A Comparison of Computational Practices and Student Challenges Across Three Types of Computational Modeling Activities Integrating Science and Engineering

Satabdi Basu, SRI International, satabdi.basu@sri.com
Arif Rachmatullah, SRI International, arif.rachmatullah@sri.com
Kevin McElhaney, Digital Promise, kmcelhaney@digitalpromise.org
Nonye Alozie, SRI International, maggie.alozie@sri.com
Hui Yang, SRI International, hui.yang@sri.com
Nicole Hutchins, University of Florida, nicole.hutchins@ufl.edu
Gautam Biswas, Vanderbilt University, gautam.biswas@vanderbilt.edu
Kelly Mills, Digital Promise, kmills@digitalpromise.org

Abstract: Computational models (CMs) offer pre-college students opportunities to integrate STEM disciplines with computational thinking (CT) in ways that reflect authentic STEM practice. However, not all STEM teachers and students are prepared to teach or learn programming skills required to construct CMs. To help broaden participation in computing and reduce the potentially prohibitive demands of learning programming, we propose alternate versions of computational modeling that require low or no programming. These versions rely on code comprehension and evaluation of given code and simulations instead of code creation. We present results from a pilot study that explores student engagement with CT practices and student challenges in three types of computational modeling activities.

Introduction and rationale

Over the past decade, numerous science, technology, engineering, and mathematics (STEM) education research initiatives and reforms have aimed to prepare K-12 students with the computational knowledge and skills critical for engaging in authentic STEM practices and applying scientific knowledge to the solution of real-world problems (National Research Council [NRC], 2012). Computational models (CMs) are widely recognized as an effective means to forge deep connections between Computational Thinking (CT) and STEM disciplines by providing explicit mechanisms for constructing and visualizing scientific phenomena (e.g., Sengupta et al., 2013; Lee et al., 2020). A 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). However, developing scientific CMs by learning programming along with disciplinary content poses challenges for students, particularly younger ones (Sengupta et al., 2013), and also poses challenges for teachers who are not trained in programming instruction, often affecting their confidence and increasing their time needed for preparation and teaching (Langbeheim et al., 2020).

To address these challenges with developing CMs, our study presents two additional types of computational modeling activities: (1) evaluating CMs (instead of developing them), and (2) evaluating simulations (without viewing the simulation code). Each type of computational modeling emphasizes different CT practices (Weintrop et al., 2016). Using these three types of computational modeling (developing CMs, evaluating CMs, evaluating simulations), we have developed three versions of a STEM+computing curriculum that integrates earth science, engineering and CT for fifth and sixth grade students. In this paper, we describe a pilot study using these three curricular versions, examine how students engage in computational practices in each of the versions, and identify what challenges, if any, they encounter in each version.

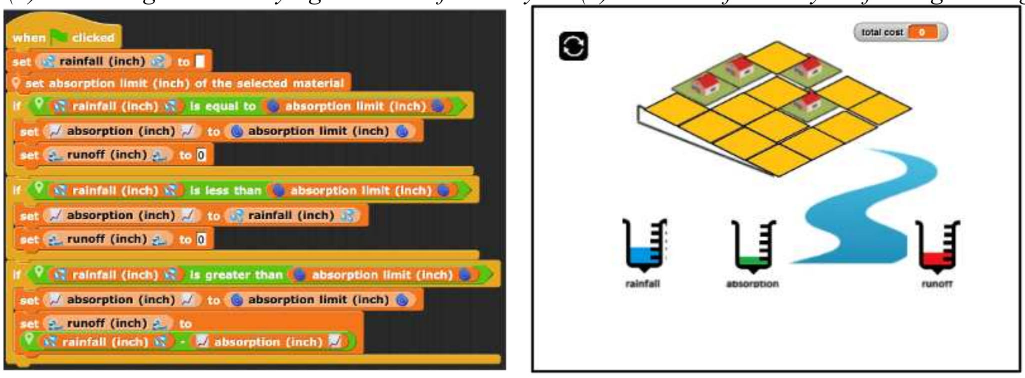
While CM development is an established way to integrate CT with STEM disciplines, a few alternate approaches have been studied. For instance, in 'decoding' activities (Rabinowitz et al., 2023), students elucidate connections between provided code and the represented scientific processes, thus fostering 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. Research also demonstrates the benefits of learning using simulations such as conceptual understanding, inquiry skills, and knowledge acquisition (e.g., de Jong et al., 2018). However, the relative affordances of evaluating CMs or simulations relative to using or constructing them have not yet been investigated. Determining the unique affordances of each type of computational modeling for students' science, engineering, and CT learning is critical to enabling educators to implement STEM+computing instruction in ways that meet teachers' and students' needs.

