From 'Use' to 'Choose': Scaffolding CT Curricula and Exploring Student Choices while Programming (Practical Report)

Nicholas Lytle
Veronica Catete
nalytle@ncsu.edu
vmcatete@ncsu.edu
North Carolina State University
Raleigh, North Carolina

Amy Isvik
Danielle Boulden
Yihuan Dong
North Carolina State University
Raleigh, North Carolina

Eric Wiebe
Tiffany Barnes
wiebe@ncsu.edu
tmbarnes@ncsu.edu
North Carolina State University
Raleigh, North Carolina

ABSTRACT

As computing skills become necessary for 21st-century students, infused computational thinking (CT) lessons must be created for core courses to truly provide computing education for all. This will bring challenges as students will have widely varying experience and programming ability. Additionally, STEM teachers might have little experience teaching CT and instructing using unfamiliar technology might create discomfort. We present a design pattern for infused CT assignments that scaffold students and teachers into block-based programming environments. Beginning with existing code, students and teachers work together 'Using' and comprehending code before 'Modifying' it together to fix their programs. The activity ends with students 'Choosing' their own extensions from a pre-set list. We present a comparison of two implementations of a simulation activity, one ending with student choosing how to extend their models and one having all students create the same option. Through triangulating data from classroom observations, student feedback, teacher interviews, and programming interaction logs, we present support for student and teacher preference of the 'Student-Choice' model. We end with recommended strategies for developing curricula that follow our design model.

CCS CONCEPTS

• **Social and professional topics → Computational thinking**; *K-12 education*;

KEYWORDS

Use-Modify-Create, Lesson Design, Student Choice

ACM Reference Format:
Nicholas Lytle, Veronica Catete, Amy Isvik, Danielle Boulden, Yihuan Dong, Eric Wiebe, and Tiffany Barnes. 2019. From 'Use' to 'Choose': Scaffolding CT Curricula and Exploring Student Choices while Programming (Practical Report). In *Workshop in Primary and Secondary Computing Education (WiPSCE '19), October 23–25, 2019, Glasgow, Scotland Uk.* ACM, New York, NY, USA, Article , 6 pages.

WiPSCE '19, October 23–25, 2019, Glasgow, Scotland Uk

ACM ISBN XXX…$15.00
DOI: 10.1145/3361721.3362110

1 INTRODUCTION

It is becoming increasingly necessary for every child to have experience with 21st-century Computational Thinking (CT) skills [28]. However, these skills have typically been taught within elective Computer Science classes or outside of school activities [17]. To reach *all* students, CT must be integrated into required K-12 courses. This type of CT integration poses several challenges. First, lessons must not only focus on key CT concepts but must also integrate domain knowledge. Further, these activities must be designed with the understanding that they may be the first introduction to programming for many students. Successful integration depends on equipped and capable teachers, though many do not feel they have the prerequisite background required to teach computing [6]. Professional development gives teachers much needed experience with CT skills [22], however, to reach all students, we must develop solutions that can be readily adopted by both experienced and inexperienced teachers while keeping students actively engaged.

This report investigates the iterative design of a 4-day computing-infused curricula developed for a grade 6 middle-school science classroom. Through multiple iterations of content refinement, we have identified the need for integrating student-choice through differentiated options in our curricula. This allows students with varying levels of prior programming experience to complete the assignment to a level of their own choosing, focusing on the aspects that interest them, thus fostering their interest in STEM and computing. This design also balances the teachers need to have a sense of control/comfort over the classroom in order to continue the adoption of CT and computing into their core content classroom.

This paper reviews related initiatives and our prior work and findings on developing computing infused curriculum for science classrooms. The paper then describes our latest curriculum feature on adding student choice, investigating both student programming behavior and teacher affect. Our data is triangulated using a combination of code traces, exit tickets, and classroom observations [16]. We further support our findings with post-hoc teacher interviews.

2 RELATED WORK

CT and the use of computational tools has been shown to enable deeper learning of STEM content areas for students [5] as well as increase student retention and learning performance in computing courses and outreach programs [4, 9]. Middle grades has been identified as a critical age range to study the potential for developing CT. In this age-range, block-based programming curricula [11], and

visual programming environments have been shown to improve student performance and affect[21].

