lee

et al., 2011). During the "use" stage, students are consumers of a program created by someone else. In "modify," students alter features of an existing program or incorporate new features over time. In "create," ownership of the program begins to shift to the student through application of attained computing skills on a new computational project (Lee et al., 2011). The UMC sequence is widely recognized as a valuable approach to scaffold students' understanding of CT beyond surface level understanding of programs and foster student ownership of the programming process (Lytle et al., 2019).

However, when CT is integrated into disciplinary education, for example, science+CT, developing a functional program or a science-based CM is typically not the end goal but rather a step in a broader problem-solving process such as making a scientific prediction or creating an engineering design solution using the CM (McElhaney et al., 2020). For instance, students might develop kinematics CMs to determine the height from which a drone can drop a medical package safely or the distance from which a truck should start braking to halt at a Stop sign (McElhaney et al., 2023). In such integrated science-CT settings, the UMC progression is often not sufficient for STEM disciplinary learning, and only supports CT learning. For example, in Project GUTS, a program that engaged youth in CMs for scientific inquiry on water resources, the UMC progression resulted in student learning gains that were primarily in the areas of CT, and not in grade-level appropriate science content learning (Lee, 2024). Students were not focused on making scientifically accurate models and often made modifications to the program/model that were purely visual and unrelated to the science content.

To enhance students' science learning in an integrated science-CT context, the practice of decoding can enable students to make connections between code and the scientific phenomenon it represents (Lee, 2024). Decoding involves identifying the scientific process represented in the CM, examining how the CM embodies the scientific processes, and assessing the extent to which the CM reflects student knowledge of the scientific process (Lee, 2024). For both decoding and creation of science-based CMs, background science knowledge plays a critical role in preparing students for these CM activities. Unlike programming tasks like game development where learners may have everyday experience with the mechanics of games, science knowledge underpins CMs in very specific ways in integrated science+CT contexts (Basu et al., 2016; Hutchins et al., 2020). Students therefore need ample opportunities to learn the science content prior to engaging in CM activities. We conduct this study using a refined curriculum unit that supports students to develop and refine a conceptual scientific model before engaging them in a range of science-based computational modeling activities

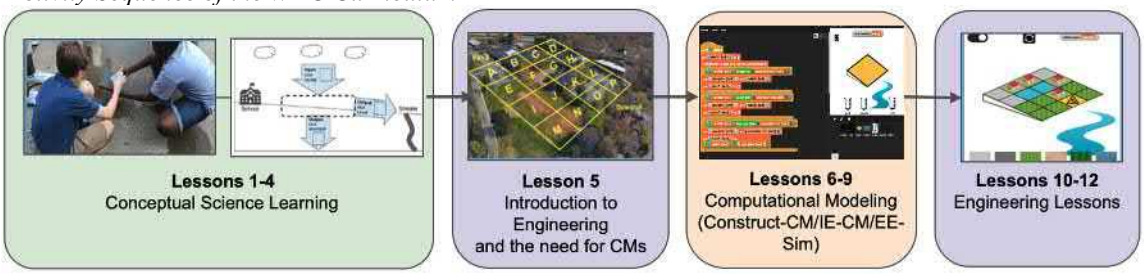
Methods

Study context: Water Runoff Challenge (WRC) curriculum unit description

We developed three versions of a curriculum unit that integrates earth science, engineering and CT, called the WRC, designed for fifth and sixth-grade science classrooms. In the WRC, students are tasked with redesigning a schoolyard to minimize water runoff while meeting design criteria such as budget constraints, accessibility for individuals with physical disabilities, and usage. Each version of the curriculum includes 12 lessons spanning about 10 hours of instruction and addresses three key disciplines: Earth science, CT, and engineering (see Figure 1). The science and engineering lessons are identical across all versions, but the CT lessons vary.

