Invited Article

TEACHING COMPUTATIONAL THINKING TO ENGLISH LEARNERS

Sharin Jacob*
Ha Nguyen
University of California at Irvine

Colby Tofel-Grehl
Utah State University

Debra Richardson
Mark Warschauer
University of California at Irvine

Computational thinking is an essential skill for full participation in society in today's world (Wing, 2006). Yet there has been little discussion about the teaching and learning of computational thinking to English learners. In this paper, we first review what computational thinking is, why it is important in education, and the particular challenges faced in teaching computational thinking to speakers of English as a second language. We then discuss some approaches for addressing these challenges, giving examples from two recent K–12 initiatives in which we have been involved.

Keywords: coding, computational thinking, computer science, elementary school, STEM

Computational thinking represents an analytic approach to solving problems utilizing concepts essential to computing (Wing, 2006). Stephen Wolfram succinctly describes computational thinking as the ability to formulate thoughts and questions in a manner that is communicable to a computer to achieve desired results (Weber, 2018). Consensus on an exact definition of computational thinking has not yet been achieved (Barr & Stephenson, 2011; Grover & Pea, 2013), but in general, scholars agree that computational thinking skills include automation, abstraction, algorithmic thinking, modularization, and data analysis (International Society for Technology in Education & Computer Science Teachers Association, 2011). Abstraction is foundational to computational thinking (Bennedsen & Caspersen, 2006; Kramer, 2007). Given this requirement, certain dispositions or mindsets are fundamental to being successful in computational thinking, including positive attitudes toward mistakes, ambiguity, complexity, persistence, communication, and multiple paths to solutions (International Society for Technology in Education & Computer Science Teachers Association, 2011). In practice, computational thinking involves navigating multiple layers of abstraction at any given time, identifying which components to include or exclude within the scope of a given problem or model. Algorithms represent the automation of these abstractions (Wing, 2006)—for example, by programming the instructions a computer can use to carry out a specific set of tasks in a particular order to solve a problem. Automated processes permeate our daily

lives, ranging from percolators that brew coffee, to thermostats and stop lights for environmental and traffic control, to self-checkout machines in grocery stores, ATMs in banks, and digital personal assistants in our homes. Each of these automated applications aggregates a host of algorithms, some of which may be relatively simple while others may feature dizzying levels of abstraction.

There are key similarities and differences between computer science, computational thinking, and coding. While computer science refers to the study of computers, computational thinking represents an approach that is generalizable to a broad array of disciplines. Computational thinking skills are critical to solving long-standing problems in the biological, physical, and social sciences, while providing foundational tools for the nascent study of the digital humanities. Although computational thinking is often operationalized through computer programming, its instruction does not require the use of computers (Bell, Alexander, Freeman, & Grimley, 2009), but rather can be enacted through an "unplugged" approach. Unplugged activities provide multimodal avenues for students to learn essential computing concepts, which typically are later reinforced through computer programming exercises.

Computational thinking can also be taught outside the context of computer science through cross-curricular integration. While there is a plethora of research on integrating computational thinking with Science, Technology, Engineering, and Mathematics (STEM) subjects (Jona et al., 2014; Weintrop et al., 2016), there is an emerging literature on its integration with literacy curricula. Jacob and Warschauer (in press) developed a three-dimensional theoretical framework for exploring the relationship between computational thinking and literacy (computational thinking *as* literacy, computational thinking *through* literacy, and literacy *through* computational thinking). This conceptual approach explores ways in which literacy skills facilitate computational thinking and, conversely, how students' existing computational thinking skills can be leveraged to promote literacy development while illustrating pedagogical implications for such integration.

Why Teach Computational Thinking?

Given the pervasiveness of computational artifacts in society and our daily lives, students are increasingly required to adopt computational approaches in solving everyday problems. Meanwhile, with the rise of automation and artificial intelligence, economists predict that up to 800 million jobs will be automated by 2030 (McKinsey Global Institute, 2017). Success in this dynamic workforce requires students to think computationally and navigate multiple levels of abstraction to find innovative solutions for perplexing problems. Beyond workforce preparation, developing students' computational thinking skills fosters civic engagement, allowing students to participate as scholars who increasingly utilize computation to ameliorate social ills, whether in search of a cure for cancer or the elimination of hunger. Scholars argue that these computational thinking skills have such a pervasive impact on social communication and interaction that they represent fundamental literacies (diSessa, 2000; Jacob & Warschauer, in press). Despite these trends, educational institutions continue to teach discrete reading, writing, and mathematics skills while failing to emphasize the development of computational thinking skills. Updating current pedagogical practices through educational policy initiatives can emphasize these skills and dispositions by explicitly integrating computational thinking objectives into the curricula of core subject areas.