Many introductory activities are motivated at the pedagogical and psychological level by constructionism [19], and self-determination theory [7]. Many popular computing education tools today like Scratch [24] use “Wide-Wall" learning environments motivated by constructionist philosophy to give students an environment that enables them to create anything. These environments have been shown to increase student engagement in their learning [12], which can be further explained by Self-Determination Theory, specifically the three needs of autonomy, competency, and relatedness required for developing intrinsic motivation in students [7]. Freedom of choice in constructionist learning environments allows for greater development of autonomy and competency, and sharing of personal artifacts in the social setting of the classroom can help develop relatedness. While popular and motivated, these fully open-ended create environments may be hard to implement in practice [23].

Although, infusing CT directly into a STEM course improves the teacher’s mastery of their disciplinary concepts with new instructional approaches [27], in courses where students solve open-ended problems, teachers lose much of the control they traditionally have over the learning process and may become uncomfortable [5]. Teacher’s self-efficacy and discomfort are important factors to consider when helping core subject teachers add computing into their classes. According to Pajares, efficacy beliefs help determine how much effort people will spend on an activity, how long they will persevere when confronting obstacles, and how resilient they will prove in the face of adverse situations - the higher the sense of efficacy, the greater the effort, persistence, and resilience [18]. Research has shown that a teacher’s self-efficacy is closely linked to their level of adventurousness in teaching. As research by Frykholm suggests, a teachers self-efficacy in combination with tolerance for discomfort will affect their willingness to adopt new materials and curricula [8]. In order to better support teachers, they not only need ‘more training’ but also curricular resources that promote and scaffold their ability to adopt new teaching practices.

One promising avenue to improved support resources for teachers is curricular materials that follow a scaffolded intensity of interaction. Research by Lee et al. suggests that using a Use-Modify-Create (UMC) learning progression can promote the acquisition and development of CT while also limiting the anxiety from activities that teachers may have previously perceived to be “too hard" for students [13]. Sentance et al. takes this a step further by specifying learning practices to engage in at the Use level. Their new model, Predict, Run, Investigate, Modify, Make (PRIMM) “incorporates activities that scaffold learning for students and provides a structure for lessons" [25]. By dedicating a portion of the lesson on reading and planning before modifying code, students can develop a sense of what the code is doing. As Lister shows, a student’s ability to accurately follow code is highly correlated with student confidence to independently write code [14]. The next step, modify is used by both Sentance and Lee and is a scaffolded way to encourage small code changes, or focus on recognizing misconceptions between code logic and world logic [10]. The Create step of these two frameworks is still open-ended and can leave stress for teachers. Teachers often find it difficult to facilitate the potentially limitless possibilities students may choose and students also find difficulty reasonably scoping a project within their level of possible ability [23]. In order to alleviate both students lack of direction and teachers inability to support all extensions, our prior findings suggest implementing a more limiting Choose phase to the curricula. The sections below describe our intervention.

3 CONTEXT

3.1 Background

In our prior work developing infused computing curricula for STEM courses, we found that the largest differentiator in both teacher and student engagement in our integrated science-CT curricula was the degree to which teachers embraced a learner and facilitator role [3]. Teachers who became comfortable with open-ended assignments with diverse solutions were able to promote student engagement in the curricula. Having these open-ended assignments initially also created challenges as lower-skilled students struggled without having some support on how to use the environment. Conversely, higher-performing students needed differentiated challenges as finishing their tasks early lead many to engage in ‘off-task’ behavior.

Based on these results, we set two primary goals for our next years implementation. One was to create lesson designs that better eased novice students and teachers into programming environments. The second was to create lesson designs that promoted student self-guided exploration, differentiated challenges, and scaffolded teachers adopting a facilitator role.

In follow-up work, we adapted the “Use-Modify-Create" (UMC) model [13] to create an assignment design that scaffolded students into developing simulations. Teachers lead students in 4 days of an infused lesson in which they first do an unplugged activity [2], then use code, modify it, and finally create their own code. In an experimental study, teachers and students found the UMC version to be more engaging and less difficult than students who created code all three days[15]. While successful at scaffolding students, we found having all students program the same thing did not afford the goal of having open project-based learning. We describe the original curriculum below then detail our changes in section 4.1.

3.2 Curriculum

We designed a 4-day, CT lesson about Food Webs - a scientific topic for 6th grade students. In the food web curriculum, students learn about how energy is transferred from producers to primary and secondary consumers. They do this through developing a simulation in a block-based environment, Cellular [1], an extension of the block-based programming language Snap! [9]. We describe each daily segment of the activity below.