Curricular context: Three variations of the Water Runoff Challenge (WRC)

We adapt an instructional design framework for integrating science, engineering, and CT using CMs (McElhaney et al., 2020). In this approach, an engineering design challenge is defined such that its solution is governed by a science phenomenon. Students must model the phenomenon computationally so that the CM can be used to design, test, and optimize solutions to the challenge. In the Water Runoff Challenge (WRC) unit, students redesign a schoolyard to reduce water runoff while meeting criteria related to cost, accessibility, and usage. Each unit variation spans 12 45-minute lessons where students first investigate water runoff, then engage in CM activities to either construct or identify a functional water runoff CM, and finally use a simulation based on the CM to design the schoolyard. The CT lessons engage students in either constructing CMs (Construct-CM); interpreting and evaluating CMs (IE-CM); or exploring and evaluating simulations (EE-sim).

In the Construct-CM version, students engage in unplugged activities to learn requisite CT concepts such as sequences, variables, and conditionals, then construct their CMs incrementally using a block-based, domain-specific modeling language (Figure 1a). The IE-CM version reduces the programming demands by having students interpret and analyze code (sometimes incomplete or buggy) instead of creating it. Students still engage in unplugged activities to learn the needed CT concepts. The EE-sim version involves no programming and hides the simulation code from students. Instead, students conduct experiments with the simulation and find patterns in the generated data to identify a functionally correct simulation. In the IE-CM and EE-sim versions, students assume the role of someone who works adjacent to the CM developer, iteratively providing feedback to the developer based on their evaluation of different versions of the CM until they identify a fully functional CM.

Figure 1
(a) A Working CM underlying one Grid of Schoolyard (b) 16 Grids of Schoolyard for Engineering Design



```
when clicked
set rainfall (inch) to
set absorption limit (inch) of the selected material
if rainfall (inch) is equal to absorption limit (inch)
    set absorption (inch) to absorption limit (inch)
    set runoff (inch) to 0
if rainfall (inch) is less than absorption limit (inch)
    set absorption (inch) to rainfall (inch)
    set runoff (inch) to 0
if rainfall (inch) is greater than absorption limit (inch)
    set absorption (inch) to absorption limit (inch)
    set runoff (inch) to
        rainfall (inch) - absorption (inch)
```

(a) (b)

All three versions of CM activities engage students in the CT practices (Weintrop et al., 2016) of generating and analyzing data by testing CMs and simulations and looking for patterns in the generated data to determine condition(s) under which the CM/or simulation works. All three versions also engage students in the CT practice of assessing CMs (either their own or one that is provided). Both Construct-CM and IE-CM engage students in the CT practice of programming. Only the Construct-CM version addresses the CT practices of designing CMs and constructing CMs. Construct-CM also engages students in the practice of troubleshooting and debugging as they develop their CMs, though all three versions include a pseudocode debugging activity, thereby engaging all students with the practice of debugging. In this activity, students are asked to interpret a 14-line pseudocode segment, identify three lines with bugs and suggest how to fix them.