Although efforts to teach computational thinking in K–12 schools have been promising, the United States lags behind other nations in training its students in computer science, largely due to the lack of a systematically mandated computer science curriculum. Efforts dedicated to broadening participation in computer science, such as the Computer Science for All initiative enacted by Obama in 2016, seek to teach all students, and especially those from traditionally underserved backgrounds, to become developers, rather than simply consumers, of technology (Smith, 2016). Despite these efforts, shortages in

the STEM workforce are even more severe in the computer science area, where only 500,000 students will graduate by 2020 to fill 1.4 million positions nationwide (Bureau of Labor Statistics, 2015). English learners, who as one of the fastest growing populations in U.S. schools remain dramatically underrepresented in computer science courses and careers (Martin, McAlear, & Scott 2015), present a valuable resource to address this need. While Latinos constitute 54% of California K–12 enrollment (Ed Data, 2018), they represent only 22% of Advanced Placement Computer Science test takers in the state (College Board, 2017). Furthermore, Latinos, low socio-economic students (SES), and English learners receive 50% less computer science instruction than do their peers (Martin, McAlear, & Scott, 2015). Realizing these students' underdeveloped talents would not only greatly benefit them but also enhance the potential for future U.S. technological innovation and progress.

In attempts to broaden participation of students from traditionally underserved groups, researchers have tried to identify and target the underlying causes of underrepresentation in computer science. Findings indicate that compared to their privileged counterparts, students from multicultural, low-SES backgrounds lack computer and internet access at home and school (McFarland et al., 2017) and perceive fewer role models from diverse backgrounds working in computer science fields (Royal & Swift, 2016). Accessibility and visibility are further hindered by the lack of representation in the media of computer scientists from culturally diverse backgrounds (Royal & Swift, 2016). These factors are exacerbated by a lack of computer science exposure at home and at school (Google & Gallup, 2015; Wang, Hong, Ravitz, & Moghadam, 2016). Increasing students' available resources for, exposure to, and identification with the computer science discipline is critical to addressing underrepresentation of students from multilingual, multicultural backgrounds (Mercier, Barron, & O'Connor, 2006; Packard & Wong, 1999; Teague, 2002).

Challenges in Teaching Computational Thinking to English Learners

Teaching computational thinking to English learners brings its own challenges and opportunities in terms of content, cognitive and linguistic demands, and widespread stereotyping against certain groups of learners. The strict syntactic demands of coding often limit student productivity by increasing time spent on debugging and error correction (Bennedsen & Caspersen, 2012). Text-based programming languages are unintuitive and challenge emerging readers; content demands are exacerbated by lower levels of computer and internet access that hinder opportunities to practice for students from culturally and linguistically diverse backgrounds (McFarland et al., 2017). Furthermore, much of the existing curricula typically lack culturally responsive materials that motivate students by bridging home and formal learning environments (Brown & Doolittle, 2008). Taking into account their content needs, differential access to technology at home, and diverse family and cultural backgrounds, providing access for these students requires not just technological resources, but also new pedagogical tools for engagement.

In addition to content demands, successful computational thinking also requires particular dispositions or mindsets (Goode, Margolis, & Chapman, 2014; International Society for Technology in Education & Computer Science Teachers Association, 2011). The process of debugging and troubleshooting calls for persistence, comfort with ambiguity, and a positive view of making mistakes. Students from affluent homes who have had previous exposure to computer science enjoy the advantage of acquiring problem-solving strategies unique to computer science elsewhere—and oftentimes through prior informal and formal learning environments. Furthermore, unlike math and science, computational thinkers must develop the ability to deal with open-ended problems (International Society for Technology in Education & Computer Science Teachers Association, 2011). In addition, computer science problems involve multiple solutions, which makes computational thinking difficult to assess (Fuller et al., 2007), rendering these assessments prone to teacher beliefs and biases. While efficiency, simplicity, elegance, and usability represent some of the criteria used to measure the quality of abstractions in computational thinking

(Wing, 2006), in any given problem seemingly ineffective or mistaken solutions may lead to one of many potentially correct solutions. Tracking these myriad approaches to a solution presents a challenge for many students—particularly English learners, who may face linguistic challenges in articulating their own problem-solving processes and solutions. Teacher misperceptions and faulty beliefs about these students may result in misdiagnosing errors in a student's work when the student is, in fact, practicing novel and innovative approaches to problem solving (Ryoo, Lee, Sandoval, & Goode, 2013). To promote equitable instruction for linguistically diverse students, teachers need to recognize the nature of computer science content and develop an in-depth understanding of students' problem-solving processes. Further research is needed on the types of linguistic scaffolds and supports that develop students' computational thinking skills and facilitate their acquisition of content knowledge in computer science.

