2

COMPUTATIONAL THINKING TODAY

Shuchi Grover

Introduction: From Academia to Mainstream

Daily news reflects the spirit of a time. Words and phrases appearing in mainstream media are an indicator of the prevailing ideas of an age. Google News, which indexes daily mainstream news articles from all over the world could therefore be considered a reliable indicator of the zeitgeist – in a way that neither Google Search nor Google Scholar is. The appearance of the phrase "computational thinking" in Google News articles in the January to August period of 2020 has grown by more than a factor of 10 from the same time window in 2013 – from the meager 35 or so articles (spread over 3 pages of Google News results) in 2013 to about 260 over 26 pages of results in 2020. The only logical conclusion is that the idea of computational thinking, or CT, is increasingly becoming a part of mainstream consciousness.

The choice of 2013 as a comparison to today is only somewhat arbitrary. By 2013, there was a healthy awareness of CT in the computer science (CS) education research world with about 100 pages of results on Google Scholar displaying roughly 1,000 articles (compared to the 35 articles in mainstream publishing and news sites). It was in 2013 that Code.org was launched, and with it came a significant fillip to computing education at the national policy level in the US as well as globally. It was the year Simon Peyton-Jones' keynote at SIGPLAN talked about *Computer Science as a School Subject*. And of course, it was also the year that Grover and Pea's (2013) "state of the field" paper on CT appeared in the January issue of Educational Researcher, the prestigious flagship journal of the American Educational Research Association – it was one of the first papers to interpret Wing (2006)'s definition of CT and describe it to the broader educational research community as a composite skill with component

DOI: 10.4324/9781003102991-2

elements largely drawn from CS. It was also the year an EdSurge article, *Learning to Code Is Not Enough* (Grover, 2013), made the compelling argument that while programming was indeed a necessary skill for the new age, CS learning focused on a coding language is not enough unless we also consciously focus on teaching broader CS ideas and deeper CT skills.

The story of CT's growth in mainstream consciousness is not just one of numbers though. Not only has the number of news stories (on Google News) grown more than 11-fold in these last seven years, the diversity of news stories bears testament to the spread of CT in many more realms of human experience. Where once the provenance of CT stories was largely school education (or K-12 schooling, as we call it here in the United States), today the sources of new stories include industry (especially how companies are funding the development of CT skills through partnerships with academic entities), current events (such as the UK's A-level and GCSE exam-grade debacle in the midst of the Covid-19 pandemic), best-selling business book titles (such as John Maeda's "How to Speak Machine: Computational Thinking for the Rest of Usw"), and drama (in the form of "Algorithmic Theater"), even as stories from primary and secondary school education still abound. Perhaps the most telling news stories of CT's consistent spread in 2020 are ones from higher education. Institutions around the world are creating new courses that call out CT as a key ingredient and teach applications of computing and computational techniques (such as MIT's *Introduction to Computational thinking*, Northwestern University's MBAi program, and Purdue's new agriculture informatics course) or using CT in research in novel spaces (for example, the use of CT in gerrymandering research at Boston University), or viewing CT as a key means to inspire women in Science, Technology, Mathematics and Engineering (STEM) (as described by the first female director of MIT CSAIL, Daniella Rus).

A few things are amply evident in these news stories. CT has currency today well beyond K-12 school education where it enjoys healthy support and has, in fact, helped steer the adoption of computing and programming in primary and secondary classrooms (Curzon et al., 2020). As these news stories attest, both in academic and non-academic settings, people describe CT as a vehicle for engaging with computing and participating in 21st-century careers and culture. CT is seen as a 21st-century literacy, and as a 21st-century skill alongside critical thinking, creativity, collaboration, and communication (as described by Grover, 2018). CT is listed among skill sets that separately call out programming and digital literacy suggesting there is a distinction drawn between other digital skills and CT. CT is seen to be synonymous with computational problem-solving, but it is also seen as a means to computational creation and participation (Kafai, 2016) and computational action (Tissenbaum, Sheldon, & Abelson, 2019). Some have argued that the explosion of interest in CT is ***because*** it is seen as a problem-solving skillset for everyone, not just

