Jl. of Computers in Mathematics and Science Teaching (2023) **41**(3), 1–29

Promote Computational Thinking of Middle-School Students Through SPARC-Integrated Science Instruction

JIANLAN WANG
Texas Tech University, USA
jianlan.wang@ttu.edu

YUANLIN ZHANG
Texas Tech University, USA
y.zhang@ttu.edu

ARTHUR JONES
Texas Tech University, USA
arthur.l.jones@ttu.edu

RORY ECKEL
Texas Tech University, USA
rory.eckel@ttu.edu

JOSHUA HAWKINS
Texas Tech University, USA
joshua.hawkins@ttu.edu

DARREL MUSSLEWHITE
Lubbock-Cooper Independent School District, USA
dmusslewhite@lcisd.net

Despite the importance of computer science education and computational thinking, there have been limited examples of computer science education at K-12 classrooms that authentically represents the work of computer scientists, especially programming. One reason is the lack of a measurable definition of computational thinking and a programming language

friendly to K-12 students. In this case study, we used a text-based declarative programming language named SPARC to integrate computer science into the topic of the atomic structure. We also designed a rubric for the assessment of computational thinking that included two categories of abstraction and programming. We examined the experiences of two 8th-grade students representing those who have some experience with computer science and programming and those without any experience. The findings support the feasibility of integrating computer science and science and exposing middle-school students to text-based programming. We also discussed the pedagogical concerns in light of the students' challenges.

Keywords: computational thinking, abstraction, programming, middle school, computer-integrated science instruction

INTRODUCTION

It is widely accepted the importance of integrating computer science in K-12 education as an answer to the growing demand of highly competent workforce in the 21st century (Aho, 2012; Barr, Harrison, & Conery, 2011). However, such an integration rarely happens successfully (National Research Council, 2010). Educators are facing several challenges. First, there are few clear-cut learning outcomes entailed by computer science education. Computational Thinking (CT) is believed to be a competence most appropriately developed by computer science education (Weintrop, Beheshti, Horn, Orton, Jona, Trouille, & Wilensky, 2016; Wing, 2006; Wing 2011). However, it is loosely defined. Different studies define CT in unique ways that meet their research objective and programming environments (Basawapatna, Koh, & Repenning, 2010; Marcu et al., 2010; Sengupta et al., 2013; Werner, Denner, Campe, & Kawamoto., 2012). There seems to be an agreement among various definitions that CT involves components like modeling, abstraction, and algorithm. However, these concepts lack measurable definitions or criteria for computer science education at the K-12 levels. The next generation science standards (NGSS) explicitly require CT (National Research Council, 2012) but provide limited practical guidance for teachers. Thus, K-12 educators are likely to either mistake computer science for a mysteriously complex field beyond students' competency or oversimply it as a computer game for students' entertainment.

Secondly, it is unclear whether computer science should be incorporated as a separate discipline or integrated into other K-12 curricula (Grover & Pea, 2013). It is believed that computer science education can and should start as early as middle school where students start to build their affinity for computer science (Grover, Pea, & Cooper, 2014; Settle, Franke, Hansen, Spaltro, Jurisson, Rennert-May, & Wildeman., 2012). However, there are few mandatory computer science courses at middle schools. Efforts of integrating computer science mainly take place at elective courses or after school programs (Kelleher, Pausch, & Kiesler, 2007; Grover, Pea, & Cooper, 2015; Werner et al., 2012) as if it is for a special group of students, such as those who are interested in or good at science or mathematics. Scholars claim that the essence of computer science, such as CT, overlaps with the mentality of other core subjects like mathematical thinking or scientific reasoning (Sengupta et al., 2013; Weintrop et al., 2016). This underlying connection suggests reciprocal benefits from the integration of computer science and core subjects at middle schools. However, there is limited empirical evidence supporting that hypothesis. Unsurprisingly, computer science is not a core subject at middle-school education.

Thirdly, there are limited declarative programming languages appropriate for middle-school students. Programming is a critical component in computer science (Wing, 2011). However, few programming languages are friendly to middle-school students that resemble the work of computer scientists. Sophisticated languages like Java and Python are too complex for programming novices like middle-school students (Grover & Pea, 2013). Block-based graphic programming environments tailored for middle-school students to design games or create animations normally use approaches of "drag-and-drop", "assembling graphic tiles", or "selection from a drop-down bar" (Maloney, Burd, Kafai, Rusk, Silverman, & Resnick, 2004; Werner et al., 2012). Those approaches enhance students' interest toward computer science but may mislead their perception of computer scientists' work as "clicking on the mouse". In addition, bypassing the challenge of programming may not help students develop core competences in computer science, such as CT. It seems to be a challenge for computer science educators to find the balanced point between the engagement and authenticity in students' experience with computer science.

In this study, we tried to integrate computer science with middle-school science through the language of SPARC. We designed a curriculum about representing the atomic model with SPARC and piloted it with two 8th-grade students through online instruction during the pandemic. Our research questions are:

Q1. How well did the students master the scientific content knowledge involved in this SPARC unit?
Q2. What were the two students' experiences with the computational-thinking tasks in this SPARC unit?
Q3. How did the two students perceive this SPARC unit?

THEORETICAL FRAMEWORK

SPARC as an answer set programming language

SPARC is an Answer Set Programming (ASP) language designed to represent knowledge through a declarative approach (Balai, Gelfond, & Zhang, 2013). Being declarative means that SPARC describes knowledge in terms of objects and relations among them rather than a procedural sequence of actions. Correspondingly, an answer set is a set of objects that satisfy certain relations. A typical example of ASP is the map coloring problem (Kowalski, 2014) that asks how to color each state of the united states, given three colors, so that no two adjacent states have the same color. In this problem, there are two objects of state and color. State has the values of the name of the 50 states and color has the values of red, green, and blue. The relation between the two objects in English is "States X and Y are adjacent, then they have different colors" or "State X has the color of R, State Y is adjacent to State X, then State Y has the color of non-R". After translating those statements into the language of SPARC, the system will return with the visualization of the map of the united states with no two adjacent states having the same color. The strength of SPARC is the use of simple logic programming to solve complex problems.

ASP normally has three aspects (Kowalski, 2014): 1) objects with domains; 2) relations between objects; 3) knowledge or rules that specify relations. The three aspects in SPARC correspond to the sections of SORTS, RULES, and PREDICATES respectively. Figure 1 is a SPARC example of predation. The code in *sorts* describes that the objects in this example are organisms that have a domain of two values, i.e., snake and eagle. The code in *predicates* describes a general relation of "preysOn" between organisms, which in English is "preysOn (X, Y) denotes that organism X preys on organism Y". The code in *rules* describes the knowledge of specific relations, which in English is that "eagles prey on snakes". With those codes, SPARC can answer questions like "do eagles prey on snake?", i.e. *"preysOn(eagle, snake)"*, and return with the answer of "yes". Other questions that can be answered are like *"preyOn(X, snake)"*. Here *X* or words with the first let-

ter capitalized (such as *"What"* or *"Hello"*) are interpreted by SPARC as a variable. Therefore, organism names in rules are written in small letters as "eagle" and "snake". It is easy to see how SPARC can be analogous to modeling in math or science. Take a linear algebraic relation of y=3x for example, "Sorts" defines the range and domain of the two variables x and y, "Predicates" defines a general linear relation of y=ax, and "Rules" defines a specific format of the linear relation in terms of y=3x. In the next section, we will discuss how we define CT using SPARC.