Language learners from marginalized backgrounds also confront pervasive stereotyping in computer science. Computer scientists are often perceived as nerdy males who wear unglamorous glasses and possess inborn, prodigious talent (Aspray, 2016), a stereotype fed by beliefs that interests, talents, and abilities are innate to certain, often privileged, groups (Margolis, 2010). Rather than viewing achievement gaps in computer science as the result of gaps in students' innate abilities, culturally responsive approaches recognize the systemic sociocultural and historical inequities that lead to differential learning opportunities for students from underserved groups. Diversity initiatives, such as Science for All (Lee & Fradd, 1998) and Computer Science for All (Smith, 2016), recognize these disparities and maintain that all students are capable of achievement regardless of cultural, linguistic, and socioeconomic backgrounds. Furthermore, these initiatives promote instructional practices that draw upon students' wealth of cultural and linguistic resources to enrich learning and promote identification with the field.

Approaches to Teaching Computational Thinking to English Learners

Although research on quality computer science instruction for language learners is sparse, effective instructional practices for English learners in STEM have been well established in these findings: (a) engaging language learners in science and math requires intensive linguistic scaffolding to understand discipline-specific discourse structures and demanding technical language (Snow & Katz, 2010); (b) providing English learners multiple opportunities to practice problem-solving skills in language-rich environments allows them to simultaneously develop academic language proficiency and content knowledge (Lee & Fradd, 1998); (c) engaging language learners in inquiry-based, collaborative peer-to-peer talk motivates students to use newly acquired language (Zwiep & Straits, 2013); and (d) integrating these instructional practices with culturally responsive materials connects the STEM curriculum to students' lives and communities (Brown & Doolittle, 2008). Consequently, linguistic scaffolding and culturally responsive pedagogies can be both supportive and effective in the instruction of computational thinking. Because a relationship exists between computing and students' sense-making—students use informal language and everyday experiences to inquire about and explain algorithm compositions—teachers can thus build instruction on the intersections between students' everyday knowledge and computational thinking practices.

A focus on students' sense-making in inquiry-based learning allows them to learn and retain computational thinking patterns more than does teacher-directed instruction (Ioannidou, Bennett, Repenning, Koh, & Basawapatna, 2011). Inquiry-based learning in computer science includes practicing computational thinking in drafting initial approaches to problem solving and experimenting with multiple strategies; such inquiry also entails the teaching and learning strategies that allow for students' hands-on investigations to uncover major concepts, instead of memorization of discrete facts (Goode, Chapman, & Margolis, 2012). Inquiry-based learning approaches that positively contribute to the knowledge, skills, and

attitudes in computer science among diverse learners can also be scaled to larger programs (Margolis, Goode, & Binning, 2015).

Linguistic Scaffolding

The vocabulary, syntax, and features in academic language used for describing computational thinking processes are distinct from everyday language. When learning computing and computational thinking, students are expected to acquire content-specific vocabulary (e.g., algorithm, loop) and distinguish between nontechnical terms and their common-usage counterparts (e.g., steps, repeat). The functions of inquiry-based learning in computer science, including describing and interpreting data, proposing solutions, and communicating findings, might be unfamiliar to English learners in particular.

Explicit vocabulary instruction is an instrumental approach to scaffolding student learning of both content-specific and general language conventions (Buxton, Lee, & Santau, 2008). Teachers can model the computational thinking concepts in everyday language and then provide vocabulary instruction after students have mastered the concept. For example, teachers can explain the idea of an algorithm as a series of connected steps by giving an everyday example of students' morning routine as an algorithm: getting up, brushing teeth, eating breakfast, going from home to school. Teachers then would introduce the word "algorithm" and explain it as a list of steps written for the computer to accomplish a task. Studies suggest that this delayed approach avoids overtaxing working memory and results in more effective learning of both concepts and language (Brown & Ryoo, 2008; Ryoo, 2015).

It is important that teachers consider students' oral and written language development in learning computational thinking. Studies have documented the benefits of creative computing to developing literacy skills among traditionally marginalized youth. Peppler and Warschauer (2012) observed how Brandy, a nine-year-old girl with cognitive disabilities, developed the metalinguistic awareness to improve her reading and writing ability through programming multimedia artifacts. Brandy began to make the connection between reading the code blocks and combining them in semantically meaningful ways. Through the process, Brandy regained her interest in the traditional literacy form of reading and writing and took up a more central social position in the after-school computer clubhouse (Peppler & Warschauer, 2012). Just like the researchers' and after-school staff's noticing of Brandy's emergent text-making abilities, teachers should monitor the language development of English learners. Formative assessment of students' discourse and artifacts provides teachers the opportunity to take note of and support students' emergent text-making abilities, underscoring the meaning of computational thinking concepts and their use in both everyday contexts and programming environments.