Day 1 - The curriculum began with an “Unplugged" activity [2] in which students reviewed definitions and components of a Food Web (e.g. consumer types, energy transfer). This ended with students completing a worksheet, lead by the instructor, describing the behavior of agents in the model through pseudo-code. This prepared students for developing these logic models within the programming environment.

Day 2 - Focused on the “Plant" agent (producer), which grows based on the solar energy given by the “Sun". Students had plant and sun code provided and *used* and read through the working code in order to become familiar with the different conditions. The

Figure 1: The Food Webs Cellular Stage with each animal 'Choice' present.

teacher led students in exploration by changing the initial input (the solar energy intensity), the cutoff conditions (how much energy is needed to transition) and the amount of energy lost through transitioning. Students recorded how those changes affected the speed in which flowers changed state on a worksheet.

Day 3 - Focused on the "Bunny" agent (primary consumer). Some bunny code was provided at the outset, though the given bunny behavior did not conform with their idea of the actual model (e.g. bunnies never ate when they got low on energy). Teachers led the students in *modifying* the existing code in order to make it conform to the existing ideas they had discussed on Day 1.

Day 4 - Focused on *Create* in the UMC model. Teachers lead the class through how to develop the "Fox" code, the model's secondary consumer, with students following along. The class ended with students modifying the initial conditions, bunny and fox code at-will in order to determine how it affected the final model.

This version of the curricula (further referred to as **'Fox-Only'**) was successful at easing students and teachers into the programming environment and into coding [15]. However, students on the final day all code the *same* feature lead by the teacher. As this moves us away from more project-based, student-guided learning, we adapted this curricula to afford student-choice.

4 METHODS

4.1 Adding Student Choice

We modified the final day of the activity to promote student self-guided exploration and move teachers towards acting as facilitators. This curricula, further referred to as **'Student Choice'** substituted the activity where students changed variables in the simulation on Day 4 with time for students to make extensions from a set of options. Each student was given a piece of paper with a list of possible extensions. The first two options were to extend the functionality of the *bunny* or *fox* code to include the new feature of reproduction. The third option was to extend the *sun* code such that every couple of days, the weather changed. The final three options were adding new agents to the Food Web: another herbivore - *mouse*, another carnivore - *eagle*, or an omnivore -*bear*. These animal options are shown in Figure 1.

We developed these extensions by taking into account prior student requests for code extensions in previous implementations of the curricula. We ensured that each choice had a meaningful connection to the topic content and that each addition to the student's model would change the food web (new animals interacting with each other/weather changing the plants etc.) in a manner to promote scientific inquiry. Every new animal code had the same programming tasks shown in Figure 2 as the Fox, differing only in what type of food they ate based on their consumer type (feature 4). In this way, students could develop a complex simulation (with different actors interacting) without having to learn many more code blocks. In a manner similar to a Parson's problem [20], we listed the necessary blocks for completing the code for each option to ease student cognitive load [26].

Our intent was to promote teachers acting as facilitators during programming sessions. With this assignment design, students were more likely to have widely varying code, meaning teachers could not employ the standard teaching strategy of instructing at the front, forcing them to act as facilitators. However, teachers were supported in the fact that student options are limited to a pre-set list. Each teacher would also be given a "Cheat-Sheet" showing what each of the different animal codes should look like.

4.2 Implementation/Data Collection

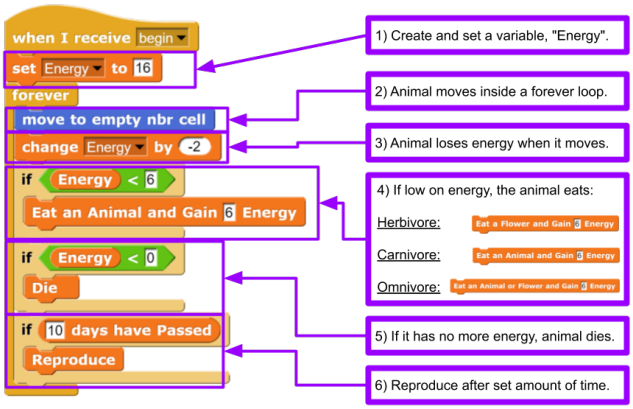

Figure 2: Final Fox Code with one extension choice (Reproduction). Teachers lead students in programming features 1 through 5. Every animal choice had similar code except for Feature 4 where animals eat different things based on their agent type (e.g. herbivores eat plants).