Figure 1

Activity Sequence of the WRC Curriculum



The WRC begins with four science lessons where students are introduced to the flooding problem, learn about water runoff, and engage in a hands-on activity to deepen their understanding of how surface material properties influence water absorption and runoff. Students develop pictorial models to illustrate how some rainfall is absorbed by the ground, depending on the surface material's absorption capacity, while the excess stays on the surface as runoff, potentially leading to flooding. These lessons develop students' understanding of the relationships among rainfall, absorption, and runoff.

In lesson 5, students attempt to redesign the schoolyard on paper using a 4x4 grid, while considering key design criteria. They manually calculate the cost and runoff performance of their designs. This exercise underscores for students the limitations of manual testing and the need for CMs that automate testing and refine designs to optimize runoff performance while meeting design requirements. Lessons 6-9 engage students in one of the three different types of computational modeling activities. Students either construct a CM from scratch (construct-CM), interpret and evaluate given CM code (IE-CM), or explore and evaluate simulations based on CMs (EE-sim). By the end of lesson 9, all students will have either constructed or identified (through iterative evaluation) a functional CM or simulation that correctly predicts water runoff for 1 out of the 16 grids of the schoolyard based on specific surface materials and rainfall amounts. Toward the end of lesson 9, students engage in a paper-and-pencil embedded assessment activity measuring students' science proficiency and ability to interpret and debug pseudocode. More details about the different computational modeling activities and the embedded assessment are provided in the following sections.

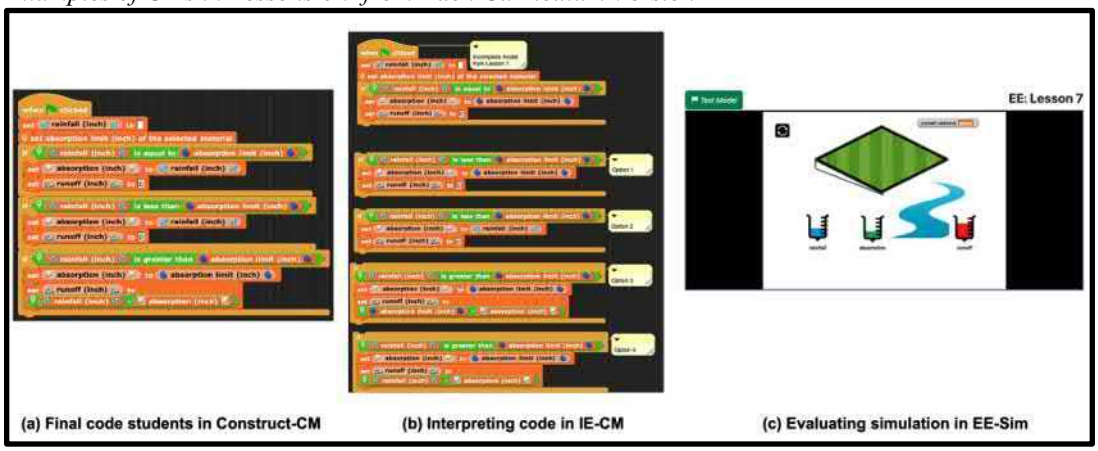
Once students develop or identify a functional CM or simulation, they use it in lessons 10-12 to design a solution for the engineering challenge. Students select surface materials to resurface a 4x4 schoolyard grid, comparing trade-offs between multiple designs while considering cost, accessibility, and runoff performance. The curriculum concludes with an optional presentation lesson where students showcase their final schoolyard designs and justify the effectiveness of their designs.

Computational modeling activities

Students engage with different computational modeling practices in Lessons 6-9 in the three instructional versions. In the Construct-CM version, students begin with a set of paper-and-pencil activities to learn necessary CT concepts, such as sequences, variables, and conditional statements, using a dice-game context (e.g., if DiceA is equal to 5, then set ScoreA to 4 and ScoreB to 3). Following this, students construct their CMs using a block-based, domain-specific modeling language (DSML) (Hutchins et al., 2020). Their goal is to simulate water runoff for a unit surface area (one square) based on specific surface materials and rainfall amounts. The DSML includes domain-specific components like rainfall, water absorption, runoff, and surface materials, as well as CT constructs necessary to create the model. Students construct their CMs over the course of three lessons, programming and testing one if-condition per day. Figure 2a shows a final correct CM.