programmers or even computer scientists to possess (Curzon et al., 2020). As we design for pedagogies for teaching computing and CT, we must heed the fast-changing computing context – the increasingly pernicious possibilities of biases in algorithms, for example – and use CT as a vehicle for "critical computing" (Ko et al., 2020; Washington, 2020). *There is also a need to teach computing with a focus on CT so that the skills developed are conceptual and creative, and can survive the impending onslaught of artificial intelligence and obsolescence of skills such as vanilla programming.*

This chapter traces a brief history of CT in its various flavors and avatars from the early days of computing to the present day. It describes what CT has come to mean today, addresses key approaches to engaging learners in CT experiences, and discusses the vast space of CT integration into various learning contexts and domains including attempts to bring coherence to the discourse of CT integration. It ends with a reflection on open questions and areas for further inquiry.

From Knuth to Wing: Computational Thinking as Problem-Solving and Disciplinary Thinking from/of CS

For many in education, Jeannette Wing's 2006 article provided the impetus to attend to CT or "thinking like a computer scientist", as an essential problem-solving skill for the new generation of learners. Wing's article resonated with the broader scientific community also because it came on the heels of enormous activity in the field of science in the early 2000s, where the use of computational modeling tools and algorithms for working with large datasets was tangibly transforming the very fabric of science – a new reality that was compellingly described in the influential "Towards 2020 Science" report (Emmott & Rison, 2005).

Although Wing was the first to make a compelling case for CT as a universal, foundational skill in this age of ubiquitous computing, the idea of "computer science thinking" has been written about sporadically since the 1960s. Problem-solving practices of CS were discussed as early as 1968 (by G.E. Forsythe). Among the earliest articles to articulate *elements of CS thinking* was Donald Knuth's Algorithms in Modern Mathematics and Computer Science (Knuth, 1981), in which he wrestled with the question, "Do most mathematicians have an essentially different thinking process from that of most computer scientists?" Interestingly, he referred to the thinking processes of computer scientists as "**computer science thinking**". Knuth listed key elements of mathematical thinking (MT), and compared and contrasted them with elements of computer science thinking (CST). CST overlapped with MT in the areas of formula manipulation, representation of reality, reduction to simpler problems, abstract reasoning, information structures, and algorithms (with formula manipulation exhibiting only mild connection with CST). He also noted that

generalization and formula manipulation involve the general idea of pattern recognition. He observed that, unlike MT, CST does not deal with infinity, and conversely some kinds of CST are not part of MT, namely, (1) dealing with complexity (since MT does not care about efficiency, economy, or cost of operations) and (2) the dynamic notion of "state" of a process. The thorny issue of variable "assignment" and its distinction with mathematical equality was the subject of another Knuth piece – "As we have remarked, mathematicians had never used such an operator before; in fact, the systematic use of assignments constitutes a distinct break between computer-science thinking and mathematical thinking" (Knuth & Pardo, 1980).

A second noteworthy article, in my view, that defines and explains the idea of CS-related skills and practices *as separate* from knowledge components is Denning's (2000) – *Computer Science: The Discipline.* In addition to discussing key content topics that comprise computing, Peter Denning outlined "Standard Concerns of the Field" and stated that "every practitioner of the discipline must be skilled in four basic areas: algorithmic thinking, representation, programming, and design". It is useful to note that Denning described programming as taking "algorithmic thinking and representations and embody them in software that will cause a machine to perform in a prescribed way. This skill includes working knowledge of different programming languages (each having its own strengths and limitations), program development tools (which aid testing, debugging, modularity, and compatibility)" and design as including "many practical considerations such as engineering tradeoffs, integrating available components, meeting time and cost constraints, and meeting safety and reliability requirements".