Figure 1. Example of a SPARC model about predation.

Computational thinking

CT was introduced by Wing (2006) as "computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (p. 33). In her later work (Wing, 2011), she clarified that "computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (p.1). Other definitions include CT being a thought process of formulating solutions to a problem that can be represented by computational steps (Aho, 2012), an awareness of the computational aspect of the world and the ability to use computer science techniques to explore the computational nature of the world (Royal Society, 2012), and an algorithmic approach with a high-level of abstraction for problem solving (García-Peñalvo, & Cruz-Benito, 2016). Those definitions give a general sense that CT as the way people solve problems like computer scientists, but do not specify details like how problems are solved. The terms like system, model, process, steps, and algorithm appear repetitively in different definitions and sometimes are used interchangeably, which configures a vague contour of CT that yields limited practical guidance to middle-school educators.

Further efforts have been taken to decompose CT. Weintrop et al. (2016) proposed a taxonomy that broke down CT into four categories of skills: data and information skills, modeling and simulation skills, computational skills, and system thinking skills. Those categories overlap with each other. For instance, modeling and systematic thinking overlap because a model is a system. Grover and Pea (2013) listed elements (Table 1) that "are now widely accepted as comprising CT and form the basis of curricula that aim to support its learning as well as assess its development" (p. 39). Like the CT taxonomy, abstraction is believed to have multiple levels which are defined variously in different areas. In information science, the different levels of abstraction are the instances of being informed by an existing system, the creation of new instances of informing, and the creation of new designs for informing (Cohen, 2009). Regarding algorithm, abstraction has different levels based on whether the algorithm is free to programming language or not (Perrenet, Groote, & Kaasenbrood, 2005). The different levels of abstraction in software architectures are determined by the architectural structure and functionality (Medvidovic, Taylor, & Whitehead, 1996). The ambiguity from various definitions makes it an unwieldy task for K-12 teachers to determine the learning objectives while implementing CT practices.

After synthesizing various definitions, we took into consideration two factors while defining CT: 1) aligning CT with modeling in science education (Hestenes, 1987); 2) narrowing down CT categories in existing frameworks to a couple of practices achievable to middle-school students. As illustrated in the previous section, the structure of SPARC is analogous to modeling as they both aim to construct a generalizable relation between variables. Two skills are critical in this process: abstraction and programming. We define abstraction as a process of constructing a system of objects and relations (i.e., a model) that represents generalizable rules of knowledge as the solution to a certain problem. Different levels of abstraction require various amounts of cognitive load. From a lower level to an upper one, it is decreasing detailed information and increasing possibility of generalization. Specifically, abstraction has three levels of symbolizing perceivable *objects*, building *relations* connecting symbolic objects, building *patterns* involving symbolic objects and relations that have the power of predicting (Table 1). Teacher scaffolding is probably needed to protect students from the frustration of not being able accomplish a CT task. Scaffolding refers to partial information imparted to students about significant steps of a task but is insufficient to accomplish the task. We separated the two situations by using (i) for with scaffolding and (ii) for without. We used the sign of "-" to indicate that students failed a task and it was completely accomplished by the teacher.

Programming is another important aspect of CT (Grover & Pea, 2013; Wing, 2011). It represents programmers' both grammatic ability that guarantees their communication with a computer language and algorithmic reasoning underlying problem solving. Linn (1985) described a chain of cognitive accomplishments related to programming. The first link was comprehension, which was understanding the syntax of a program and the ability to make minor changes. The second link was design, which was building a program using procedural skills to solve a problem. The third link is problem-solving, which is transferring problem-solving skills to a new system. Adopting Linn's structure in middle school, Werner et al. (2012) designed 3 programming tasks with hierarchical difficulty levels, which were to adjust parameters in pre-established syntax known as "event handler" (Task 1), to adjust existing event handlers (Task 3), and to create new event handlers (Task 2). In this study, we defined three levels of programming skills, which were the *awareness of the programming environment* (i.e., SPARC in this study), understanding the *syntax of programming language*, and being capable to manipulate the *dynamic process of programming* (Table 1). Debugging was embedded in the third aspect as it guaranteed the successful operation of SPARC. Like abstraction, programming tasks accomplished with/without scaffolding were coded as (i) and (ii) respectively. The sign of "-" indicated that a teacher programmed for the students.

Table 1

Theoretical structure and assessment criteria of computational thinking.

Component	Levels	Elements of CT (Grover & Pea, 2013)
Abstraction (Assessed from written or verbal articulation)	A1. **Object:** Be able to symbolize objects to appropriate representations. Accomplished by teacher: A1- Accomplished by student with scaffolding: A1(i) Accomplished by student without scaffolding: A1(ii)	• Abstractions and pattern generalization • Systematic processing information • Symbol systems and representations
	A2. **Relation:** Be able to correctly describe specific relations between objects. Accomplished by teacher: A2- Accomplished by student with scaffolding: A2(i) Accomplished by student without scaffolding: A2(ii)	
	A3. **Pattern:** Be able to correctly describe a general pattern of relations of interest. Accomplished by teacher: A3- Accomplished by student with scaffolding: A3(i) Accomplished by student without scaffolding: A3(ii)	

Component	Levels	Elements of CT (Grover & Pea, 2013)
Programming (Assessed from computer)	P1. **Awareness of the programming environment**: Be able to input information at the appropriate places of SORTS, RULES, PREDICATES, and QUERIES. Accomplished by teacher: P1- Accomplished by student with scaffolding: P1(i) Accomplished by student without scaffolding: P1(ii)	• Algorithmic notions of flow of control • Conditional logic • Debugging and systematic error detection
	P2. **Syntax of programming language**: Be able to use correct syntax for objects, variables, relations, and questions. Accomplished by teacher: P2- Accomplished by student with scaffolding: P2(i) Accomplished by student without scaffolding: P2(ii)	
	P3. **Dynamic process of programming**: Be able to debug the program and successfully run the program. Accomplished by teacher: P3- Accomplished by student with scaffolding: P3(i) Accomplished by student without scaffolding: P3(ii)	
Not included in this study	N/A	• Efficiency and performance constraints • Structured problem decomposition • Iterative, recursive, and parallel thinking

Literature review

Research regarding computer science education has been mainly carried out at the college level but not much in K-12 (Blikstein & Wilensky 2009; Hambrusch et al. 2009). Kelleher et al. (2007) examined the programming environment of storytelling Alice in comparison to general Alice. Both Alice environments involved the drag-and-drop format of programming, which in the authors' words were "removes the possibility for making syntax errors" (p.1455). The difference was that generic Alice contextualized programming in pre-established stories whereas storytelling Alice provided high-level animations and a gallery of characters and scene elements that allowed students to create their own stories. The pedagogical support was story-based tutorials where students were walked through the process of creating a story, so they were familiar with the programming environment and procedure. The participants were 88 girls separated in two groups receiving generic and storytelling Alice respectively. It was found that both groups were equally successful in understanding basic programming syntax associated with Alice, such as events, loops, and parameters. The storytell-

ing Alice group demonstrated more engagement in terms of the time spent on programming and expressed more interest in continuous use of Alice.