Figure 2

Examples of CMs in Lessons 6-9 from Each Curriculum Version



Students in the IE-CM version also begin with paper-and-pencil activities to learn CT concepts, mirroring the Construct-CM version. However, instead of creating CMs, students use the same DSML to analyze and interpret given CMs—some functional but incomplete and some including bugs (see Figure 2b). Students assume the role of a collaborator working alongside the CM programmer, evaluating different CM versions and providing iterative feedback until they identify a fully functional CM.

In the EE-sim version, students focus on using simulations for investigations and analyzing data patterns to identify functionally correct simulations without engaging with the underlying code. Similar to the IE-CM version, students assume the role of a collaborator working alongside a simulation developer. They test the simulations, compare the outputs to their own expectations based on their conceptual understanding of runoff, and provide feedback as needed to the developer.

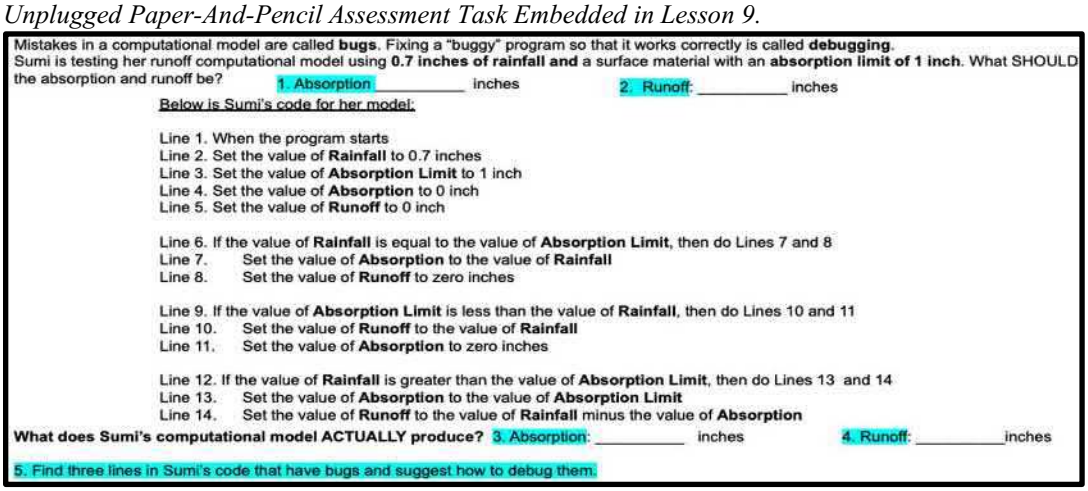
Embedded science + CT assessment

We developed an embedded assessment task with 5 items (see Figure 3) to be administered to students at the end of lesson 9 that measures specific science and CT proficiencies that we conjecture all students will have the opportunity to learn in lessons 1-9. Students are tasked with helping a hypothetical character named Sumi fix her code for modeling water runoff. Because errors in science-based CMs can stem from both science and CT, the first two assessment items assess students' proficiency with applying the conceptual runoff model to predict absorption and runoff for a given surface material (henceforth referred to as 'science proficiency'). Then, in assessment items 3 and 4, students are asked to interpret Sumi's buggy code and predict its output. Finally, in assessment item 5, students must identify three bugs in the code and propose fixes for each.

