BOOK REVIEW

Beyond Coding: How Children Learn Broader Values through Programming Marina Bers (2022) 232pp. $US25 paperback, MIT Press, Cambridge MA, ISBN 9780262543323

While coding is typically integrated into STEM fields (science, technology, engineering and mathematic) (Jona *et al*., 2014; Weintrop *et al*., 2016; Sengupta *et al*., 2018), *Beyond Coding* situates coding as a necessary new literacy for full participation in today's society. There is a small but growing body of work that examines the relationship between coding and literacy (Kafai and Proctor, 2022; Bers, 2019; Jacob and Warschauer, 2018; Vee, 2017). Borrowing from Vee (2017), literacy is defined in *Beyond Coding* as a practice or skill that is critical for maintaining status and economic prosperity, and that restructures the way we know the world and think about it. Just as literacy in reading and writing is necessary for modern society to function, coding is similarly highly valued in today's society and embedded in the infrastructure of our daily lives. *Beyond Coding* compares the transition from oral to written communication, and its subsequent notions of literacy, to the transformation that is taking place through the widespread use of coding – and argues that the kinds of literacy achieved through learning to code should be learned at a very young age.

The essence of the Coding as Another Language (CAL) curriculum and pedagogical approach lies in discovering the interconnectedness of coding and literacy. The literature on how children learn to read and write is used to understand better the learning processes that underlie coding. While Bers admits that much research is left to be done on how coding is learned, early functional magnetic resonance imaging (fMRI) scans taken during her time at MIT suggest that parts of the brain responsible for language are also used during programming. Siegmund *et al*. (2014) used fMRI to study the brains of 17 subjects as they analysed short source-code snippets and found that the language-processing areas of the brain were activated during coding tasks. Based on these preliminary data and Bers's roots in the constructionist tradition, literacy is used as a starting point for understanding how children learn to code.

Bers compares the symbolic nature of written communication with the symbolic nature of coding, arguing that both natural and artificial languages are 'a socially situated system of representations with communicative and expressive functions' (p.89). To explain, both natural and programming languages are symbolic; that is, they use symbols to assign meaning (Bers, 2019). Just as syntactic features of language carry meaningful representations of real-world phenomena, programming syntax consists of meaningful representations a computer can interpret to achieve desired results. Furthermore, *Beyond Coding* argues that natural and programming languages are used to communicate and foster personal expression (Bers, 2019). Students develop computational artefacts that are both personally meaningful and can also be shared with their broader communities. The book leans toward the socioemotional and collaborative connections achieved through coding rather than focusing on problem-solving for technical purposes.

The Coding as Another Language curriculum presented in the book is an excellent example of how coding and literacy can reinforce one another (Bers, 2019). The curriculum design process was influenced by research on early childhood education, rooted in cognitive research on reading and writing processes in young children. The premise of the curriculum and pedagogical approach is that coding and language are symbolic representations that can be used to communicate and express ideas that others can then interpret. The CAL curriculum considers the learning trajectories through which students progress as they engage in an integrated coding and literacy curriculum. It draws parallels between well-established research on the stages of literacy development to identify

DOI:10.13169/prometheus.39.2.0137

six stages of coding development: emergent, coding and decoding, fluency, new knowledge, multiple perspective and purposefulness.

The CAL curriculum has been designed by Bers to be implemented with either KIBO robotics kits or the Scratch Jr. programming language – both technologies she and her team developed and much used worldwide. KIBO is a developmentally appropriate robotics kit for children 4 to 7 years old, including wheels, motors, light output, a sound recorder and a variety of sensors (sound, light and distance sensors). Wooden programming blocks can be decorated with art and recyclable materials. Scratch Jr. is a free programming application for children of the same age range that runs on tablets and laptops and uses block programming to allow children to create their own imaginative stories and games through a drag-and-drop process that replaces the need for either typing or mouse use; it is a simpler version of the better-known language, Scratch, that is typically used for children in upper elementary and middle school grades.

Bers's curriculum is aligned with Common Core English Language Arts standards and the Computer Science Framework. There are four units at each grade level, from kindergarten to second grade, with 45-minute lessons that are typically taught once a week, totalling approximately 24 lessons per year. The curriculum also includes storybooks, such as *Grace Hopper: Queen of Computer Code,* to teach students about various pioneers in the field of computer science, as well as stories, such as *Stellaluna,* to teach children about themes that are relevant to coding, such as persistence. In addition, each of the units is aligned with powerful ideas in both coding and literacy. For example, the unit on debugging is compared with editing and audience awareness in the writing process.