Basawapatna, Koh, & Repenning (2010) explored the use of scalable game design at the middle school level. Scalable game design was flexible to students of different levels in exposure to computer science, programming skills, etc. The authors used the programming environment of AgentSheets that allowed students to create games easily without worrying about details of implementation. According to the authors, AgentSheets was also powerful because it was compatible to advanced technology like artificial intelligence and Java. CT was a learning outcome not clearly defined but reified as specific patterns in different cases, like push and pull, absorption, and diffusion. Thus, it was unclear what competences pertaining to scalable game design were transferable or generalizable. From both observations and a post-reflective survey, the authors found that scalable game designs were entertaining to middle-school students. Other than interest, there was no assessment of students' cognitive competences. The authors also mentioned that the pedagogies of peer collaboration and "cheat sheets" as supplementary learning materials.

Werner et al. (2012) designed Fairy assessment aimed to evaluate middle-school students' CT. They used an environment, known as Alice, for a computer game where users can control characters to accomplish tasks using drag-and-drop programming. They prepared three tasks that required different levels of CT, which in this study referred to algorithmic thinking, abstraction, and modeling. Task 1 assessed whether students understood the basic framework of the environment, could place instructions at the correct place, or recognized the need to change parameters. Task 2 assessed whether students understood the program's default execution sequence and adjust instructions accordingly using conditional logic. Task 3 assessed whether students could create new instructions by adjust the program's default execution sequence. They applied those tasks with 311 students in elective after-school classes. With the data, the authors claimed that the assessment was a promising strategy because it was engaging and could successfully gauge a range of CT. However, the authors did not define the concepts of algorithm, abstraction, and modeling. For example, Task 3 was different from Task 2 in the absence of "problem-solving", yet Task 3 itself was a "problem" that students needed to solve. Due to this ambiguity, it is hard to transfer CT elements assessed in this study to other contexts with different programming environments. In addition, the graphic drag-and-drop way of programming was student-friendly but not authentic, which probably could not prompt students to think like computer scientists.

Grover, Pea, & Cooper (2015) designed a computer science curriculum for middle school called Foundations for Advancing Computational Thinking (FACT). In this study, CT as one learning outcome referred to the algorithmic flow such as sequence, loop, and conditional logic. Pedagogy wise, FACT was claimed to use the inquiry-based approach of 5E (i.e. engage, explore, explain, elaborate and evaluate) and cognitive apprenticeship where students worked on both teacher-directed assignments and open-ended projects to model solutions to computational problems. This study took place in a middle-school classroom with 26 students receiving face-to-face instruction and another 28 students receiving online FACT in a self-paced and collaboratively manner. The authors used a variety of assessment including several multiple-choice tests as formative assessment, pre-post tests on knowledge of algorithmic flow, a post test about transferring algorithm to text-based programming like Java and Pascal, a post project, pre-post surveys and post interviews on the perception of computer science. It was found that FACT in both settings were equally effective in promoting students' algorithmic ability. Students were able to transfer the knowledge of algorithmic flow to text-based programming but had problems with loops and variables. The experience with FACT promoted students' understanding of the work of computer scientists. The authors did not specify how algorithmic flow like sequence and loop was important to students in other disciplines. In addition, the transfer of knowledge indicated little about students' performance in text-based programming.

There are even fewer studies about integrating computer science with core middle-school subjects. Marcu et al. (2010) tried to integrate computer science into an engineering project in a 4-week summer camp for 53 middle-school students. PicoCricket Kits were used in this project with which the students were expected to create objects like a kinetic sculpture, an amusement park, and a fairy tale house. The students were assigned the roles of electric, software, design, and civil engineers who oversaw different aspects of a project. Programming was semi-structured that students needed to drag and drop codes in pre-established syntax. Besides role-playing group work, pedagogical support included hands-on exploration, instantaneous support, and post project demonstration. As indicated by pre-post surveys, the participants perceived the experience in this summer camp as being fun and they felt that they learned something without specifying what the thing was. Besides, the participants developed their understanding of the work of engineers and computer scientists and shifted their attitudes towards the two areas from negative (e.g. being boring and hard) to positive (i.e. being fun and easy).

Sengupta et al. (2013) introduced the framework of integrating computer science into middle science education using an agent-based visual programming environment called CTSiM. In CTSiM, students used a drag-and-drop interface to create their program where they arranged primitives or manipulated parameters. This environment could visualize the enactment of programming with the actions of an agent in science scenarios of Newtonian mechanics and an ecosystem in a fish tank. The students refined their model based on the instantaneous feedback from the system about how the agents operated or the comparison with the model created by experts. According to the authors, CTSiM could promote students' understanding of science content knowledge and programming constructs like loops and variables. They measured 24 middle-school students' science content knowledge and observed pre-post differences with the students (N=15) receiving one-on-one scaffolding from researchers in the school library. There was no measurement of CT.

The studies reviewed above show that the efforts of integrating computer science into middle schools have been focused on the affective learning outcome in terms of students' engagement or motivation toward computer science. CT as the cognitive learning outcome has been seldom systematically measured or the measurement of CT cannot be generalized to different programming languages. Besides, most of the studies used pre-post measurement to reflect on the efficacy of a computer science project but left it a black box how students interact with the project. Without that knowledge, it is questionable to ascribe the pre-post difference, if any, to students' experience with computer science. Grover and Pea (2013) pointed out that future research regarding computer science at K-12 should be about the definition of CT at K-12 and the integration of computer science and other subjects. In this study, we scrutinized two middle-school students' experience with a SPARC-integrated science unit and their performance in CT tasks,

METHODS

Context and participants

We applied the method of case study (Ary et al., 2010) to examine in-depth the experience of two eighth-grade students from a public middle school in an urban district. Tyler (pseudonym) was a male Hispanic student and Amber (pseudonym) was a female Hispanic student. Before participating in this project, neither of them had any experience with formal learn-

ing of computer science. SPARC and computer models were brand new to them. Tyler claimed that he had used Java and Python before, but probably at a superficial level because he originally referred to Python as "Raptor". Compared to Tyler, Amber had no experience with programming at all. This project took place in one of their elective courses taught by Mr. C who had participated in a workshop about SPARC. This course was given online due to the COVID pandemic. The SPARC unit contained 8 lessons that lasted for 4 weeks (Table 2). In each lesson, Mr. C video recorded his instruction ahead of time that included the introduction of computer models built with SPARC, the screenshot of his demonstration of programming in SPARC, and the explanation of student tasks. Two undergraduate Teaching Assistants (TA) major in computer science worked one-on-one with Tyler and Amber respectively. They were both male. Their tasks were three-fold: 1) controlling the pace of the instructional video so the students would work on CT tasks on their own before accessing the answer from Mr. C; 2) answering the students' questions about Mr. C's instruction or CT tasks; 3) asking questions so the students could talk out loud their reasoning while they were working on the CT tasks.