There is substantial overlap between Denning's areas of CS "practice" (as well as Knuth's elements of CST) and the articulation of CT by K-12 CS educators and researchers following Wing (2006), including (among others), Grover and Pea (2013), the Computing At School group in the UK (Csizmadia et al., 2015), the K-12 National CS Framework in the US (k12cs.org), and Bocconi et al. (2016). They converge on CT as comprising **algorithmic thinking**, **representation** (and **abstraction**), **generalization** and **pattern recognition, decomposition, creating programs**, **debugging and systematic error processing**, and **evaluation**. There are minor distinctions in that debugging, modularity, and evaluation are sometimes called out separately today as opposed to being encapsulated in other practices related to programming and design. However, the idea of a set of CS-inspired problem-solving skills of which programming is but one (in addition to algorithmic thinking, representation, and design), is common to all these formulations. Lastly, even though Wing (2006) made it amply clear, the National Academies Workshops on CT (NRC, 2010) helped underscore the view that CT is more than programming. The separation of programming from algorithmic thinking, representation, and design by Denning (2000) serves to provide an answer to the oft-asked question, "*Is CT any different from programming?*"

Approaches to Engaging Learners in CT

Today CT is recognized as a set of skills and problem-solving strategies that have their roots in CS, as well as mathematics, design, and engineering (Shute et al., 2017). CT has, in fact, been instrumental in the adoption of teaching CS and programming in K-12 education in many countries around the globe that have called out CT as a key part of learning computing and programming (Bocconi et al., 2016; Hubwieser et al., 2015), with India most recently declaring it as part of their 2020 National Education Plan. CT pedagogy within and across nations usually includes both coding as well as non-coding activities. CT is also taught and applied in the context of learning other subjects (discussed in section "Integrating CT in Non-CS Classrooms"), an approach that is seen by many as promising and productive to ensure more sustained engagement with CT. See Chapter 3 in this volume for a discussion on the progression of CT learning experiences.

> It is important to remember that much like problem solving, CT is a skill or competency that develops over time (as with all skills) and one-off experiences are insufficient. Also, CT does not really have a "content" component that needs to be taught. Skills are developed ***in context***, through educational experiences that require learners to recognize a need for them and employ them as appropriate. As such, CT can be developed in several learning contexts, in both CS and non-CS classrooms. It can be taught through various pedagogical approaches.

CT in CS through Learning Coding

One of the goals of CS learning experiences especially when teaching programming should be to engage in CT, and help learners recognize and use CT skills to aid the problem-solving process. Many CS curricula underscore the need to teach CT as part of programming which is by far the most popular vehicle to develop CT skills. *If taught well,* programming helps learners engage in the elements of CT. However, recent research in block-based programming environments has shown that learners can create programs without a robust understanding of programming concepts (Grover, Jackiw, & Lundh, 2019; Salac & Franklin 2020) or purposefully engaging in CT. Grover, Pea, and Cooper (2014) argue that an overemphasis on programming environments and syntax can result in shallow programming experiences without deeper engagement in CT practices; infusing CT in CS and learning of programming requires thought and deliberate design to ensure that learners engage in ways that develop computational habits of mind. For example, activities and learning experiences can be designed such that learners

are required to consciously engage in algorithmic thinking through planning code before programming – an exercise that also helps learners appreciate the layers of abstraction in the context of programming (Armoni, 2013; Bagge & Grover, 2020; Waite et al., 2018), recognize and appreciate elements of algorithms (sequence, repetition, and selection), problem decomposition, debugging, and pattern recognition across problems. When the curriculum ensures well-designed engagement with CT, learners understand programming concepts more deeply, are better able to design and create better artifacts and are able to demonstrate transfer to new programming and problem-solving contexts (Grover, 2021b; Pea, 2015; Hutchins et al., 2020a, 2020b).