While curriculum alignment with standards is of interest to schools, a major strength of Bers's approach and book is how she transcends narrow pragmatic concerns of (only) matching to educational standards or (solely) preparing students for future careers. Rather, Bers puts forth a sweeping and refreshing vision of playful, community-centred learning. She situates the entire coding landscape within an ethical Buberian dichotomy between the I-thou and the I-it relationships. The I-thou relationship centres connections and interactions between people as they construct computational artefacts, and the I-it relationship situates individuals against the products or material artefacts they create. The book adopts a sociocultural approach to coding by promoting the I-thou relationship in its underlying theory and its established curriculum. Children learn through interaction, which provides rich possibilities for children's cognitive, socioemotional and ethical development. For example, I-thou relationships can engage young children in universal virtues, such as curiosity, open-mindedness, perseverance, patience, optimism, honesty, fairness, generosity, gratitude and forgiveness. While the book acknowledges that these virtues may vary across communities, cultures and nations, they still exist in some form in every culture. Therefore, discussing their myriad manifestations can lead to intercultural communication and multicultural awareness. This vision is as inspiring as it is refreshing, and one wishes it would be more influential in early childhood pedagogy and curriculum.

As for the relationship between coding and literacy, Bers acknowledges the differences between them, but argues that the purpose of the curriculum is found in their shared practices. In both coding and literacy frameworks, Bers focuses on the creation of projects, the creative design process, the need for revision and the sharing of the final projects 'as a way to express our individuality, interests, passions, and identities' (p.3). This last practice is particularly relevant today as generative artificial intelligence programs, such as chat ChatGPT, begin to automate both coding and writing processes. It is feared that such programs will replace writing in the same way it was once feared that calculators and symbolic math programs would replace mathematics and calculus. The humanistic elements that *Beyond Coding* weaves into the CAL theory, curriculum and pedagogical approach give purpose to learning to code and write that go beyond mastering discrete skills to encompass more humanistic practices, such as the expression of individualistic interests, passions and identities, which have value in themselves that transcend the mere functionality of algorithms.

Computers and Constructionism

Bers's approach to both the technologies and curriculum that she has developed, and that she articulates in the book, is that of 'constructionism', which arose from the works of Jean Piaget and later his student, Seymour Papert. Piaget argued against transmission models of learning in which the teacher imparts knowledge, a constructivist approach to learning in which students iteratively construct knowledge based on real-world experiences. Papert, who served as Bers's doctoral supervisor and greatly influenced her, took this constructivism one step further to argue that students construct knowledge by creating 'objects to think with' or any public entity that imparts new knowledge and that the computer is the perfect tool for such a task.

> Constructionism shares constructivism's view of learning as 'building knowledge structures' through progressive internalization of actions … It then adds the idea that this happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe (Papert, 1991, p.1).

In the mid-1960s, Seymour Papert and Marvin Minsky co-founded MIT's Artificial Intelligence Lab and created their first educational programming language to test their theory of constructionism, Logo. While there was great excitement around this work, there was little evidence that Papert's constructionist use of the Logo programming was effective for promoting learning, according to several studies published in the 1980s (e.g., Pea and Kurland, 1984; Clements, 1985; Clements and Meredith, 1993). Since then, Papert's constructionist approach has served as a motivating theory behind many educational uses of computers, some of which have been successful, such as the state laptop programme in Maine (see discussion in Warschauer, 2006), and others of which have not, such as the One Laptop per Child Program, promoted by the MIT Media Lab and implemented around the world (see Warschauer and Ames, 2010). In examining this latter programme, Ames (2019) suggests that constructionism was spawned in part by MIT's hacker culture and the sociological ideologies of the predominantly male, upper-middle-class members of this culture, who fondly recall their own creative, playful learning with technology, and fail to appreciate how much it was shaped by the broader privileges and support they had, including from the family members and friends who could support their entry into computing.

Bers – a female immigrant from South America breaking into the US computer science field – has a very different background and experience. Nevertheless, given her deep constructionist roots, Bers grapples with developing a cohesive curriculum instead of allowing students to construct their learning environments freely. Bers argues that the curriculum is still in line with constructionist pedagogy as it (1) ensures that powerful ideas are covered in developmentally appropriate ways, (2) promotes playfulness and exploration and (3) highlights areas of socioemotional growth and character development. She further refers to the powerful ideas in coding underlying the curriculum that are 'deeply rooted in a discipline, are personally useful, inherently connected with other disciplines, and are grounded in intuitive knowledge that a child has internalised over a long period of time' (p.91).