Table 2
Eight lessons in the SPARC unit

	Content
1	Introduction to computer science and computer modelling in the context of family members. **Abstraction.** Students accessed the *Objects* of family members (e.g. John, Peter, Sarah, and Linda), *Relations* (e.g., John is the father of Sarah), and *Rules* (e.g., father(X,Y) denotes that X is the father of Y) **Programming.** Students signed up for online SPARC, copied and pasted the syntax provided, e.g. "#people = {john, peter, sara, linda}.", "father(#people, #people).", "father(john, peter)"
2	Continuation with the model of family members by including the rule of "mother". **Abstraction.** Students expanded the model in Lesson 1 by including more objects and the rule of "mother" (e.g., mother(X,Y) denotes that X is the mother of Y) and "parent" **Programming.** Students typed new objects and rules (e.g., "parent(joann, linda)") into existing syntax and asked questions in *Query* about different rules, like "parent(joann, linda)" that returned "yes".
3	Introduction to variables **Abstraction.** Students accessed words with the first letter capitalized as a variable that meant what, like X, Who, and TheMotherofLinda. The note father(Who, peter) means who is the father of peter, **Programming.** Students asked questions in *Query* that contained variables. The uppercase first letter features as a placeholder. They asked "father(Who, peter)" that returned "Who = john".

	Content
4	Introduction to the if symbol ":-" **Abstraction.** Students created "identical" relations for two different names such as "father" and "dad". The syntax, dad(X, Y) :- father(X, Y), means that X is the dad of Y if X is the father of Y. **Programming.** Students used the if symbol ":-" in *Rules* to simplify the syntax. They added the rule of "dad(X, Y) :- father(X, Y)." and asked questions in *Query* like "dad(john, peter)" that returned "yes".
5	Shifting from the context of family members to that of the atomic model and the periodic table. **Abstraction.** Students accessed the *Objects* of the first 20 elements in the periodic table. The *Relations* are the unique symbol for each element. The *Rule* of symbolFor (E, S) denotes that the chemical symbol of the element "E" is "S". **Programming.** Students inputted the symbols of the first 20 elements, such as "symbolFor(hydrogen, h)." Then they asked queries about the symbols of specific elements. For example, the query "symbolFor(hydrogen, H)" returned "H=h".
6	Expand the model from chemical symbol to atomic number **Abstraction.** Students created *the Relations* of the atomic numbers of the same 20 elements by referring to the periodic table. The *Rule* of atomicNumber(E, N) denotes that the atomic number of the element E is N. **Programming.** Students inputted the atomic numbers of the first 20 elements, such as "atomicNumber(carbon, 6)." Then they asked queries about the atomic numbers of specific elements. For example, the query "atomicNumber(carbon, 6)" returned "yes".
7	Expand the model by including proton number **Abstraction.** Students created *the Relations* of the proton numbers of the same 20 elements by referring to the periodic table. The *Rule* of protonsOf(E, N) denotes that the proton number of the element E is N. The atomic number of an element is the same as its proton number. Thus, another rule is that protonsOf(E,N) = atomicNumber(E,N). **Programming.** Students inputted the proton numbers of the first 20 elements, such as "protonsOf(carbon, 6).". Alternatively, they used the "if" syntax of "protonsOf(E, N) :- atomicNumber(E, N)." to avoid reinputting the proton numbers. Then they asked queries about the proton numbers of specific elements. For example, the query "atomicNumber(carbon, X)" returned "X = 6".
8	Expand the model by including neutron number and mass number **Abstraction.** Students created the *Relations* of the mass number and neutron number of the first 20 elements by referring to the periodic table. One set of *Rules* includes neutronsOf(E,N) and massNumber(E,N). Another *Rule* is massNumber(E, N) = protonsOf(E, N) + neutronsOf(E, N). Programming. Students inputted the mas numbers of the first 20 elements, such as "neutronsOf(carbon, 6)." They also learned the "and" symbol "," and typed in "massNumber(E, M) :- protonsOf(E, P), neutronsOf(E, N), M = P + N." Then the asked queries about the mass and neutron numbers of specific elements. For example, the query "neutronsOf(carbon, N)" returned "N = 6".

As shown in Table 2, we contextualized SPARC in the topics of family members and the atomic model. The former was to engage students in SPARC-based computer science with a "low-floor" topic with which they

were familiar. They were expected to understand what building a computer model means, which in student-friendly language was to teach "dumb" computers something that we know using their language. We selected the latter because the atomic model was a key concept required by the 8th-grade science standard in the state of Texas.

8.5 (A) describe the structure of atoms, including the masses, electrical charges, and locations, of protons and neutrons in the nucleus and electrons in the electron cloud (Texas Essential Knowledge and Skills, Grade 8)

The two students had learned the concepts about the atomic model from their science courses. Contextualizing SPARC in this topic could reveal the role of students' content knowledge in their learning of SPARC and their development of CT. During *Abstraction*, the focus was on knowledge representation. The students could capitalize the first letter of names and elements based the regulation (e.g., "John", "Carbon"). Meanwhile, they did not need to worry about signs or punctuation, such as the period at the end of a rule (e.g., "father(John, Peter)", "protonsOf (Carbon, 6)"). The focus during *Programming* was on the syntax so SPARC could function properly. The students needed to follow the rules of SPARC rigorously, such as lower cases for objects (e.g., "john", "carbon") and a period ending a rule (e.g., "father(john, peter).", "protonsOf (carbon, 6).").

Data collection and analysis

To answer the first research question, we conducted one-on-one pre-interviews with the two students to assess their scientific content knowledge. Five questions (Appendix) about the atomic model were adapted from the released questions of Grade 8 State of Texas Assessments of Academic Readiness (STARR). A periodic table was provided during the pre-interview. These questions assessed the students' ability of reading the periodic table and their knowledge of the atomic structure, such as the relationship between the numbers of protons, neutrons, and electrons. The students talked out loud their thinking while answering these questions. The interviewer provided necessary support when the students could not recall key knowledge. The entire interviews were video-recorded and later transcribed for the analysis of the students' scientific content knowledge. In addition, we video recorded all the eight lessons when each student worked with an undergraduate TA. The students' content knowledge demonstrated in the vid-

eos was coded as D (Correct domain knowledge) or D- (Incorrect domain knowledge). D and D- were coded by one expert in science education in the research team.