Sumi's code is presented using pseudocode so that students in all instructional versions, including EE-sim who have not been exposed to programming, can engage in this task. The pseudocode mirrors the runoff CM in that it includes the same set of programming concepts, such as variables, relational operators, and conditional statements. We designed the three errors or bugs to be in lines 9, 10, and 11, and pertain to science concepts instead of merely being syntactical errors or errors related to programming concepts only. The errors in Lines 10 and 11 are purely science-related (they are computationally accurate statements where the absorption and runoff variables are assigned to incorrect values), while the Line 9 bug is at the intersection of science and CT. Specifically, Line 9 should state "If Rainfall is less than Absorption Limit," Line 10 should be "Set the value of Absorption to the value of Rainfall," and Line 11 should say "Set the value of Runoff to 0 inches." This paper focuses on students' performance on different items of this embedded assessment task.

Figure 3

Unplugged Paper-And-Pencil Assessment Task Embedded in Lesson 9.


Mistakes in a computational model are called **bugs**. Fixing a "buggy" program so that it works correctly is called **debugging**.
Sumi is testing her runoff computational model using **0.7 inches of rainfall and** a surface material with an **absorption limit of 1 inch**. What SHOULD the absorption and runoff be? 1. Absorption _____ inches 2. Runoff: _____ inches

Below is Sumi's code for her model:

Line 1. When the program starts
Line 2. Set the value of **Rainfall** to 0.7 inches
Line 3. Set the value of **Absorption Limit** to 1 inch
Line 4. Set the value of **Absorption** to 0 inch
Line 5. Set the value of **Runoff** to 0 inch

Line 6. If the value of **Rainfall** is equal to the value of **Absorption Limit**, then do Lines 7 and 8
Line 7. Set the value of **Absorption** to the value of **Rainfall**
Line 8. Set the value of **Runoff** to zero inches

Line 9. If the value of **Absorption Limit** is less than the value of **Rainfall**, then do Lines 10 and 11
Line 10. Set the value of **Runoff** to the value of **Rainfall**
Line 11. Set the value of **Absorption** to zero inches

Line 12. If the value of **Rainfall** is greater than the value of **Absorption Limit**, then do Lines 13 and 14
Line 13. Set the value of **Absorption** to the value of **Absorption Limit**
Line 14. Set the value of **Runoff** to the value of **Rainfall** minus the value of **Absorption**

What does Sumi's computational model ACTUALLY produce? 3. Absorption: _____ inches 4. Runoff: _____inches

5. Find three lines in Sumi's code that have bugs and suggest how to debug them.

Participants, settings, and classroom implementation

We gathered data from 253 students from 5th and 6th grades who consented to participate in our research study. The students were distributed across five different schools within a large district in the southeastern United States. The five schools included a private Montessori school, a public charter school, and three public schools. Student demographics varied significantly across these schools. The charter and the Montessori schools were predominantly white with 82% and 58% of students identifying as white, respectively. Among the public schools, one had a predominantly Latine student population (72%), another was predominantly Black (58%), and the third comprised a diverse mix of Black (32%) and Latine (54%) students. The percentage of students eligible for free-lunch programs ranged from 30% to 81%, reflecting the socio-economic diversity of the schools. Additionally, the proportion of multilingual learners was higher in public schools, where more students required translation support. Within the five schools, we worked with six teachers (4 science teachers, 1 math teacher, and 1 art teacher) with varied teaching experience and STEM-related experiences. While teaching experience varied from 3 to 41 years, most teachers had limited prior experience with computer simulations and block or text-based coding. All six teachers identified as female, and out of the six, two teachers identified as white, two as African American, one as Latine, and one as mixed race (Latine and African American).

All students engaged with a single instructional version of the WRC unit, while each teacher taught two of the three instructional versions concurrently to different classes of students. All teachers participated in the same five-day professional learning (PL) workshop. Day 1 focused on Earth science concepts and Day 2 on engineering design activities. Days 3-5 were dedicated to the 3 CM versions with Day 3 focusing on EE-sim, Day 4 on IE-CM, and Day 5 on Construct-CM. This progression for the teacher workshop followed the Use-modify-create (Lee, 2011) progression commonly used for supporting students' programming. To further support teachers'

enactment of the WRC unit, we designed comprehensive resources, including instructional slides, a detailed teacher guide with lesson plans, and tutorial videos for the different computational modeling environments.