In order to evaluate the differences between our original Fox-Only and new Student-Choice versions, we tested both versions in a quasi-experimental design. Two 6th-grade science teachers from the same school were recruited to teach both the Food Webs curriculum. Neither had prior experience teaching the activity. Teachers were trained on each curricula prior to implementation. In the Winter, both teachers taught one class each using the Fox-Only

model. 3 months later, in the Spring, they each taught one class using the Student-Choice model. Each classroom contained around 25 students, though we only have consenting information from 13 and 15 from Spring (28 total) and 17 and 21 from Winter (38 total) respectively. Teachers instructed the curricula while researchers observed, but the research team made themselves available if teachers or students needed programming help. All class periods were the exact same length: 50 minutes.

The initiatives were evaluated through a triangulation of multiple data sets including student programming traces, student exit-tickets, observations, and semi-structured interviews with teachers [16]. For every period, at least one research team member was present taking observation notes, focusing on the students' interactions within the environment as well as how the teacher was teaching the lesson. After the conclusion of each day's activity, students took an end-of-activity "Exit Ticket" in which they answered a series of questions about the activity. As we were focused on difficulty perceptions between curricula, we focus only on student responses to the question, "Please use the following scale to rate how difficult or easy the lesson was today." Responses were on a 1-5 Likert scale ranging from Very Easy to Very Difficult. After both implementations concluded, we interviewed teachers in a group semi-structured interview asking about their experience teaching both versions of the curricula.

In addition, every consenting student's interaction data within the programming environment was recorded. From this, we were able to collect a series of code 'snapshots' for each student, what student programs looked like at any given point in time during the activity. The series of snapshots in order produces a student's code trace for the assignment. We focus primarily on student code traces for the fourth day as this is where the assignments differed. For each of the requirements for the Fox code as well as for all of the possible choice options on Day 4, we developed auto-grading code to determine if a given student snapshot had completed a task or not. We then ran this auto-grader on each snapshot in each student code trace to determine *what* choices students made in developing their simulations and *when* they finished each choice option.

5 RESULTS

5.1 Classroom Observations

In all 4 classrooms, teachers spent the initial half of the classroom leading students in the creation of the Fox. There were instances during each session of teachers forgetting a step, but this did not greatly impede leading the classroom and all teachers were able to correct. After walking the students through each of the steps of the Fox code, the teachers then walked around the room helping individual students with the respective next part of the assignment. For the Fox-Only condition, this involved debugging some students' code and answering questions about different run conditions. For Student-Choice, teachers walked around the room helping individual students. Teachers did not seem to have difficulty handling the variety of student choices. Both, in fact, elected to not use their cheat sheet, instead just memorizing the differences each animal choice had with the Fox code they already knew how to do.

Similar to our previous studies, researchers observed many students who finished early in the Fox-Only condition engaging in off-task behaviors. This would happen for upwards of 20 minutes as students who finished the Fox tended to do so with plenty of time remaining. This 'off-task behavior' was rarely seen in the Student-Choice classrooms. Researchers observed many instances of students trying to get other students, their teachers, or even the researchers to see what progress they had made in their environment. As no students had finished all the features, both classes doing the 'Student-Choice' condition asked their teachers if they could continue working on this the following day.

5.2 Student Programming Data

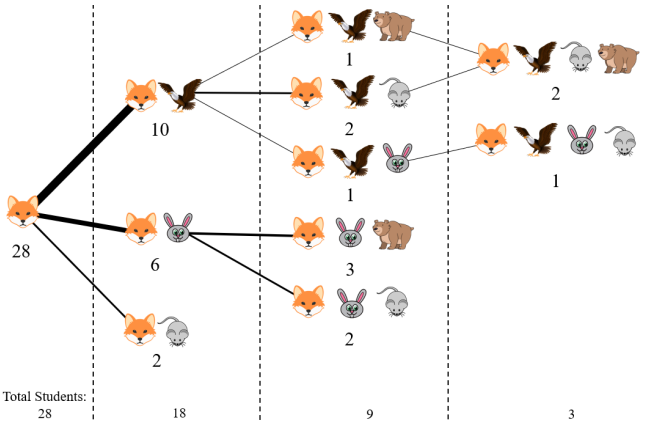

Figure 3: The code trace of how students choose to implement each extension option.