The development of literacy skills also occurs in group discussion and student collaboration. During those activities, several students have opportunities to practice computational thinking discourse and build on each other's ideas. Such discourse is optimal for learning computer science content and language if it is part of a process of collaborative inquiry-based learning (Fradd, Lee, Sutman, & Saxton, 2001; McNeill & Krajcik, 2007). Peppler, Warschauer, and Diazgranados's (2010) study of diverse elementary students in peer game-critique groups provides a compelling example of the benefits of collaborative discourse. Students who learned to critically evaluate peer-created videos in talk and text were found to enhance both their computational thinking and language skills (Peppler et al., 2010).

Culturally Relevant Curriculum

Computer science education that values students' agency, sociocultural background, and authorship significantly engages students, especially those who are traditionally underrepresented in computing (Ryoo et al., 2013). Intervention for English learners should make computational thinking relevant by drawing from students' own funds of knowledge and contexts (Basu & Barton, 2007). Students from

underrepresented groups often favor relational learning—learning together with peers and making connections between learning and their communities and culture—over noncollaborative or competitive approaches that make them feel isolated (Anderson & Adams, 1992). Culturally relevant teaching that values interdependence and collaboration would therefore validate student identities and backgrounds beyond instruction of content knowledge, and prepare English learners for the demands of creative thought and social negotiation in developing computational thinking.

There are many approaches to teaching computing and computational thinking to English learners in culturally responsive ways. For example, teachers can ask students to create digital storytelling projects or identity texts, dual-language artifacts that draw on students' backgrounds, families, and interests in programming environments. Teachers can also utilize pair programming, where two students work simultaneously on a program, code, or design: one student plays the role of the "driver," actively writing codes and controlling the keyboard, while the other student becomes the "navigator," checking the correctness and efficiency of the program. Students who pair program perceive more confidence and enjoyment in their work, produce higher quality programs, and are more likely to persist in computing than those who do not (McDowell, Werner, Bullock, & Fernald, 2006). The notion of pair programming can be extended to collaborative learning to create multimedia products such as games, music, and models and simulations; these activities allow students to build learning communities as well as explore personal backgrounds and interests. Teachers can also showcase examples of computer science applications in cultural designs, which allows students to connect computational thinking practices to their own lives (Goode et al., 2012). This process will enhance students' computational thinking skills while enriching their identity, cognition, and language use (Cummins, Hu, Markus, & Montero, 2015).

In the following sections, we present two examples of responsive teaching in which we are involved that are both culturally and linguistically sensitive: CONECTAR, a National Science Foundation (NSF)-funded project that focuses on integrating computer science into the English Language Arts curriculum in elementary schools, and Project STITCH, an NSF-funded project that developed the curricular units and professional development program to introduce electronic textiles (e-textiles) into middle school curricula.

CONECTAR

CONECTAR (Collaborative Network of Educators for Computational Thinking for All Research) is a project created by the research-practice partnership between the University of California-Irvine (School of Education and School of Information and Computer Sciences), the Orange County Department of Education, and the Santa Ana Unified School District (SAUSD). The project's goal is to develop and pilot instructional materials for teaching computational thinking in Grades 3–5 in the SAUSD, targeted at the district's Latino students (96%) and English language learners (60%). As such, it is among the first to examine the linguistic and sociocultural processes that underlie English learners' success in mastering computational thinking; it also examines the role of computational thinking in an English Language Arts curriculum (Jacob & Warschauer, in press).

In the first year of the project (2017–2018), five teachers piloted a draft curriculum (adapted from the Computer Science for All in San Francisco initiative; see Smith, 2016) using Scratch, a block-based programming environment, in their classroom. In this plan, ongoing feedback on the implementation process and collaborative curricular development between researchers and teachers allows the project to develop approaches to teaching computational thinking in ways that meet the needs of English learners in the SAUSD. The collaboration not only happens during the school year, but also occurs during a week-long summer institute, where teachers and researchers modify the lessons based on experience in the classroom, integrate linguistic scaffolding into each lesson, and micro-teach to reflect on instructional strategies. At the time of this writing, the first summer institute has been completed and teachers have

successfully implemented development of the first year of the curriculum based on the principles of linguistic scaffolding and culturally relevant pedagogy elaborated below.

Linguistic Scaffolding

Explicit vocabulary instruction. To begin the project, teachers utilize a range of strategies to introduce computational thinking concepts to students, conducting explicit instruction with vocabulary cards in introductory units, when students first experiment with the basic elements of Scratch. Teachers model the vocabulary in everyday terms and engage students in exploratory activities in the programming environment before restating the target vocabulary. The vocabulary cards become class resources that students can refer to in subsequent activities, with teachers re-emphasizing concepts as they come up in specific lessons.