After the PL workshop, teachers were randomly assigned to 2 instructional versions each. Based on the school information, we divided the schools into two broad categories, each encompassing 3 teachers. The first category included the public schools with higher needs characterized by larger class sizes and language barriers. The Montessori and charter schools were included in the second category. We randomly assigned teachers (in a blocking fashion) to the instructional versions in a way that included teachers from each school category in each instructional condition. All 6 teachers implemented the WRC unit toward the end of the school year, with implementation spanning a range of 3-9 weeks depending on how much time teachers allotted to the unit weekly. Teachers enacted the embedded assessment task as a classroom activity by explaining the activity and then giving students some time to write their responses in their individual notebooks. We observed that teachers spent roughly 25-30 minutes on the assessment task.

Data sources, scoring, and analysis

Student responses to the embedded assessment task (see Figure 3) constitute our primary data source for this paper. Additionally, we administered a questionnaire asking students about their prior experience with programming. For the scoring of student responses to the five parts in the assessment task, we developed a rubric that assigned different codes to different categories of student responses for each sub-part. We not only coded for correct responses but also for different types of incorrect responses so that we could better characterize student difficulties with this integrated science-CT task. The codes were later converted into scores, for a maximum possible score of 13 points on the task (2 points for science proficiency, 2 points for code interpretation, 9 points for debugging). The coding team consisted of 4 researchers and was led by the first author of the paper who trained the team on the application of the rubric and helped resolve any questions and discrepancies in coding. The team trained by coding 10% of the student responses (randomly selected) together to establish initial consistency, and then coded another 20% to establish inter-rater reliability between pairs of coders (Cohen's κ at ≥ 0.94 for both pairs). Any discrepancies among coder pairs were discussed to reach a consensus on the codes. The remaining 70% of student responses were then randomly divided among the 4 graders and coded individually.

To examine differences in student performance on the three target constructs–science proficiency, code interpretation ability, and debugging skills–across curriculum versions, we conducted multiple regression analyses by controlling for students' prior coding experience, their teacher, and their grade level. To examine the relationships among the three target constructs, we also conducted Pearson's correlation tests to explore the mutual correlations, and hierarchical regression analyses to explore predictors of the constructs. For the hierarchical regressions, we ran simplified models to prevent biased results on account of the limited sample size and examined: (1) whether students' code interpretation ability is predicted by prior coding experience and/or science proficiency, and (2) whether debugging skills are predicted by prior coding experience, science proficiency, and/or code interpretation ability.

For our coding and analysis reported in this paper, we used a sample of 188 student responses because many students had skipped responding to this assessment task. Though the number of students in each instructional version was similar at the time of assignment, the IE-CM and Construct-CM versions lost disproportionately more students due to a lack of active consent for study participation and missing responses for the embedded assessment task. Students and their parents consented to participate in the research study before working on the WRC unit, hence lack of consent is not related to students' assigned instructional versions. Because both IE-CM and Construct-CM versions engage students in the programming CT practice (Weintrop et al., 2016), we combine the IE-CM and Construct-CM versions for the analysis reported in this paper and label them as the 'Programming' condition which is distinct from the EE-sim version or 'Non-programming' condition.

Findings

Student performance on the integrated science + CT task

Student performance on the integrated science-CT task (see Table 1) varied widely with scores ranging from 0 to 13 (max possible score), mean score of 7.71, and median score of 10. In general, most students (156 of 188) demonstrated science proficiency, correctly predicting the absorption (0.7 inches) and runoff (0 inches) in parts 1 and 2 of the task. However, less than half the students could interpret the given code accurately (81 of 188) and correctly state the output from the given code (0 inches of absorption and runoff). We found that several students (n=39) indicated that the code would produce 0 inches of absorption and 0.7 inches of runoff. We conjecture that at least some of these students recognized that the variable assignments for absorption and runoff were swapped in Lines 10 and 11 of the code, and believed that the bug would swap the values of absorption and runoff produced

by the code. We also found some evidence of the superbug error (Pea, 1986) where 11 students predicted that the code would produce 0.7 inches of absorption and 0 inches of runoff, consistent with the idea that the computer always produces the correct result.