We analyzed 66 student traces - 38 from those who did Fox-Only and 28 from those who had the Student-Choice version. As both had to complete the Fox code before their conditions differed, we examined their Fox code completion rate. The students in the Choice condition had a higher rate of completing the Fox code (15/28) than the Fox-Only Condition (15/38) though a Chi-Square with Yates correction does not find this to be significant ($p = .38$).

We then turn to the programming patterns of the 28 students in the Student-Choice condition beginning with the 10 who made no extension. In stepping through the code trace data, 4 of these students did complete the fox code and finished with much time to spare (20 minutes left in the class on average). However, these students elected to instead tweak Fox code, switch the number of foxes in the world, and try different run-time conditions for testing out their simulations. The other 6 spent the entire time attempting, but not completing, the Fox code. Most of these students had difficulty completing tasks relating to the "Energy" variable including setting it in the beginning, and using it to check if the animal should eat or die.

We found 30 extensions developed by 18 students who made at least one. All of these were animals (no one chose the weather task). 7 students extended Bunny Code from Day 3, 10 students elected to make Eagles, 8 worked on Mice, and 5 made Bears. Of the 18 students that made an extension, half (9) programmed 2 or more additional animals and 3 programmed 3 additional. Students in the "Choose" activity took a variety of paths towards developing their

simulations. We outline this in Figure 3. Here, the number under each state represents how many students attempted to complete that animal and each edge describes a transition to a different simulation. Here, we can see that the most common first additions were to add Eagle (10 students) or Bunny (6 students).

5.3 Student-Perceived Difficulty

On each of the four days for both initiatives, students were asked to fill out an end-of-day exit ticket. 31 of the 38 consenting students in Fox-Only responded for all 4 days, though only 15 of the 28 consenting students in Condition 2 did. Students were asked each day how difficult they thought the lesson was. A Friedman Test for the responses for the Fox-Only condition does not give a significant difference in student perceived difficulty across the 4 days ($p = .10$). This corroborates our previous study which found no perceivable difference in difficulty across days[15]. For the Choose condition, a Friedman test finds no significant difference in difficulty as well ($p = .92$). Mann-Whitney U Tests also find no significant differences in difficulty responses *between* conditions in Days 1 ($p = .22$), 2 ($p = .34$) 3 ($p = .52$) or 4 ($p = .91$).

5.4 Teacher Interviews

An end-of-initiative semi-structured group interview with teachers was conducted to understand their thoughts on how the two implementations compared. Both teachers felt that the UMC progression found in the curricula was beneficial for students, especially since many had never used the programming environment before. They did suggest that the 'Unplugged' and 'Use' day be combined as they found those the easiest to get through and finished both early.

Both teachers expressed preference for the Student-Choice condition saying it was "cool that [students] could decide how many of each thing they could add". They thought the students were more engaged in the Choice condition remarking that "[students] kept running up to me the whole class, 'look, look what mine's doing!' 'Look what I've got.' They were pretty excited about it." The Choice condition, having more species in the simulation, also better aligned with scientific standards. A Food **Web** is defined as having multiple actors of different types interacting (i.e. multiple primary/secondary consumers) while a Food **Chain**, only has one of each type, like the Fox-Only condition. Teachers "really liked [the Choice condition] how it was more like a food web than a food chain where you add a whole bunch of other little things." When asked, both teachers said they would do the choice version again.

Neither teacher expressed discomfort with acting as a facilitator for individual student projects even if they didn't have an answer for a student question immediately. A teacher remarked that she was able to answer questions "most times" with only "one I couldn't figure out but then one of the other kids figured it out". While they felt they could teach the Choice version without researchers present next time, they acknowledged having done the Fox-Only condition before the Choice version might have helped in their preparation. In addition to combining the Unplugged and Use days, teachers expressed the need for more time for students to program on their own finding the final 20 minutes of the last class insufficient. Both teachers, as previously described, allowed students to continue making their simulations on the day after the initiative was done.