Emergent literacy skills. We worked collaboratively with the teachers to embed computer science and language objectives into each lesson and to develop the linguistics frames for students' sense-making of computational thinking. When learning computational thinking, students use everyday language at varied levels of sophistication to explain concepts, negotiate code meanings, and propose alternative solutions. The frames are developed for three language proficiency levels: emerging (low), expanding (medium), and bridging (high). In theory, the linguistic frames are grounded in the systemic functional linguistics perspective, which states that language is tightly woven in social contexts and that the language meaning-making process constantly adapts to changing human interaction (Halliday, 1973). In addition, the frames draw on the parallels between language and programming: both rely on syntactic sequencing and social negotiation among speakers/programmers to create meaningful constructs (Grover, 2015). In practice, teachers can model the sentence frames to students and use reminders (e.g., flashcards, placemats, handouts) to encourage students to utilize more advanced academic discourse. The linguistic frames are useful for formative assessment, as teachers monitor and facilitate emerging literacy skills in individual reflection and group discussion of student programs.

Culturally Relevant Curriculum

Our approach in this project to promote culturally relevant curriculum is twofold: enhance students' identification with computer science and engage students in culturally responsive pedagogies. First, the curriculum gives students the space to explore and create interest-driven and personal artifacts. At the end of each of the five units, students participate in a culminating activity that builds on the concepts they have learned. For example, the final unit focuses on creative storytelling, where students have a chance to write and program an interactive story under the theme "Choose your own adventure." The open-ended nature of the assignments and the possibility to reuse and build on existing works from the Scratch community allow for the exploration and showcasing of students' identities, while scaffolding students at different levels of programming and language competence. This approach is similar to Burke and Kafai's (2010) proposal that coding can reinforce programming and composition skills, especially when the projects are of personal interest to students. In addition, the curriculum includes multiple activities—namely, pair programming, group debugging, and peer critiques—that facilitate student relational learning and collaborative discourse. The culmination of student artifacts also helps teachers track students' progress in attaining computational thinking and literacy skills.

Second, we work with teachers to compile English Language Arts lesson plans that include storybooks about computational thinking processes (e.g., problem-solving) and inspirational women scientists. Studies have shown that students embrace relatable role models and the qualities associated with them in the STEM fields (Aish, Asare, & Miskioğlu, 2018). The storybooks aim to open up the conversation with

students to increase their ability belief, motivation, and identification with STEM in general and computer science-related fields in particular (Wang & Degol, 2017).

Project STITCH

Project STEM Teaching Integrating Textiles and Computing Holistically (STITCH) is a curriculum and professional development project designed to facilitate the evolution of a curricular approach to STEM content that integrates computer science into secondary classrooms. Using electronic textiles (e-textiles), Project STITCH requires students to program microprocessors to gather and process the data needed to solve a range of authentic problems drawn from physics, chemistry, earth science, and life science in Grades 6–12. The project allows students to explore the process of designing solutions to fit everyday problems. E-textiles are sewable, often wearable, projects that involve sewing microprocessors to actuators such as LED lights, buzzers, and sensors with conductive thread. By sewing circuits using traditional crafting materials and new sewable technologies, students design solutions that are intellectually rigorous as well as culturally and personally meaningful.

In the first year of the project (2016–2017), 18 secondary teachers received training and subsequently taught Project STITCH in their classrooms. Focused feedback from teachers led to curricular revisions and instructional scaffolding to better meet the needs of native and Latino populations being served across the Intermountain region of the western United States. Findings indicate tremendous potential for e-textiles to help shape both student and teacher perceptions of who engages with science (Howell, Tofel-Grehl, Fields, & Ducamp, 2016).

Scaffolding for Identity

Because early exposure to meaningful and relevant science experiences acts as a predictor of future science career interest (Tai, Lui, Maltese, & Fan, 2006), engaging underrepresented students in science early on is extremely valuable. One way to promote student participation in science is to provide them with opportunities to engage in projects that capture their interest. Embedding science learning in the context of students' everyday lives and culturally significant practices (Petrich, Wilkinson, & Bevan, 2013), such as e-textiles, provides a meaningful way to engage new student populations in STEM.

Without the opportunity to connect with science personally, students tend to retain—and magnify—negative feelings toward science (Basu & Barton, 2007). E-textiles offers culturally responsive opportunities for English learners and other underrepresented students to engage in designing and making circuitry (Searle & Kafai, 2015). While much work with English learners focuses on skills development, Project STITCH puts forth a fused model of skill and identity development. When making projects using e-textiles, greater participation is noted from students who previously did not engage in science because e-textiles provides an opportunity for these students—that is, compared to most other science instruction, it is language neutral. This allows those students whose understanding and contributions were limited by language an opportunity to shine.