For the debugging item, we found that only 52 students (28%) could correctly identify and fix the bug in Line 9, while 99 and 97 students correctly debugged Lines 10 and 11 respectively. Many more students were successful with debugging when the bugs were solely related to science understanding (Lines 10 and 11) and did not require understanding control structures and the flow of control in the CM. Further, our findings showed that 74 students believed there was a bug with Line 4, and suggested fixing the bug by setting the value of absorption to 0.7 instead of 0 inch. The ability to identify that absorption should be 0.7 and not 0 inches demonstrates students' science proficiency but indicates a lack of CT understanding about variable initialization and variable assignment and a tendency to hardcode variable values. This finding is consistent with programming challenges documented in the literature for young students (Kohn, 2017; Sorva, 2018).

Table 1

Student Performance on the Embedded Assessment Task Expressed as Mean (SD)

	All students (n = 188)	Non-programming condition: EE-sim (n = 104)	Programming conditions: IE-CM + Construct-CM (n = 84)
Science Proficiency (max=2)	1.72 (0.60)	1.69 (0.61)	1.75 (0.60)
Code Interpretation (max=2)	1.15 (0.83)	1.28 (0.83)	0.99 (0.80)
Debugging Skills (max=9)	4.84 (3.16)	5.62 (3.07)	3.88 (3.02)
Overall task performance (max=13)	7.71 (3.66)	8.59 (3.44)	6.62 (3.64)

Based on a regression model comparing the EE-Sim non-programming condition with the two programming conditions (Construct-CM and IE-CM), while controlling for students' prior coding experience, grade levels, and teachers, we found no significant difference in students' science proficiency between these groups (t = 1.27, p = .206, effect size = 0.09). This finding is consistent with our expectations based on the equality of the science instructional experience across conditions. However, students in the EE-sim condition scored significantly higher than students in the programming conditions on both code interpretation (t = 3.40, p < .001, effect size = 0.36) and debugging skills (t = 2.12, p = .035, effect size = 0.57). This result suggests that students did not need exposure to programming to successfully engage in code interpretation and debugging in a science-based CM. To further investigate these results, we explore the factors that predict student performance on the code interpretation and debugging constructs below.

Relations between science proficiency and computational modeling practices

We conducted Pearson's correlation tests among science proficiency, code interpretation, debugging skills, and prior coding experience. The results showed that science proficiency was statistically significantly correlated with code interpretation scores (r = .22, p = .004) and debugging skills (r = .38, p < .001). Additionally, code interpretation scores were significantly correlated with debugging skills (r = .15, p = .041). However, prior coding experience was not significantly correlated with science proficiency (r = -.05, p = .538), code interpretation (r = .01, p = .870), or debugging skills (r = -.10, p = .177).

Table 2

Results of Hierarchical Regression Analysis. Note: b Represents Unstandardized Estimates

Predictor	Code Interpretation (R^2= .05)				Debugging Skill (R^2 = .16)			
	b	SE	*t*	*p*	*b*	SE	*t*	*p*
Intercept	0.638	0.194	3.285	.001	1.592	0.709	2.246	.026
Prior coding experience	0.039	0.129	0.305	.761	-0.561	0.456	-1.232	.220
Science proficiency	***0.303***	***0.103***	***2.950***	***.004***	***1.904***	***0.373***	***5.102***	***<.001***
Code interpretation	-	-	-	-	0.284	0.269	1.057	.292

A hierarchical regression analysis (Table 2) showed that science proficiency was a significant predictor of code interpretation scores (p < .05), while prior coding experience is not. For debugging skills, only science proficiency emerged as a significant predictor (p < .05). Neither code interpretation scores nor prior coding experience are significant predictors of debugging skills by themselves. Both the correlational and regression analyses constitute evidence that students without prior coding experience or instructional experience creating or

interpreting code are able to perform well on code interpretation and debugging in the context of science-based CMs, as long as they have strong science knowledge and the tasks are in pseudocode (so that language-specific syntax is not required).