To answer the second question, we first divided the videos into CT tasks and coded how each student worked on these tasks based on Table 1. We did not code the parts when the students watched the instructional videos prepared by Mr. C. However, we paid attention to whether the student copied the answer directly from the videos to accomplish a task. If they did it, we coded that task with "-". Take a task of *Abstraction* for example, one student was asked to articulate the chemical symbol of hydrogen. If the student answered that "You need to use the rule of symbolFor to describe that the symbol for the element of hydrogen is H.", this task would be coded as "A2(ii)" because this student could build a specific *Relation* without the support from the TA or the video. In another task of *Programming* about the same topic, if a student typed in "symbolFor(hydrogen, H)" on his/her SPARC platform and the system returned an error message. With the prompt from the TA, this student added a period at the end of this sentence, but the syntax was still erroneous. This student could not fix the problem and then watched the video to realize that "H" should be "h". This task would be coded as "P3-" because this student was able to use the correct format of syntax at the correct place but failed with debugging until referring to the video. Two of us coded each video separatively, tracked disagreement, and met to reach an agreement. Cohen's Kappa (Cohen, 1960) was used to examine the inter-rater reliability. Finally, we calculated the percentages of different codes of tasks. Not all the CT tasks were designed for the highest level of *Abstraction* or *Programming*. For example, the task of "how to represent the atomic number of Hydrogen" did not involve general patterns (A3). Thus, we did not measure the students' CT with the frequencies of different levels of tasks, but the percentages of tasks not accomplished by students ("-"), tasks accomplished by students with TA scaffolding ("i"), and tasks accomplished by students without scaffolding ("ii").

To answer the third question, we conducted one-on-one post interviews with the two students. In this interview, we asked the two students about three themes, including their opinions about integrating SPARC at middle schools, their feedback to the SPARC unit, and their feedback to the instructional support from both the TA and Mr. C's video. Some exemplary questions were "Which aspect or concept of this SPARC unit did you find most interesting? Most challenging?", "We applied SPARC in the contexts of both family members and the atomic model. Which context did you enjoy more?", and "Which one between Mr. C's instructional videos and the TA's

support did you find more helpful?". We audio-recorded the post interviews, transcribed students' answers, and then compared between the two students to draw the conclusions.

FINDINGS

In this section, we will use italic words for the direct quote of the conversations in the interviews and class videos with the TAs and the students' work of programming on the SPARC platform. Our note will be in the brackets of "[ ]".

Q1. How well did the students master the scientific content knowledge involved in this SPARC unit?

During the pre-interview, Tyler encountered difficulties in the first question (Appendix). The interviewer helped Tyler locate the four elements of Indium, Scandium, Aluminum, and Zinc in the periodic table. Tyler first ruled out Option A because "*I have never heard of neutrons being outside of nucleus"*. He hesitated about the meanings of the numbers in a cell of an element and mistook the atomic mass (i.e., 45 for Scandium) for the proton number. Then he read the numbers "2-8-9-2" at the bottom and stated that "*Every number is how many electrons are on the shell. The shell is every ring around the nucleus*". Tyler added all the numbers together and concluded that the electron number is 21 rather than 45 for Scandium. He used this method to rule out Options B and C. Tyler knew that Option D was the right answer but did not know why. Then the interviewer had a conversation with him as shown below

"Tyler: *It has been a while since I have done this, I try to remember, is it how you figure out the neutrons, use protons minus electrons?* – [Fuzzy memory]
Interviewer: *Let's work together to refresh your memory. What does the inside of an atom look like?*
Tyler: *In the center you have the nucleus. That's where the protons and neutrons are. Outside the nucleus, the shells or rings, on the rings is the electrons.* – [Remember the atomic structure]
Interviewer: *Is there a relationship between the numbers of protons, neutrons, and electrons?*

Tyler: *Yes, we have a bunch of equations to figure that out, but I cannot remember a single one. I just remember the base of it. Are the protons and neutrons the same?* – [Cannot remember the relations between protons, electrons, and neutrons]
Interviewer: *Probably, do you have a reason for that?*
Tyler: *We have little letters for the equations, p for proton, n for nucleus, and e for electrons, something is equal to whatever the other one is, but I cannot remember them.*
Interviewer: *The number of protons is equal to the number of electrons. Do you know why?*
Tyler: *It's kind of opposite, it has to go into the neutrons?* – [Probably did not learn why]
Interviewer: *What kind of charges do protons have?*
Tyler: *Protons are positive and electrons are negative.* – [Remember the charge types]
Interviewer: *Then what if the numbers of protons and electrons are not equal?*
Tyler: *The atom would be one way or the other, if there are more protons, it would be a proton atom or positive atom.*
Interviewer: *Exactly, but atoms are neutral, what does that tell you?*
Tyler: *You have the numbers of protons and electrons, add them up, that's how many neutrons?*
Interviewer: Do electrons have mass?
Tyler: *I think so, I never came across that question before.* – [Probably did not learn why]
Interviewer: *The mass of electrons can be ignored compared to that of protons and neutrons. The number beneath the chemical symbol is the mass number for an element, which is composed of the number of protons and the number of neutrons. How do we figure out the neutron number once we know the mass number and the proton or electron number?*
Tyler: *Mass number minus proton number. Now I remember."*

As shown in the conversation, Tyler probably learned the atomic model from his science course by memorizing facts such as protons and neutrons are inside the nucleus and electrons outside are on different shells. However, his memory was fragmented and faded already. Tyler could not recall the equations and did not take the prompts from the interviewer, which suggests that he probably did not understand science meanings underneath those equations. After the interviewer explained the equations about the atomic model, Tyler was able to provide correct answers and reasoning for the rest

questions. The interviewer later asked Tyler what could be done to reinforce his memory of the equations. Tyler answered that "*For the equations, it is hard to explain. I don't have a certain way of remembering things. Really to me, it's just that I think about it, sometimes it pops out, sometimes I cannot remember, like the equations. I know we have talked about this*." Thus, it could be inferred that Tyler probably learned this topic by rote and did not organize well the information received from the science teacher.

In the interview with Amber, she could not solve the questions either at the beginning. The interviewer gauged her understanding of the atomic model with the conversation below:

"Interviewer: *Could you talk about your impression of the model of an atom?*
Amber: *An atom is like a molecule. A molecule is made of protons, neutrons. It is like, atoms are around us, air and stuff like that*. – [Misuse of molecule]
Interviewer: *You mentions protons and neutrons. How are they arranged or located in an atom?*
Amber: *They are located in the middle of an atom.* – [Remember the atomic structure]
Interviewer: *Protons and neutrons in the center. Anything else that can be found in an atom?*
Amber: *Electron cloud, like rings round, that's an electron cloud, an electron cloud is electrons.*
Interviewer: *Sounds great. Since you mentioned rings, electrons should orbit the center, why do you think it is called a cloud?*
Amber: *Because it is made of all the electrons. It's a certain amount on each ring.* – [Remember the term of electron cloud but does not understand it]
Interviewer: *That makes sense. How about protons?*
Amber: *Atomic number, the atomic number matches proton.* – [Remember one relation]
Interviewer: *Excellent. How about neutrons?*
Amber: *I got neutrons and protons mixed up. I know how to find the protons, I don't know how to find the neutrons.* – [Cannot remember other relations]
Interviewer: *Do you think there is a relation between proton number and neutron number?*
Amber: *There is a relationship, but I don't know what it is. I know the proton is the atomic number, and neutrons are like subtraction, I don't know.* – [Fuzzy memory]