Scaffolding Using Explicit Instruction

Project STITCH engages students in a faded-scaffold approach to supporting the development of computing and computational thinking. Students are encouraged to use code not of their design, which allows them to consider the applicability of the technology before decomposing it. Students then move into modifying the code, adjusting the timing and purposes of the light blinks on their initial project. After students use the code in this way, they work collaboratively to comment and translate the code, which can be scaffolded in either English or a student's native language. This process affords teachers many opportunities and multiple strategies for differentiation, and also allows for assessment of conceptual

understanding independent of a student's language skills. In addition, such activity is also scaffolded through the use of group work to support language learners. After students achieve success in their commenting and translating of code, they move into remixing and writing simple code for their projects.

Culturally Relevant Curriculum

Project STITCH's approach to engaging culturally relevant learning is two-pronged. First, we seek to improve students' engagement through a language-neutral set of projects that allow for science learning beyond the confines of a worksheet. We work to improve student identity by disrupting normative classroom discourse structures and roles in order to provide a new entry point for students with less linguistic space within the classroom. Because e-textiles-based learning involves making hands-on models of computational circuits, students are engaged in multiple ways throughout various aspects of the project. This approach has led to greater buy-in and engagement from students during project work, which in turn has encouraged positive shifts in student perception of their teacher's support of them in learning science (Tofel-Grehl et al., 2016). Second, Project STITCH aims to address ways of shifting teachers' private misconceptions of who can successfully engage in science. When teachers work with students on e-textiles projects, they often report changes in their own perspective on who can do science. These shifting teacher perceptions, coupled with changing student beliefs, create a powerful fulcrum for creating a more open and diverse community of learners in science.

Finally, e-textiles also create terrific opportunities to connect to English learners' families and communities by taking advantage of the prominent role of textiles and sewing in many immigrant communities. As a middle school teacher in our project explained:

> For the first time in their academic careers, many of my Latino students received instruction and help with their homework from their parents or family members. Many Latino parents in our town do not speak English and often express their frustrations of not understanding their children's homework, not being able to help their children, and not knowing what homework is due. E-textiles helped generate interest in my students' schoolwork through something as simple as sewing. (Tofel-Grehl & Searle, 2017, pp. 8–9)

Conclusion

Effective teaching of computational thinking to English learners overlaps substantially with other forms of content-based instruction. At the same time, as seen in the discussion above, computational thinking has distinct characteristics that create both challenges and opportunities. Analysis of computer code can be used to build meta-awareness of computational semiotics, and the visual nature of certain programming languages, such as Scratch, can scaffold literacy development. Most important, projects involving computational thinking, whether in creating stories or making e-textile projects, provide ample opportunities for students to express and develop their own identity—an important element of the successful second language curriculum.

The entire field of computational thinking in education is now taking shape. As it does, it will be critical for TESOL educators to put their stamp on it, so that the way we teach computational thinking best meets the needs of our diverse students.

References

Aish, N., Asare, P., & Miskioğlu, E. E. (2018, March). *People like me: Providing relatable and realistic role models for underrepresented minorities in STEM to increase their motivation and likelihood of success.* Paper presented at the 2018 IEEE Integrated STEM Education Conference (ISEC). Princeton, NJ.

Anderson, J. A., & Adams, M. (1992). Acknowledging the learning styles of diverse student populations: Implications for instructional design. *New Directions for Teaching and Learning*, *49*, 19–33.

Aspray, W. (2016). *Participation in computing: The National Science Foundation's expansionary programs*. New York, NY: Springer.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K–12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48–54.

Basu, S. J., & Barton, A. C. (2007). Developing a sustained interest in science among urban minority youth. *Journal of Research in Science Teaching*, *44*(3), 466–489.

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology, 13*(1), 20–29.

Bennedsen, J., & Caspersen, M. E. (2006). Abstraction ability as an indicator of success for learning object-oriented programming? *ACM SIGCSE Bulletin*, *38*(2), 39–43.

Bennedsen, J., & Caspersen, M. E. (2012). Persistence of elementary programming skills. *Computer Science Education*, *22*(2), 81–107.

Brown, B. A., & Ryoo, K. (2008). Teaching science as a language: A "content-first" approach to science teaching. *Journal of Research in Science Teaching*, *45*(5), 529–553.

Brown, J. E., & Doolittle, J. (2008). A cultural, linguistic, and ecological framework for response to intervention with English language learners. *Teaching Exceptional Children*, *40*(5), 66–72.