Discussion and future work

This study demonstrates how different approaches to CT-integrated science instruction (with and without programming practices) relate to fifth and sixth graders' performance on a paper-and-pencil science+CT integrated assessment task embedded within the curriculum. Our findings highlight the critical role of science proficiency in enabling students to successfully engage in CT practices such as code interpretation and debugging. This finding underscores the importance of domain knowledge in CT-integrated instruction. In the WRC unit, all students engaged with science modeling through multiple representations in Lessons 1-4—pictorial, mathematical, and simulation-based—in order to hone their science proficiency before engaging in computational modeling. Our findings align with previous studies showing a correlation between science and CT proficiency in CT-integrated science instruction (Basu et al., 2017; Rachmatullah & Wiebe, 2022). We conjecture that the role of science proficiency in predicting CT proficiency is more pronounced in our embedded assessment because the assessment was intentionally designed to include code with science-based errors.

Further, our findings also shed light on the potential role of programming in CT-integrated instruction. Specifically, we found that students in the simulation-only condition who did not engage in programming outperformed their peers in the programming conditions on items requiring interpretation and debugging of pseudocode. This result suggests that students with a strong science understanding can successfully engage with CT-integrated tasks that use pseudocode representations, even without programming experience. This result can be explained by a number of factors. For example, this finding may be specific to our assessment task where CT practices like debugging relied more on science conceptual understanding than knowledge of programming. It is also possible that students in the simulation-only condition may have started with greater science proficiency prior to WRC enactment, or become more proficient on science modeling concepts due to better classroom enactment of the science lessons by their teachers, and this higher overall science proficiency may have contributed to better performance on CT tasks in the embedded assessment. Another possible explanation is that being exposed to programming is not necessary for learning the CT practices of code interpretation and debugging when the practices are embedded in the context of pseudocode. With our current dataset alone, we are unable to pinpoint the contributing factor(s). As future work, we plan to analyze additional data sources from the larger research study such as data from students' pre-post summative assessment, students' curricular artifacts, and classroom enactment data to better understand the relations among students' prior science and CT expertise, and science and CT learning due to the WRC instructional versions.

Though our findings are currently limited to one embedded assessment task and are also limited by the unequal distribution of students in the three instructional versions, they have significant implications for science+CT instruction, teacher training, and integrated assessment design. In early grades, CT-integrated science learning can be achieved without requiring programming. Instead, the focus can be on engaging students in other CT practices, such as evaluating CMs, which may be more developmentally appropriate and impactful. For assessment design, our findings point to the need for intentional design that elicits student proficiency with science (domain knowledge) and CT separately and is accessible to all students, irrespective of their familiarity with the programming practice. From the perspective of teacher training, our findings emphasize the importance of preparing teachers to separately identify and address students' proficiencies and challenges with science and CT concepts and recognize the connections between science and CT learning. This insight can help teachers use integrated science+CT formative assessments more effectively and provide more individualized support to students to foster more effective learning experiences.

References

Basu, S., Biswas, G., & Kinnebrew, J. S. (2017). Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. User Modeling and User-Adapted Interaction, 27, 5-53.

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.

Hutchins, N. M., Biswas, G., Zhang, N., Snyder, C., Lédeczi, Á., & Maróti, M. (2020). Domain-specific modeling languages in computer-based learning environments: A systematic approach to support science learning through computational modeling. International Journal of Artificial Intelligence in Education, 30, 537-580.