Sometimes curriculum designers choose to foreground specific CT practices. For example, Fields and Kafai (2020) center debugging in the learning of CT and programming. Futschek and Moschitz (2011) focus on building algorithmic thinking with unplugged activities with cards before they move to Scratch. Waite and colleagues focus on teaching abstraction through conscious engagement with design and planning in programming (Waite et al., 2018). Dominguez and colleagues focus on algorithmic thinking, abstraction, and decomposition as entry points for CT in preschoolers' activities (Grover, Dominguez, Kamdar, Vahey, Moorthy, Rafanan & Gracely, 2019; Grover, Dominguez, Kamdar, Leones, Vahey, & Gracely, 2021). Recent research has also proposed a data-centric approach to developing CT skills (Grover, 2020; Gu, Heller, Li, Ren, Fisler, & Krishnamurthi, 2020) and in introductory computing more broadly. It is worth noting that while data abstraction, data organization, and data representation are indeed called out as essential aspects of CT (Grover & Pea, 2013; ISTE/CSTA, 2011), common CT practice in classrooms has thus far tended to focus on algorithmic thinking. This proclivity for data-centric approaches to CT however may see a change with the growing attention to data practices and data science that are now also seen as entry points for AI in K-12 classrooms (Touretzky et al., 2019).

Engaging in CT through Unplugged and Non-Programming Activities

There are approaches besides programming that can help learners build CT and apply CT skills. Unplugged or non-programming activities – that do not use any computing devices – are popular, especially with younger learners who may not be familiar with coding. *CS Unplugged* (Bell et al., 2002; Bell & Vahrenhold, 2018) has been a popular curriculum to teach CS concepts through engaging, offline activities. *CS Unplugged* activities also clarify for teachers what CT skills are embedded in each of their activities. Although some efforts have introduced children as young as six to programming through software environments like Scratch Jr and Kodable, and tangible computing tools (e.g., Bers et al., 2014; Papadakis, Kalogiannakis, & Zaranis, 2016; Weintrop & Grover, 2020), many

believe that it is easier to engage younger learners as well as non-specialist primary grades teachers in ideas of problem decomposition, pattern recognition, and algorithmic thinking through unplugged activities (Huang & Looi, 2020). Curzon and McOwan (2017) believe there is a great deal to be gained by looking for a simplified CT progression for younger learners that builds on everyday ideas that are easily relatable, for example, stories, puzzles, magic tricks, board games, and even song and dance. However, the idea of algorithmic precision is one that needs to be reinforced in unplugged settings; this can be demonstrated by having different students executing the same algorithm and examining whether the same outcome is achieved every time. There is value in picking examples that are less susceptible to fall prey to a lack of precision in instructions (such as the (in)famous peanut butter and jelly sandwich activity). Such precise activities may include dance routines or rules for how to play simple games (and pick a winner) or navigating a path on a grid. Curzon et al. (2020) suggest that as students progress through levels of schooling, precision, completeness, cohesion, and elegance will improve. Kamdar et al. (2020) found unplugged activities to be useful in helping preschool learners (ages 3–5), including some at the preliterate state, by engaging them – both at home and in school – in algorithmic thinking (often using navigational activities that also build crucial spatial thinking skills), task decomposition, and abstraction (through sorting and labeling by grouping multiple objects according to common themes or characteristics).

Additionally, helping students experience CT and programming concepts in contexts outside of coding through engagement in unplugged activities and contexts that are familiar and tractable before they experience them in coding, has been shown to be beneficial (Hermans & Aivaloglou, 2017). There are examples of digital (non-programming) microworlds and online games designed specifically to target one or more aspects of CT skills. For example, in their research, Grover and colleagues successfully demonstrated the use of designed non-digital and digital activities to engage learners in the ideas of variable and variation, multiple forms of operators and expressions (arithmetic, relational, string, and Boolean), and iteration – ideas that are foundational to CT and coding (Grover, Jackiw, & Lundh, 2019). Their digital activities drew inspiration from the research of the benefits accruing from conceptual exploration with dynamic representations in mathematics education such as the Geometer Sketchpad (Jackiw, 2004).