Interviewer: *It's OK. Do you know why proton is the atomic number? Or you have been told that this is a fact?*
Amber: *Well, my science teacher told me one time, and I forgot.* – [Probably did not learn why]
Interviewer: *It's alright. Neutrons plus protons is equal to the atomic mass. Which number in the cell of zinc is the mass number?*
Amber: *It is 45. D is correct, the proton is 30.*
Interviewer: *Why is the number of electrons also 30?*
Amber: *Good question, I don't know. I forgot how to find the electrons.* – [Fuzzy memory]
Interviewer: *Electrons are equal to protons."*

Similar to Tyler, Amber's knowledge of the atomic model was fragmented. She could recall some concepts like electron cloud, protons, and atomic number, but could not explain them well. She probably learned this topic by rote and her memory of the facts about the atomic structure faded as well. After receiving the instruction from the interviewer, Amber could solve the rest questions with occasional consultation with the interviewer, such as a question of "*Do we have to use neutrons, protons to find electrons?*". While being asked what could be done to reinforce her memory, she was unsure but stated that "*I don't know. It depends. Some topics, I can remember easily, some that make sense to me, but not the other.*" The cases of Tyler and Amber corresponded to a common challenge in science education about how to reinforce students' long-term memory of content knowledge. We did not have any codes of "D-" during the entire SPARC unit. Neither Tyler nor Amber demonstrated any errors or difficulties with the topic of the atomic model.

Q2. What were the two students' experiences with the computational-thinking tasks in this SPARC unit?

There were 54 CT tasks for Tyler and 54 ones for Amber. The Cohen's Kappa was 0.81 ($p<0.001$) between the two raters on the 108 tasks, which justified the inter-rater reliability of the rubric (Table 1). We summarized in Table 3 the percentages of different task codes from the two students. In Tyler's case, the percentages of the Abstraction tasks not accomplished by Tyler ("-"), accomplished with TA support ("i"), and accomplished without TA support ("ii") were 3.1% (i.e., 1/32), 6.2% (i.e., 2/32), and 90.7% (i.e., 29/32). The percentages of "-", "i", and "ii" for the Programming tasks

were 9.1% (i.e., 2/22), 22.7% (i.e., 5/22), and 68.2% (i.e., 15/22). The counterparts in Amber's case were 16.7% (i.e., 5/30), 33.3% (i.e., 10/30), and 50.0% (i.e., 15/30) for Abstraction tasks and 20.8% (i.e., 5/24), 54.2% (i.e., 13/24), and 25.0% (i.e., 6/24) for Programming ones. Three patterns stood out: 1) The percentage of "-" was lower than that of "i" and "ii". Generally, the SPARC unit was comprehensible to both students. 2) The percentages of "-" and "i" increased and the percentage of "ii" decreased from Abstraction to Programming for both students. Programming tasks seemed to be more difficult than Abstraction ones since both students needed more support from the TA or Mr. C's instructional videos. 3) The percentages of "ii" for both Abstraction and Programming in Tyler's case were larger than their counterparts in Amber's case. This difference might stem from Tyler's experience with programming that enabled him to accomplish CT tasks more independently.

Table 3

Task codes between the two students

Abstraction									
	A1-	A1(i)	A1(ii)	A2-	A2(i)	A2(ii)	A3-	A3(i)	A3(ii)
Tyler	0	0	2	0	1	20	1	1	7
Amber	0	0	1	1	6	12	4	4	2
Programming									
	P1-	P1(i)	P1(ii)	P2-	P2(i)	P2(ii)	P3-	P3(i)	P3(ii)
Tyler	0	0	1	2	0	2	0	5	12
Amber	1	1	0	3	4	2	1	8	4

A close look into the tasks revealed that the difficulties in Abstraction existed at the beginning when the two students built a model about family members, especially when variables were first introduced. The conversation below was an example in Lesson 4 when Amber worked with the TA on the task of "whom John is the father of" when there were three children. The expected answer was "father (john, Whom)" and "Whom" could be a capital letter or any words with the first letter capitalized. Amber failed this task probably because she was unfamiliar with the rule of abstraction associated with SPARC. Tyler's only "-" task in Abstraction was about family members as well. After entering the topic of atomic model, Tyler could follow the regulations to build SPARC models about an atom. In comparison, Amber encountered more difficulties. For example, Amber stated that "*atomic number is E and N*" when the task was to build a rule of "atomicNumber(E,

N)". After the TA asked that "*what do you mean by that?*", Amber answered with "*I don't know how to word it*" rather than the expected answer of "the atomic number of an element E is the number N".

"TA: *How should I write a query to see whom John is the father of? Assuming John is the father of those three children, how would I get that?* – [A3 task]
Long pause from Amber
TA: *We would be using variables for this.* – [TA support]
Amber: *Well, wouldn't you put father, Linda, Sara, and Peter?*
TA: *What's that?*
Amber: *Wouldn't you put X?* – [Miss the name of the rule]
TA: *So we know John is the father of somebody, but we don't know who somebody is, so our rule is going to start with father obviously, and we know we want to find whom John is the father of, so the first argument is gonna be John, right?* – [TA support]
Amber: *Yes.*
TA: *So what would we put for the next part?*
Long pause from Amber
TA: *You mentioned X.* – [TA support]
Amber: *Father, parenthesis, X, X, X?* – [Misunderstanding of "X" as a variable]
TA: *There only needs to be one X at a time. John is the father of X and X is the three children, so you don't need three Xs.* – [Task accomplished by TA]"

In regard to Programming, Tyler had two "P2-" tasks with one about family members and the other about the atomic model (Figure 2). In both cases, Tyler copied complex rules directly from Mr. C's instructional video. As shown in Figure 2, the rule of "*neutronsOf(E, N) :- massNumber(E, M), protonsOf(E, P), N = M – P.*" was to define "neutronsOf" with "massNumber" and "prontonsOf". It was one of the most challenging tasks in the entire unit. Thus, it was not quite surprising that Tyler could not generate this rule on his own. Tyler did not have any "P3-" and his "P3(ii)" was more frequent than "P3(i)", which suggests that he could debug his programming anytime when there were errors with occasional support from the TA. In comparison, Amber needed more support from the instructional video ("-") or the TA ("i"). There were only 6 out of 24 tasks where she could accomplish programming that worked properly. Amber's difficulties with programming varied. At the beginning of this unit, she was uncertain where she should put rules or queries. Amber also had difficulties transferring her existing knowledge to a new context. For example, the first rule in the topic of

atomic model is "symbolFor (E, S)". Amber could articulate the relation of "symbolFor (hydrogen, h)." as "*The symbol for the element of hydrogen is H*." However, she could not correctly type this relation on SPARC even with the prompt from the TA that "father (john, peter)." was the syntax for the relation of "John is the father of Peter." Amber improved her skill of programming from practice. For example, Amber was more aware of common errors like missing a period or the difference between upper- and lower-case letters as she programmed more. Yet, she made more errors in programming and needed TA support to fix them more frequently than Tyler did.