Methods

We conducted a small pilot study using a cognitive interview approach (Willis, 2005) with 15 fifth and sixth grade students recruited individually in the Western U.S. Each student worked one-on-one with a researcher on four WRC lessons for a total of 3-4 hours. Each interview focused on either science, engineering, or one of the three CT versions. In this paper, we present analyses of nine interviews (seven fifth graders and two sixth graders) that involved the CM lessons. Three students in the Construct-CM condition and two in IE-CM reported having prior programming experience using Scratch, Code.org lessons, or Roblox. Students were randomly assigned to one of the three CM versions such that three students worked on each version. Because the students started with the CM lessons and did not engage with the initial science lessons, they were given a brief introduction to the runoff problem and the relations between rainfall, absorption, and runoff. As students engaged in various CM activities, they were periodically prompted by a researcher to think aloud and explain their reasoning. All interviews were conducted virtually, and video recorded along with recordings of students' screens. For each interview, we conducted a qualitative thematic analysis to document the nature of students' engagement with the CT practices delineated by Weintrop et al. (2016) and any challenges the students faced.

Findings

All nine students engaged effectively with the activities across the three CM versions and reported finding the activities interesting. Students working with the EE-sim and IE-CM versions found the activities to be straightforward. Construct-CM students found it challenging when first constructing their CMs, but reported that the lessons were easier once they had developed the first part of the CM. We found students' engagement with the CT practices to generally align with our expectations based on our design of the CM versions. Below, we elaborate on students' engagement with the practices of programming, troubleshooting and debugging, and assessing CMs, where the differences among CM versions were most pronounced based on our design.

Students in the EE-Sim version did not engage in the **programming practice** because they could not view the code. Students in IE-CM and Construct-CM engaged with the same programming concepts (sequences, variables, conditionals, expressions) and same block-based representation, but their engagement varied. All three Construct-CM students demonstrated a strong understanding of the target programming concepts during the initial unplugged activity in a discipline-agnostic context. But, they experienced challenges in applying this understanding to represent the relationships among rainfall, absorption and runoff while constructing their runoff CMs. For example, while programming the condition where rainfall is equal to the absorption limit of the surface material, one student was unsure how to represent the condition using an "if" block, where in the program sequence to place the "if" block, and how to program variable update statements. In contrast, IE-CM students were able to easily apply their understanding of programming concepts from a discipline-agnostic unplugged context to a water runoff context. These students could correctly interpret given code segments, predict code output, and iteratively identify a functional CM. We found only one instance where a student asked for help interpreting a variable assignment statement.

As noted above, only students in the Construct-CM version engaged in the **troubleshooting and debugging practice** as part of their runoff modeling activities. However, all students engaged in this practice as part of the unplugged pseudocode debugging activity at the end of the CM lessons. We found that Construct-CM students were more successful with this practice when debugging their own CMs compared to debugging pseudocode written by others. For the pseudocode debugging activity, we hypothesized that the Construct-CM students would be at an advantage after having engaged with debugging during CM development, and that the activity would be particularly difficult for EE-sim students who had not seen code or pseudocode previously. However, while the perceived difficulty of the activity varied (EE-sim students rated the activity to be hard compared to other students who claimed to find the activity easy or of moderate difficulty), there was no discernable difference in activity performance among students in the different versions. Most of the nine students across the three WRC versions found the pseudocode debugging activity challenging. Their difficulty with predicting the output of the pseudocode, indicates that the debugging challenges stem, among other things, from challenges with code comprehension (the programming practice).