This emphasis on intuitive knowledge is at odds with other approaches that aim to serve culturally and linguistically diverse learners in US schools, and which often emphasise the additional scaffolding and support children need to succeed. Indeed, there is a growing body of evidence that older children and adults learn computer programming best through more structured rather than purely exploratory approaches (e.g., Sengupta *et al.*, 2013). These approaches include teacher modelling (Sengupta *et al.*, 2013; Israel *et al.*, 2015); careful sequencing of learning activities (diSessa, 1993; Smith *et al.*, 1993; Hammer, 1996; Sengupta, 2011; Sengupta and Wilensky, 2011; Dickes and Sengupta, 2013); and providing worked examples (e.g., Sweller and Cooper, 1985; Atkinson *et al.*, 2000; Bunch, 2009; Guzdial and Robertson, 2010; Mulder *et al.*, 2014).

Looking specifically at upper elementary and middle school students using Scratch, our research – and that of others – suggests that students often marginalised from computer science education, including English learners, children with disabilities and low-income children, benefit from more structured approaches to computer science learning with Scratch that involves examining and modifying existing code before writing their own (e.g., Jacob *et al.*, 2020; Prado *et al.*, 2021; Salac *et al.*, 2021).

But what levels of scaffolding and structure are appropriate for the children who are the focus of Bers's work and are much younger than those referred to in the studies mentioned above. It is entirely possible that the more structured learning environments that older children and adults benefit from may be counterproductive for young children, who learn well from play. This is undoubtedly the case in specific domains, such as learning to speak a first or second language. Young children are natural language learners and do not benefit from overly didactic language instruction, such as explicit grammatical instruction.

Nevertheless, as Bers herself points out, while learning to talk is a natural process, learning to read is not – and nor is learning to code. In *Beyond Coding*, Bers enters into the 'reading wars' between proponents of structured phonics instruction and more discovery-based whole language learning. She comes down on the side of a balance between the two, which makes intuitive sense. However, she gives insufficient attention to how so-called 'balanced literacy instruction' has too often paid lip service to phonics while leaving low-income children without the tools to succeed. As one former English teacher wrote in an essay critiquing balanced literacy, 'Expecting children to independently discover the rules of written language is like expecting them to independently discover the rules of differential calculus' (Nazaryan, 2014, p.17). Would not the same, or more, apply to the rules of algorithms?

But, alas, these metaphors only go so far. While learning to code is not like learning to speak, it is also different from learning to read. Delays in reading can set children back in school, perhaps irreversibly. In contrast, little is lost if students have a positive experience coding at a young age without becoming experts, just as young children can benefit from making music or art without having to begin with piano scales or painting techniques.

Beyond that in Bers's lab, there has been scant research on the educational processes or outcomes of young children learning to code. We still know little about the types of teaching and learning experiences that foster children's greater interest in, and critical understanding of, computer science while starting to build foundational skills that will benefit them in the future – all while positively influencing their math, language, literacy and social-emotional development. By clearly and passionately describing one vision of children's learning through coding, Bers inspires all of us to think through these questions more deeply and to study how they unfold in children's lives. Her approach presents a new and attractive perspective on computing pedagogy and children's broader development in a technological society.

References

Ames, M. (2019) *The Charisma Machine: The Life, Death, and Legacy of One Laptop per Child*, MIT Press, Cambridge MA.

Atkinson, R., Derry, S., Renkl, A. and Wortham, D. (2000) 'Learning from examples: instructional principles from the worked examples research', *Review of Educational Research*, 70, 2, pp. 181–214.

Bers, M. (2019) 'Coding as another language: a pedagogical approach for teaching computer science in early childhood', *Journal of Computers in Education*, 6, 4, pp.499–528.

Bunch, J. (2009) 'Teaching tip: an approach to reducing cognitive load in the teaching of introductory database concepts', *Journal of Information Systems Education,* 20, 3, pp.269–75.

Clements, D. (1985) 'Research on Logo in education: is the turtle slow but steady, or not even in the race?', *Computers in the Schools*, 2, 2–3, pp.55–71.

Clements, D. and Meredith, J. (1993) 'Research on Logo: effects and efficacy', *Journal of Computing in Childhood Education*, 4, 4, pp.263–90.

Dickes, A. and Sengupta, P. (2013) 'Learning natural selection in 4th grade with multi-agent-based computational models', *Research in Science Education*, 43, 3, pp.921–53.

diSessa, A. (1993) 'Toward an epistemology of physics', *Cognition and Instruction,* 10, 2/3, pp.105–225.