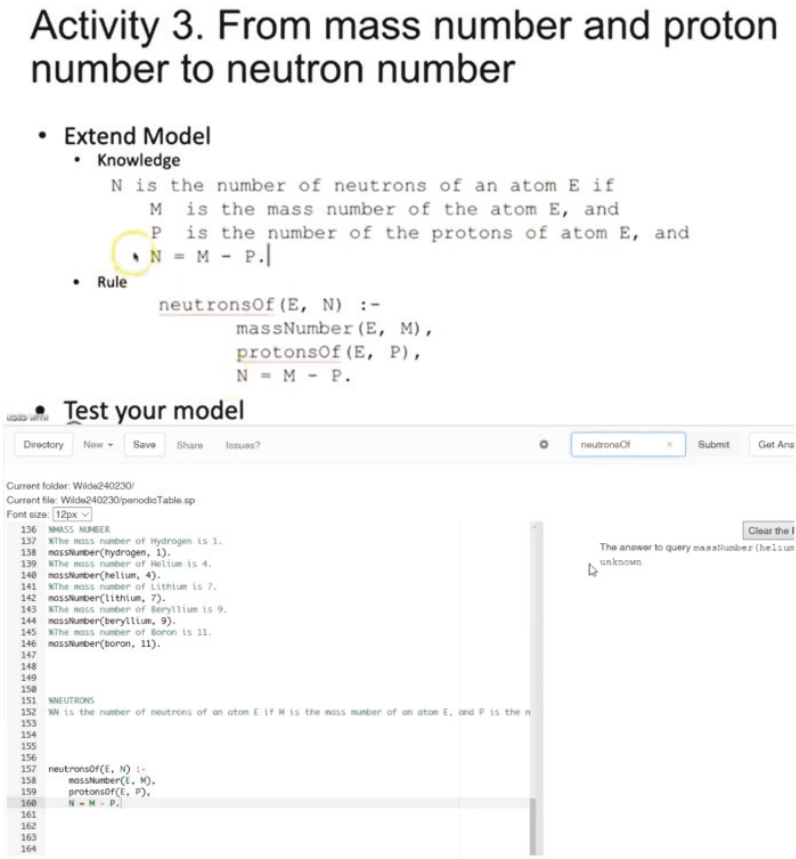

Figure 2. One "P2-" task from Tyler.

Q3. How did the two students perceive this SPARC unit?

In the post-interview, both students felt that computer science was important and should be incorporated in middle school. To them, computer science was "*to make computer learn something so it can answer questions* [Tyler]/ *make a software* [Amber]". They both commented positively about their experience with this SPARC unit. They perceived this unit as being interesting. Specifically, they both found the section of Rules most interesting and their reason was the same that Rules were the "computer language" to represent the knowledge that they knew. This is like language learners being able to communicate with native speakers using their language, which granted the leaners a sense of success. In addition, Tyler thought that Query was also interesting because "*There are so many ways that you can ask the same question.*" While being further asked which topic between family members and the atomic model was more appropriate to embed SPARC, both students selected the atomic model because it was more sophisticated. In Amber's words, "*Father and dad are basically the same words… with the atomic model, you got to teach more knowledge to the computer*." For both students, part of their interest towards SPARC seemed to come from their sense of competence of presenting sophisticated knowledge using SPARC.

"Tyler: *I also like rules because basically that's a large part of teaching computer something, that's where you put on the major details that computer knows the answer to the questions you are asking in query*.
Amber: *I think I find the rules most interesting. We are basically telling the computer using the computer language, I guess. I kind of play the role as a teacher and input the knowledge to computer*."

While being asked which aspect in the unit was most challenging, they both mentioned "Predicates" because it contained different symbols that required a stronger skill of programming, especially debugging. In Tyler's words, "*If you mistype one little thing, if you make a capital letter a lower case and vice versa, one little thing messes up everything and it won't work… Debugging would be most challenging because you got to find little bitty mistakes.*" However, they held different attitudes toward this challenge. Tyler enjoyed this challenge whereas Amber found this challenge less interesting and would love to bypass it. This difference might because Tyler "*always enjoyed learning how things work*" or he had experience with programming.

"Tyler: *I do enjoy it [debugging]. I think it's something like solving the mystery, because whenever the query would say all this stuff that we don't know what it means, I would have to figure out what it's telling me to do and where it's telling me to do it, I would have to go through and figure that out. I enjoy figuring that out*.
Amber: *When I did it [programming], I kept saying, wait, there was an error. Sometimes it was frustrating… I prefer someone else fix the error for me*."

As for the instructional strategy, they both preferred the instantaneous support from the TA over Mr. C's instructional video. Like Amber said "*He [the TA] kind of taught me through it and Mr. C just explained it. The TA was more interactive*." Amber was unsure whether this SPARC helped her understand the atomic model better. In comparison, Tyler acknowledged that SPARC reinforced his memory of the atomic model through practice.

"Tyler: *Yes, I think it [SPARC] did help. I remember in science we have already done the periodic table and learned a lot at there, but we never did any activities trying to memorize it…So I am thinking, if I had done SPARC programming before that, it probably could have been much easier since I memorized more of it*."

While being asked whether other students without experience with programming would feel the same way, Tyler stated that "*I think they [Other students without programming experience] would like it [SPARC] because I know some people dislike science more than STEM, because in STEM they got to work with technology. In science, it was more answering questions and all that. STEM is a little more active than science was. I feel it would be more fun.*" He believed that SPARC as a technology would make science more interesting to some students.

Considering that Tyler had experience with Java and Python, we asked for his opinion about programming with SPARC. First, he commented that "*Compare to Java and Python, I think SPARC is super easy. If [the symbol] is the hardest part, but not that hard compared to Jave or Python*." He believed that SPARC "*can be kind of a step stone to Java or Python*". His comment justified our hypothesis that SPARC is more friendly to middle-school students and also represents the authentic work of computer scientists. Tyler did not think that his programming experience with Java and Python could automatically transfer to the skill of programming with SPARC. He began with not paying attention to "*the stuff I was programming*", but to "what *the program was doing*". Once he got used to it, he started to "*pay*

more attention to what I was programming." Based on Tyler's comments, it seemed that an effective integration of SPARC and science requires basic knowledge of both areas. Thus, a reasonable sequence of integration probably is students learning SPARC in the context with which students are familiar, learning a scientific concept, and building a SPARC model of that concept to strengthen the knowledge.

DISCUSSIONS AND IMPLICATIONS

This study describes our efforts of integrating computer science education into middle schools through the programming language of SPARC. Like many existing studies at the K-12 level (Basawapatna, Koh, & Repenning, 2010; Marcu et al., 2010; Sengupta et al., 2013), we witnessed positive affective learning outcomes such as student engagement and interest. In addition, we did see a cognitive learning outcome of the students' competence of CT. We clearly defined a K-12 friendly version of CT as the combination of Abstraction and Programming (Grover & Pea, 2013; Weintrop et al., 2016; Wing, 2006; Wing 2011). The former is about the ability of identifying objects in a phenomenon, building relations among the objects, and generalizing patterns. This is aligned with model building in science education (Hestenes, 1987; Sengupta et al., 2013; Weintrop et al., 2016), which was probably why Tyler felt that the SPARC unit strengthened his knowledge of the atomic model. In other words, representing the atomic model with SPARC is beyond the repetition of facts to the organization of the facts in a logical order. The latter, programming, is about the communication using the computer language and algorithmic reasoning of following rigorous steps to solve a problem. Based on the findings, around 80% of the CT tasks were successfully accomplished by both students regardless of their prior experience with programming (i.e., Tyler) or not (i.e., Amber). The difference lay in their spontaneity and independency while accomplishing a task. Thus, it could be inferred that CT defined in this study was not only measurable but also achievable to regular students. In addition, we believe that our rubric (Table 1) for CT assessment is applicable to other text-based programming languages.