6 DISCUSSION

Finding no significant differences in student difficulty perceptions, having the majority of students (18/28) complete the base activity, and neither seeing nor hearing teachers report difficulty teaching the new condition, we find that the Student-Choice design was not more difficult for either students or teachers. Thus, we believe we still maintain all the affordances that our original curriculum strategy, UMC, has in easing students and teachers into programming. In addition, the choice at the end gives students a chance to select the option(s) they wish, promoting their autonomy, and choose as many options as they need to satiate their need for competency as defined in Self-Determination Theory [7]. While not explicitly designed for, the emergent sharing of materials by students seemed to promote student relatedness. We can imagine further curricula explicitly adding a section for student sharing to make sure that this relatedness is met. In analyzing the diversity of student artifacts and the pathways students took to make their models along with how engaged students were as noted by researchers and teachers, we believe that the Choice version better engaged students in creating individualized programs which aligns with previous research in the design of constructionist learning environments [12]. Following advice from teachers, we believe that further versions of this activity should include *more* time for student differentiation. Further, to more closely resemble the definitions of Create found in UMC, PRIMM, and other models, we believe students can be given opportunities to choose their own extensions after they complete some of the choices from the set provided by the instructor. It is important to note that unlike these full constructionist models, our design is situated in an infused context where it is important to match computing and domain knowledge and creating pre-set choices for students ensures they create things that are scientifically accurate. Furthermore, these infused contexts involve novice teachers and students, so we must not forgo easing them into a new learning environment. As such, we feel 'Use' and 'Modify' as necessary steps on our path towards open-ended, constructionist 'Create'.

In framing our teacher observations and responses to interview questions through Frykholm's theory, we did not find teachers experienced pedagogical or emotional discomfort through our activities. As teachers discussed and our prior studies have shown, creating the scaffolded progression of moving from 'using' code to 'creating' their own seemed to reduce teacher discomfort in teaching content. Teachers also reported no difficulty in adopting a facilitator model. This could help scale our research practice partnership efforts to help expand CT activities into additional teachers' classrooms.

7 CONCLUSION/LESSONS LEARNED

Despite its exploratory nature, this case study supports that a curricula designed for freedom of choice is useful for both teachers and students. As we did not measure CT competency, we make no argument that this design works better or best at *teaching* programming and CT concepts. The small sample size of teachers (2) and students (68) does not allow us to extrapolate that this method is best for all teachers, contexts, or classrooms. Additionally, teachers first leading the non-choice version prior to the choice version could have greatly influenced their curricula preference as well as their development of competency in teaching the material. The

activities' warm reception, however, leads us to believe that content developers could benefit from creating activities that end in perceived student freedom of choice. To aid with this, we provide the following list of recommendations for creating these activities:

Scaffold Students and Teachers - Providing the necessary programming blocks students need to complete a choice ala a Parson's Problem greatly reduces cognitive load, especially in activities where lots of choices means lots of blocks. Similarly, giving teachers a "Cheat Sheet" of answers for each choice, scaffolds their ability to act as a facilitator and debugger.

Differentiate Choices by Difficulty - This time, challenge came from adding *more* choices, but each choice was relatively the same difficulty. In the future, we wish to create choice systems that have varying difficulty to give targeted tasks for each student skill level.

Create Choices that Show Visible Change - Prioritize choices that produce immediate changes in the run of the simulation (e.g. a new animal appearing in the environment).

Create Choices that Promote Content Inquiry - Our decision to primarily focus on adding more animals came from the fact that each new animal made the Food Web more complex.

Make things Complex, not Complicated - As demonstrated in Figure 2, each animal had relatively the same set of code blocks. In this way, a lot of choices were available without adding many additional necessary blocks for students.

Draw from Student Desires - Some of the choice ideas (e.g. bear) came from responses from students in the 'Fox-Only' condition on how they wished to extend their model. We suspect if students are asked in the beginning what they'd like in their model and these choices are implementable at the end, students will engage more with the material, feeling like the creations are their own.

In the future, we will describe how to develop these student-choice, CT, activity types for professional development for K-12 teachers. We hope that in promoting successes from our implementation, we can encourage teachers to adopt, adapt, and create their own activities that make them feel comfortable facilitating and allowing students to engage in self-guided inquiry and learning.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grants 1742351 and 1640141. Anything expressed in this material does not necessarily reflect the views of the NSF.

REFERENCES

[1] Bernd Meyer Aidan Lane and Jonathan Mullins. 2012. *Simulation with Cellular A Project Based Introduction to Programming* (first ed.). Monash University, Melbourne, Australia. Online: https://github.com/MonashAlexandria/snapapps.

[2] Tim Bell, Jason Alexander, Isaac Freeman, and Mick Grimley. 2009. Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology* 13, 1 (2009), 20–29.

[3] Veronica Cateté, Nicholas Lytle, Yihuan Dong, Danielle Boulden, Bita Akram, Jennifer Houchins, Tiffany Barnes, Eric Wiebe, James Lester, Bradford Mott, et al. 2018. Infusing computational thinking into middle grade science classrooms: lessons learned. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*. ACM, ACM, 21.