Bureau of Labor Statistics. (2015). Occupational outlook handbook, 2014–2015: Computer and information technology. Retrieved from https://www.bls.gov/ooh/computer-and-information-technology/home.htm

Burke, Q., & Kafai, Y. B. (2010, June). Programming & storytelling: Opportunities for learning about coding & composition. In *Proceedings of the 9th International Conference on Interaction Design and Children* (pp. 348–351). New York, NY: Association for Computer Machinery.

Buxton, C., Lee, O., & Santau, A. (2008). Promoting science among English language learners: Professional development for today's culturally and linguistically diverse classrooms. *Journal of Science Teacher Education*, *19*(5), 495–511.

College Board (2017). AP program participation and performance data 2017. [Data file]. Retrieved from https://research.collegeboard.org/programs/ap/data/participation/ap-2017

Cummins, J., Hu, S., Markus, P., & Montero, M. K. (2015). Identity texts and academic achievement: Connecting the dots in multilingual school contexts. *TESOL Quarterly*, *49*(3), 555–581.

diSessa, A. (2000). *Changing minds: Computers, learning and literacy*. Cambridge, MA: MIT Press.

Ed Data (2018). California public schools. [Data file]. Retrieved from https://www.ed-data.org/state/CA

Fradd, S. H., Lee, O., Sutman, F. X., & Saxton, M. K. (2001). Promoting science literacy with English language learners through instructional materials development: A case study. *Bilingual Research Journal*, *25*(4), 479–501.

Fuller, U., Riedesel, C., Thompson, E., Johnson, C. G., Ahoniemi, T., Cukierman, D., . . . Thompson, D. M. (2007). Developing a computer science-specific learning taxonomy. *ACM SIGCSE Bulletin, 39*(4), 152–170. Retrieved from http://portal.acm.org/citation.cfm?doid=1345375.1345438 doi:10.1145/1345375.1345438

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: The exploring computer science program. *ACM Inroads*, *3*(2), 47–53.

Goode, J., Margolis, J., & Chapman, G. (2014, March). Curriculum is not enough: The educational theory and research foundation of the Exploring Computer Science professional development model. In *SIGCSE '14: Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 493–498. New York, NY: Association for Computer Machinery.

Google & Gallup (2015). Images of computer science: Perceptions among students, parents and educators in the U.S. Retrieved from https://services.google.com/fh/files/misc/images-of-computer-science-report.pdf

Grover, S. (2015, April). *"Systems of Assessments" for deeper learning of computational thinking in K–12*. Paper presented at the 2015 Annual Meeting of the American Educational Research Association, Chicago, IL.

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

Halliday, M. A. K. (1973). *Explorations in the functions of language.* London, UK: Edward Arnold.

Howell, J., Tofel-Grehl, C., Fields, D. A., & Ducamp, G. J. (2016). E-textiles to teach electricity: An experiential, aesthetic, handcrafted approach to science. In C. Williams (Ed.), *Teacher pioneers: Visions from the edge of the map* (pp. 232–245). Pittsburgh, PA: ETC Press.

International Society for Technology in Education (ISTE) & Computer Science Teachers Association (CSTA), (2011). Operational definition of computational thinking for K-12 education. Retrieved from http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf

Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011, April). *Computational thinking patterns*. Paper presented at the Annual American Educational Research Association Meeting, New Orleans, LA.

Jacob, S., & Warschauer, M. (in press). Computational thinking and literacy. *Journal of Computer Science Integration*.

Jona, K., Wilensky, U., Trouille, L., Horn, M. S., Orton, K., Weintrop, D., & Beheshti, E. (2014, January). *Embedding computational thinking in science, technology, engineering, and math* (CT-STEM). Paper presented at the Future Directions in Computer Science Education Summit Meeting, Orlando, FL.

Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM*, *50*(4), 36–42.

Lee, O., & Fradd, S. H. (1998). Science for all, including students from non-English-language backgrounds. *Educational Researcher, 27*(4), 12–21.

Margolis, J. (2010). *Stuck in the shallow end: Education, race, and computing*. Boston, MA: MIT Press.

Margolis, J., Goode, J., & Binning, K. (2015). Expanding the pipeline-exploring computer science: Active learning for broadening participation in computing. *Computing Research News*, *27*(9), 16–19.

Martin, A., McAlear, F., & Scott, A. (2015). Path not found: Disparities in access to computer science courses in California high schools. Retrieved from https://eric.ed.gov/?id=ED561181

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM, 49*(8), 90–95.

McFarland, J., Hussar, B., de Brey, C., Snyder, T., Wang, X., Wilkinson-Flicker, S. . . . Hinz, S. (2017). *The condition of education 2017 (NCES 2017-144).* U.S. Department of Education. Washington, DC: National Center for Education Statistics.