Our findings also suggest the feasibility of integrating SPARC with middle-school science (Grover & Pea, 2013). Neither students demonstrated any difficulties with applying SPARC in the context of the atomic model. In fact, they preferred the atomic model over family members because it granted them a stronger sense of success. Prior to this unit, both students demonstrated the same pattern of fragmented memory and superficial un-

derstanding of the atomic model. This situation is common since the content knowledge required by national or state standards (e.g., Texas Essential Knowledge and Skills) sometimes is limited to the memorization of basic facts at the middle-school level. Tyler and Amber's science teacher did not teach them the mechanism behind the equations about proton, neutron, and electron numbers probably because concepts like being electronically neutral were beyond the standards. Thus, students would need opportunities to apply an abstract concept when they are unlikely to observe its use in daily lives. In this sense, integrating SPARC into science is probably more practical and beneficial than opening a separate course about computer science alone. In this study, it was unclear how CT overlapped with scientific thinking demonstrated by the two students (Sengupta et al., 2013; Weintrop et al., 2016). However, we observed the reciprocal benefits associated with this integration as computer modeling granted students the opportunity of knowledge application and the context of science made computer science more meaningful. Considering the importance of background knowledge (Marcu et al., 2010; Sengupta et al., 2013), we would suggest using SPARC sequentially after students have learned a scientific concept rather than parallelly at their early access.

The success of this SPARC unit also supports the feasibility of exposing middle-school students to programming. Despite its complexity (Grover & Pea, 2013), programming cannot be ignored because it is a critical part of computer scientists' work (Wing, 2011). Both Tyler and Amber referred to programming as a challenge. Although Amber disliked programming, especially debugging, she managed this skill gradually with the support from the TA. She accomplished 79.2% of the programming tasks, but probably was unaware of that achievement. Compared to reiterating how complex programming could be, Amber might need the meta-cognition more about her competence of programming. We do not argue that programming has to be mandatory to all middle school students. However, we also disagree with depriving students of the chances of programming when they have the potential to manage this skill. The importance roots in how teachers can shield students from the frustration of debugging before they have been used a new computer language. SPARC has the advantage over other text-based programming languages (e.g., Java) because it is simple. The "low floor" strategy of starting with the model of family members could be effective for students to concentrate on the basics of programming. Another helpful strategy is the separation between Abstraction and Programming. Deemphasizing the rigor of syntax would ease learners' anxiety from constructing a computer model directly on a programming platform.

Acknowledgement

This work is supported by the National Science Foundation (NSF) of the United States under grant number 1901704 to Texas Tech University. The opinions, findings, and conclusions or recommendations expressed are our own and do not necessarily reflect the views of the National Science Foundation.

References

Aho, A. V. (2012). Computation and Computational Thinking. Computer Journal, 55(7), 832- 835. doi:10.1093/comjnl/bxs074.

Ary, D., Jacobs, L. C., Razavieh, A., & Sorensen, C. (2010). *Introduction to Research in Education (8th edtion)*. Thomson Learning, Belmot, CA: Wadsworth.

Balai, E., Gelfond, M., & Zhang, Y. (2013). Towards answer set programming with sorts. In *International Conference on Logic Programming and Nonmonotonic Reasoning* (pp. 135-147). Springer, Berlin, Heidelberg.

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, *38*(6), 20-23.

Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010). Using scalable game design to teach computer science from middle school to graduate school. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 224-228). ACM.

Blikstein, P., & Wilensky, U. (2009). An atom is known by the company it keeps: A constructionist learning environment for materials science using agent-based modeling. *International Journal of Computers for Mathematical Learning*, *14*(2), 81-119.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*(1), 37-46.

Corbin, J. M., & Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, *13*(1), 3-21.

García-Peñalvo, F. J., & Cruz-Benito, J. (2016). Computational thinking in pre-university education. In *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality* (pp. 13-17). ACM.

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

Grover, S., Pea, R., & Cooper, S. (2014). Remedying misperceptions of computer science among middle school students. In *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 343-348). ACM.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199-237.

Grover, S., Pea, R., & Cooper, S. (2016). Factors influencing computer science learning in middle school. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 552-557). ACM.

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin*, *41*(1), 183-187.

Hestenes, D. (1987). Toward a modeling theory of physics instruction. *American journal of physics*, *55*(5), 440-454.

Kelleher, C., Pausch, R., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1455-1464). ACM.

Kowalski, R. (2014). Logic programming. In *Handbook of the History of Logic* (Vol. 9, pp. 523-569). North-Holland.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. *Educational Researcher*, *14*(5), 14-29.

Medvidovic, N., Taylor, R. N., & Whitehead Jr, E. J. (1996). Formal modeling of software architectures at multiple levels of abstraction. *ejw*, *714*, 824-2776.

Marcu, G., Kaufman, S. J., Lee, J. K., Black, R. W., Dourish, P., Hayes, G. R., & Richardson, D. J. (2010). Design and evaluation of a computer science and engineering course for middle school girls. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 234-238). ACM.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004). Scratch: a sneak preview [education]. In *Proceedings. Second International Conference on Creating, Connecting and Collaborating through Computing, 2004.* (pp. 104-109). IEEE.

National Research Council. (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: The National Academies Press.

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

Perrenet, J., Groote, J. F., & Kaasenbrood, E. (2005). Exploring students' understanding of the concept of algorithm: levels of abstraction. *ACM SIGCSE Bulletin*, *37*(3), 64-68.

Royal Society (2012). Shut down or restart: The way forward for computing in UK schools. [2016, May 24] https://royalsociety.org/~/media/education/computing-in-schools/2012-01- 12-computing-in-schools.pdf

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351-380.

Settle A, Franke B, Hansen R, Spaltro F, Jurisson C, Rennert-May C, Wildeman B (2012) Infusing computational thinking into the middle- and high-school curriculum. In: Proceedings of the 17th ACM conference on Innovation and technology in computer science education. ACM, New York, pp 22–27

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127-147.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: measuring computational thinking in middle school. *Proceedings of the 43rd ACM technical symposium on Computer Science Education, SIGCSE* (pp. 215-220). New York, NY, USA: ACM

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–36.

Wing, J. (2011). Research notebook: Computational thinking—What and why? *The Link Magazine*, Spring. Carnegie Mellon University, Pittsburgh. Retrieved from http://link.cs.cmu.edu/article.php?a=600