Students in all three CM versions had opportunities to engage in the **assessing CMs practice** by analyzing data generated from the CMs/simulations. IE-CM students also assessed CMs by interpreting given code in unplugged contexts. Similar to the debugging practice, we found that students were most successful with this practice when they assessed the CMs they were constructing themselves (Construct-CM version) versus CMs created by others (IE-CM and EE-sim versions). In the IE-CM and EE-sim versions, we found that students were able to determine whether CMs provided to them matched the runoff phenomenon being modeled, but they needed support in determining reasons for any mismatches. In particular, for the EE-sim version, students were able to compare data generated by a buggy CM to data that should be produced by a working CM, but they were often not able to use the data to determine the underlying reasons for why a simulation was not working correctly. Additionally, we found that IE-CM students were better able to assess CMs when they had access to the computational modeling environment and could use data analysis to inform their evaluation, compared to when they had to assess CMs in an unplugged context using code comprehension alone.

Discussion and conclusion

Our study demonstrates approaches for STEM+CT integration that involve low or no programming. These approaches leverage research on scientific discovery learning using simulations and modeling tools (de Jong et al., 2018). In such research, learners typically interact with functional simulations with the goal of inferring the scientific model underlying the simulation. In contrast, we provide students with incomplete or incorrect simulations that are simple enough for them to evaluate by comparing simulation results against their already existing conceptual understanding of scientific relationships. Presenting multiple approaches for STEM+CT integration with low or no programming requirements can provide accessible pathways for integrating CT into STEM teaching and learning. This can, in turn, result in increased willingness among teachers to incorporate CT in STEM education, potentially expanding the reach of CT to broader student and teacher populations.

Proposing new approaches for STEM+CT integration requires examining students' learning processes and challenges in such activities so that learning environments can adequately support student learning. Our pilot study revealed unique affordances and challenges associated with three types of CM activities that engage students in CT practices differently. Our findings revealed that programming was one of the most challenging CT practices for students. Hence, practices that relied on programming in any way (constructing CMs, debugging CMs by looking at pseudocode only, assessing CMs by looking at code only) were also challenging for students. Students found it easier to engage in CT practices in a CM environment versus an unplugged context where they had to rely on programming alone. Not surprisingly, EE-sim students who could not see the code, faced the fewest challenges with CT practices overall. However, when students programmed their own CMs, they found it easier to engage in assessing and debugging those CMs compared to CMs created by others.

However, the relative affordances of the three CM-based approaches for integrated STEM+CT teaching and learning need to be explored further. For example, learning programming may be challenging, but does it offer benefits for science and engineering learning, and if so, who benefits? Building on the insights from this pilot study, we are refining our curriculum materials and the teacher resources to equip teachers with strategies to better support student learning and assist students in navigating the identified challenges. Additionally, we are updating the student facing materials—incorporating additional scaffolding and modifying some of the code-based representations to pseudocode-like plain language—to facilitate deeper engagement with the CT practices introduced. Although the three versions of the WRC unit demonstrate promise for integrating CT with STEM education in novel ways, the specific effects on students' CT, science, and engineering learning outcomes need to be systematically studied. Our next steps include collecting data on students' disciplinary learning across the different CM versions, as well as understanding the dynamics of teacher confidence and instructional methods when implementing these three curriculum versions.

References

de Jong, T., Lazonder, A., Pedaste, M., & Zacharia, Z. (2018). Simulations, games, and modeling tools for learning. In the International handbook of the learning sciences (pp. 256-266). Routledge.

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.

Lee, I., Grover, S., Martin, F., Pillai, S., & Malyn-Smith, J. (2020). Computational thinking from a disciplinary perspective: Integrating computational thinking in K-12 science, technology, engineering, and mathematics education. *Journal of Science Education and Technology*, *29*, 1-8.

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 M. Gresalfi & I.S. Horn (Eds.). Proceedings of the 14th International Conference of the Learning Sciences (ICLS) 2020, Volume 3 (pp. 1357-1364). International Society of the Learning Sciences: Nashville, TN.

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 Academies Press.

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.

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.

Willis, G. B. (2015). *Analysis of the cognitive interview in questionnaire design*. Oxford University Press.

Acknowledgments

We acknowledge the support of National Science Foundation DRL #2055609 and #2055597. We also thank the students who participated in our study and the anonymous reviewers whose suggestions improved our paper.