Guzdial, M. and Robertson, J. (2010) 'Too much programming too soon?', *Communications of the ACM,* 53, 3, pp.10–11.

Hammer, D. (1996) 'Misconceptions or p-prims: how may alternative perspectives of cognitive structure influence instructional perceptions and intentions?', *Journal of the Learning Sciences,* 5, 2, pp.97–127.

Israel, M., Pearson, J., Tapia, T., Wherfel, Q. and Reese, G. (2015) 'Supporting all learners in school-wide computational thinking: a cross-case qualitative analysis', *Computers and Education,* 82, pp.263–79.

Jacob, S. and Warschauer, M. (2018) 'Computational thinking and literacy', *Journal of Computer Science Integration*, 1, 1, pp.1–19.

Jacob, S., Nguyen, H., Garcia, L., Richardson, D. and Warschauer, M. (2020) 'Teaching computational thinking to multilingual students through inquiry-based learning' in *Proceedings of the IEEE Annual International Conference on Research on Equity and Sustained Participation in Engineering, Computing, and Technology* (RESPECT'20) Portland OR, March.

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

Kafai, Y. and Proctor, C. (2022) 'A revaluation of computational thinking in K-12 education: moving toward computational literacies', *Educational Researcher*, 51, 2, pp.146–51.

Mulder, Y., Lazonder, A. and de Jong, T. (2014) 'Using heuristic worked examples to promote inquiry-based learning', *Learning and Instruction,* 29, pp.56–64.

Nazaryan, A. (2014) 'The fallacy of "balanced literacy"', *New York Times,* 7 July.

Papert, S. (1991) *Mindstorms*, Basic Books, New York.

Pea, R. and Kurland, D. (1984) *Logo Programming and the Development of Planning Skills,* Technical Report 16, Bank Street College of Education, New York.

Prado, Y., Jacob, S. and Warschauer, M. (2021) 'Teaching computational thinking to exceptional learners: lessons from two inclusive classrooms', *Computer Science Education*, 32, 2, pp.188–212.

Salac, J., Thomas, C., Butler, C. and Franklin, D. (2021) 'Supporting diverse learners in K-8 computational thinking with TIPP&SEE' in National Science Foundation, *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education,* March, pp.246–52.

Sengupta, P. (2011) 'Design principles for a visual programming language to integrate agent-based modeling in K-12 science' in *Proceedings of the Eighth International Conference of Complex Systems*, Quincy MA, June–July pp.1636–7.

Sengupta, P. and Wilensky, U. (2011) 'Lowering the learning threshold: multi-agent-based models and learning electricity' in Khine, M. and Saleh, I. (eds) *Dynamic Modeling: Cognitive Tool for Scientific Inquiry,* Springer, Cham, Switzerland, pp.141–71.

Sengupta, P., Kinnebrew, J., Basu, S., Biswas, G. and Clark, D. (2013) 'Integrating computational thinking with K-12 science education using agent-based computation: a theoretical framework', *Education and Information Technologies,* 18, 2, pp.351–80.

Sengupta, P., Dickes, A. and Farris, A. (2018) 'Toward a phenomenology of computational thinking in STEM education' in Khine, M. (ed) *Computational Thinking in STEM Discipline: Foundations and Research Highlights*, Springer, Cham, Switzerland, pp.49–72.

Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Saake, G. and Brechmann, A. (2014) 'Understanding source code with functional magnetic resonance imaging' in *Proceedings of the 36th International Conference on Software Engineering,* Hyderabad, May, pp.378–89.

Smith, J., diSessa, A. and Roschelle, J. (1993) 'Misconceptions reconceived: a constructivist analysis of knowledge in transition', *Journal of the Learning Sciences,* 3, 2, pp.115–63.

Sweller, J. and Cooper, G. (1985) 'The use of worked examples as a substitute for problem solving in learning algebra', *Cognition and Instruction*, 2, pp.59–89.

Vee, A. (2017) *Coding Literacy: How Computer Programming is Changing Writing*, MIT Press, Cambridge MA.

Warschauer, M. (2006) *Laptops and Literacy: Learning in the Wireless Classroom,* Teachers College Press, New York.

Warschauer, M. and Ames, M. (2010) 'Can one laptop per child save the world's poor?', *Journal of International Affairs,* 64, 1, pp.33–51.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. and Wilensky, U. (2016) 'Defining computational thinking for mathematics and science classrooms', *Journal of Science Education and Technology,* 25, 1, pp.127–47.

Sharin Rawhiya Jacob and Mark Warschauer
Digital Learning Lab
University of California, Irvine
sharinj@uci.edu, markw@uci.edu