Kohn, T. (2017, March). Variable evaluation: An exploration of novice programmers' understanding and common misconceptions. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 345-350).

Lee, I. (2024). The Decoding Approach to CT Integration [White Paper]. The Scheller Teacher Education Program (STEP) Lab Massachusetts Institute of Technology. https://education.mit.edu/wp-content/uploads/2024/03/The-Decoding-Approach-to-CT-Integration.pdf

Lee, I., Martin, F., Jill, D., Bob, C., Walter, A., Jeri, E., Joyce, M.-S., & Lina, W. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*(1), 32–37. https://doi.org/10.1145/1929887.1929902

Lytle, N., Cateté, V., Boulden, D., Dong, Y., Houchins, J., Milliken, A., Isvik, A., Bounajim, D., Wiebe, E., & Barnes, T. (2019). Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 395–401. https://doi.org/10.1145/3304221.3319786

McElhaney, K. W., Zhang, N., Basu, S., McBride, E., Biswas, G., & Chiu, J. L. (2020). Using computational modeling to integrate science and engineering curricular activities. In *In M. Gresalfi & IS Horn (Eds.). The Interdisciplinarity of the Learning Sciences, 14th International Conference of the Learning Sciences (ICLS) 2020* (Vol. 3).

McElhaney, K. W., Basu, S., McBride, E., Hutchins, N., & Biswas, G. (2023). Design and Implementation of a Week-long, High School Curriculum Unit Integrating Physics and Computational Modeling. In *Proceedings of the 17th International Conference of the Learning Sciences-ICLS 2023*, pp. 497-504. International Society of the Learning Sciences.

Melnik, R. (Ed.). (2015). Mathematical and computational modeling: With applications in natural and social sciences, engineering, and the arts. John Wiley & Sons.

National Research Council. (2012). A framework for K-12 science education: Practices, crosscutting concepts, and core ideas. National Academy of Sciences.

NGSS Lead States (2013). Next generation science standards: For states, by states.

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. Journal of educational computing research, 2(1), 25-36.

Rabinowitz, G., Lee, I., Gupta, P., & Chaffee, R. (2023). Deepening the Integration of Computational Thinking and Science Through Decoding in a Middle School Summer Program. In *Proceedings of the 17th International Conference of the Learning Sciences-ICLS 2023,* pp. 2243-2246. International Society of the Learning Sciences.

Rachmatullah, A., & Wiebe, E. N. (2022). Building a computational model of food webs: Impacts on middle school students' computational and systems thinking skills. Journal of Research in Science Teaching, 59(4), 585-618.

Schwarz, C. V., B. J. Reiser, E. A. Davis, L. Kenyon, A. Achér, D. Fortus, Y. Shwartz, B. Hug, and J. Krajcik. (2009). "Developing a Learning Progression for Scientific Modeling: Making Scientific Modeling Accessible and Meaningful for Learners." Journal of Research in Science Teaching 46 (6): 632–654.

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

Sorva, J. (2018). Misconceptions and the beginner programmer. Computer science education: Perspectives on teaching and learning in school, 171.

Waterman, K. P., Goldsmith, L., & Pasquale, M. (2020). Integrating computational thinking into elementary science curriculum: An examination of activities that support students' computational thinking in the service of disciplinary learning. Journal of Science Education and Technology, 29(1), 53-64.

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, 127-147.

Windschitl, M., J. Thompson, and M. Braaten. 2008. "Beyond the Scientific Method: Model-Based Inquiry as a New Paradigm of Preference for School Science Investigations." Science Education 92 (5): 941–967.

Xie, C., Schimpf, C., Chao, J., Nourian, S., & Massicotte, J. (2018). Learning and teaching engineering design through modeling and simulation on a CAD platform. Computer Applications in Engineering Education, 26(4), 824-840.

Acknowledgments

We acknowledge the support of National Science Foundation DRL #2055609 and #2055597. We thank the students and teachers who participated in the study and appreciate ISLS reviewer insights on this paper.