[4] Veronica Cateté, Kathleen Wassell, and Tiffany Barnes. 2014. Use and development of entertainment technologies in after school STEM program. In *Proc. of the 45th ACM technical symposium on Computer science education*. ACM, ACM, 163–168.

[5] National Research Council et al. 2011. *Successful K-12 STEM education: Identifying effective approaches in science, technology, engineering, and mathematics*. National Academies Press, Washington, D.C.

[6] Jan Cuny. 2012. Transforming high school computing: a call to action. *ACM Inroads* 3, 2 (2012), 32–36.

[7] Edward L Deci, Robert J Vallerand, Luc G Pelletier, and Richard M Ryan. 1991. Motivation and education: The self-determination perspective. *Educational psychologist* 26, 3-4 (1991), 325–346.

[8] Jeffrey Frykholm. 2004. Teachers' tolerance for discomfort: Implications for curricular reform in mathematics. *Journal of Curriculum and Supervision* 19, 2 (2004), 125–149.

[9] Dan Garcia, Brian Harvey, and Tiffany Barnes. 2015. The beauty and joy of computing. *ACM Inroads* 6, 4 (2015), 71–79.

[10] Marianthi Grizioti and Chronis Kynigos. 2018. Game modding for computational thinking: an integrated design approach. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*. ACM, New York, NY, USA, 687–692.

[11] Shuchi Grover, Roy Pea, and Stephen Cooper. 2016. Factors influencing computer science learning in middle school. In *Proceedings of the 47th ACM technical symposium on computing science education*. ACM, ACM, 552–557.

[12] Yasmin B Kafai and Mitchel Resnick. 2012. *Constructionism in practice: Designing, thinking, and learning in a digital world*. Routledge.

[13] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *Acm Inroads* 2, 1 (2011), 32–37.

[14] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin* 41, 3 (2009), 161–165.

[15] Nicholas Lytle, Veronica Cateté, Danielle Boulden, Yihuan Dong, Jennifer Houchins, Alexandra Milliken, Amy Isvik, Dolly Bounajim, Eric Wiebe, and Tiffany Barnes. 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*. ACM, ACM, 395–401.

[16] Nicholas Lytle, Veronica Cateté, Yihuan Dong, Danielle Boulden, Bita Akram, Jennifer Houchins, Tiffany Barnes, and Eric Wiebe. 2019. CEO: A Triangulated Evaluation of a Modeling-Based CT-Infused CS Activity for Non-CS Middle Grade Students. In *Proceedings of the ACM Conference on Global Computing Education*. ACM, ACM, 58–64.

[17] Jane Margolis. 2010. *Stuck in the shallow end: Education, race, and computing*. MIT Press, Cambridge, MA.

[18] Frank Pajares. 1996. Self-efficacy beliefs in academic settings. *Review of educational research* 66, 4 (1996), 543–578.

[19] Seymour Papert and Idit Harel. 1991. Situating constructionism. *Constructionism* 36, 2 (1991), 1–11.

[20] Dale Parsons and Patricia Haden. 2006. Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52 (ACE '06)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 157–163. http://dl.acm.org/citation.cfm?id=1151869.1151890

[21] Thomas W Price, Jennifer Albert, Veronica Catete, and Tiffany Barnes. 2015. BJC in action: Comparison of student perceptions of a computer science principles course. In *Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), 2015*. IEEE, IEEE, 1–4.

[22] Thomas W Price, Veronica Cateté, Jennifer Albert, Tiffany Barnes, and Daniel D Garcia. 2016. Lessons Learned from BJC CS Principles Professional Development. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, ACM, New York, NY, 467–472.

[23] Robert Pucher and Martin Lehner. 2011. Project based learning in computer science–a review of more than 500 projects. *Procedia-Social and Behavioral Sciences* 29 (2011), 1561–1566.

[24] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay S Silver, Brian Silverman, et al. 2009. Scratch: Programming for all. *Commun. Acm* 52, 11 (2009), 60–67.

[25] Sue Sentance, Jane Waite, and Maria Kallia. 2019. Teachers' Experiences of Using PRIMM to Teach Programming in School. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 476–482. https://doi.org/10.1145/3287324.3287477

[26] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.

[27] David Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. 2014. Defining computational thinking for science, technology, engineering, and math.

[28] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.