McKinsey Global Institute. (2017). *Jobs lost, jobs gained: Workforce transitions in a time of automation.* Retrieved from https://www.mckinsey.com/

McNeill, K. L., & Krajcik, J. (2007). *Middle school students' use of appropriate and inappropriate evidence in writing scientific explanations*. Paper presented at Thinking with Data: The Proceedings of the 33rd Carnegie Symposium on Cognition. Mahwah, NJ: Erlbaum.

Mercier, E. M., Barron, B., & O'Connor, K. M. (2006). Images of self and others as computer users: The role of gender and experience. *Journal of Computer Assisted Learning*, *22*(5), 335–348.

Packard, B. L., & Wong, E. D. (1999). *Future images and women's career decisions in science.* Retrieved from https://eric.ed.gov/?id=ED430805

Peppler, K., & Warschauer, M. (2012). Uncovering literacies, disrupting stereotypes: Examining the (dis)abilities of a child learning to computer program and read. *International Journal of Learning and Media*, *3*(3), 15–41.

Peppler, K., Warschauer, M., & Diazgranados, A. (2010). Game critics: Exploring the role of critique in game-design literacies. *E-learning and Digital Media*, *7*(1), 35–48.

Petrich, M., Wilkinson, K., & Bevan, B. (2013). It looks like fun, but are they learning? In M. Honey & D. Kanter (Eds.), *Design, make, play* (pp. 50–70). New York, NY: Routledge.

Royal, D., & Swift, A. (2016, October 18). U.S. minority students less exposed to computer science. *Gallup*. Retrieved from http://www.gallup.com/poll/196307/minority-students-less-exposed-computer-science.aspx

Ryoo, J. J., Margolis, J., Lee, C. H., Sandoval, C. D., & Goode, J. (2013). Democratizing computer science knowledge: Transforming the face of computer science through public high school education. *Learning, Media and Technology, 38*(2), 161–181. doi:10.1080/17439884.2013.756514

Ryoo, K. (2015). Teaching science through the language of students in technology-enhanced instruction. *Journal of Science Education and Technology*, *24*(1), 29–42.

Searle, K. A., & Kafai, Y. B. (2015). *Boys' needlework: Understanding gendered and indigenous perspectives on computing and crafting with electronic textiles*. Paper presented at the Proceedings of the Eleventh Annual International Conference on International Computing Education Research, Omaha, NB. Retrieved from http://dl.acm.org/citation.cfm?id=2787724

Smith, M. (2016, January 30). Computer science for all. [Web log comment]. Retrieved from https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all

Snow, M. A., & Katz, A. (2010). English language development: Foundations and implementation in kindergarten through grade five. In Eli Hinkel (Ed.), *Improving education for English learners: Research-based approaches* (California Department of Education, Vol. III), (pp. 83–148). New York, NY: Routledge.

Tai, R. H., Liu, C. Q., Maltese, A. V., & Fan, X. (2006). Planning early for careers in science. *Science, 312*(5777), 1143–1144.

Teague, J. (2002). Women in computing: What brings them to it, what keeps them in it?. *ACM SIGCSE Bulletin*, *34*(2), 147–158.

Tofel-Grehl, C., Fields, D., Searle, K., Maahs-Fladung, C., Feldon, D., Gu, G., & Sun, C. (2017). Electrifying engagement in middle school science class: Improving student interest through e-textiles. *Journal of Science Education and Technology, 26*(4), 406–417.

Tofel-Grehl, C., & Searle, K. (2017). Critical reflections on teacher conceptions of race as related to the effectiveness of science learning. *Journal of Multicultural Affairs, 2*(1), 4. Retrieved from http://scholarworks.sfasu.edu/jma/vol2/iss1/4

Wang, J., Hong, H., Ravitz, J., & Moghadam, S. H. (2016, February). Landscape of K–12 computer science education in the US: Perceptions, access, and barriers. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 645–650). Association for Computing Machinery.

Wang, M. T., & Degol, J. L. (2017). Gender gap in science, technology, engineering, and mathematics (STEM): Current knowledge, implications for practice, policy, and future directions. *Educational Psychology Review*, *29*(1), 119–140.

Weber, M. (2018, March 22). Harvard EdCast: Teaching computational literacy. Message posted to https://www.gse.harvard.edu/news/18/03/harvard-edcast-teaching-computational-literacy?utm_source=facebook&utm_medium=social&utm_campaign=EdCast&utm_term=&utm_content=

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.

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

Zwiep, S. G., & Straits, W. J. (2013). Inquiry science: The gateway to English language proficiency. *Journal of Science Teacher Education*, *24*(8), 1315–1331.

*Corresponding author: sharinj@uci.edu