In a similar vein, there are several digital game-based environments that help learners engage in elements of CT. Although research on the results of game-based experiences for learning is mixed, syntheses of the game-based learning literature have found that games can indeed yield positive learning outcomes across several learning contexts. Zoombinis (Rowe et al., 2017) is an example of a game that engages learners in logical and algorithmic thinking. Taylor et al. (2019) developed a toolkit to fuse block-based programming with

game-based learning to promote CT. Given the social nature of games, game-based learning is often designed and engineered to also help promote equity and collaboration (Boyer et al., 2015).

Two key ideas are of utmost importance in the context of unplugged and non-programming approaches to CT engagement. First, a move from unplugged to coding activities at some appropriate juncture or grade level is absolutely necessary in order for learners to appreciate the power of computation and automation, as well as the specific abstractions (data and procedural) that play a role in computational solutions. Thinking in layers of abstraction (Armoni, 2013) helps learners view computational solutions in different forms and also appreciate the need for data abstractions that are suited to the task and tool at hand. Programming also affords learners the opportunities to create and share their own computational creation thus allowing for computational participation (Kafai, 2016) and computational action (Tissenbaum, Sheldon, & Abelson, 2019). Such engagement is seen as beneficial to encourage learner agency and participating in computing for community and self. Second, it is critical that learning designs help bridge to and from the earlier non-programming experiences and coding experiences (Curzon & Grover, 2020; Hoyles, 2020) in order to aid learners in the transfer of learning and making the conceptual links between experiences.

Unplugged activities have been shown to be beneficial especially for non-specialist teachers in preparing to integrate CT (Huang & Looi, 2020; Yadav et al. 2017), and as a result, unplugged activities are often included as part of teacher professional development (PD) on CT integration (Araujo, Floyd, and Gadanidis, 2019). Ketelhut et al. (2020) found that teachers want practical examples of activities they could use to incorporate CT into elementary science that were either unplugged or used tools that were already available to them. Section "Open Questions and Need for Further Empirical Inquiry" addresses open research questions related to unplugged approaches to developing CT skills.

Integrating CT in Non-CS Classrooms

> It is in all the contexts outside of CS classrooms that computational thinking (CT) truly shines with its generativity. From music, maths, social studies, history, language arts and throughout the sciences and engineering, curricular ideas can come alive with CT.
>
> *(Grover & Pea, 2018, p. 32)*

Many argue that CT in its avatar as a generative problem-solving skill is truly visible in contexts outside of CS. This practice of CT serves the realm of *Computational X* – the integration of CT to enable/enrich learning and application in a whole host of other disciplines, mainly through the vehicle of programming, and automating data and procedural abstractions and models in other disciplines.

Papert's use of the phrase *Computational Thinking* in both Mindstorms (Papert, 1980) and Situating Constructionism (Papert & Harel, 1991) was made in passing and never explained. However, one can surmise from the context and his writings that he meant "thinking *with* a computer" or using the computer *"as a tool to think with"*, to engage with topics/concepts in other disciplines. In fact, his seminal book and the theses leading up to it were aimed largely at developing a "mathematical way of thinking (MWOT)" and how children could use programming to develop MWOT (also see Papert (1972)). However, Papert's view of CT was centered on algorithmic or procedural thinking. It did not delve deeply into how computing could help children build models involving data, data organization, or data representations; or with abstractions and generalizations – although these were well within the purview of computational exploration of mathematical ideas.

In contrast, Andrea diSessa, whose context for computational learning and computational literacy was science (mostly physics), articulated the value of using computing to help students' reasoning, modeling, and abstraction abilities. Representations, and the computer as the "protean mother of metarepresentational systems" (diSessa, 2001, p. 183), were at the heart of diSessa's work to help learners engage in multiple representations in order to gain a fuller and more flexible understanding of the scientific concept at hand. More importantly, diSessa's work advocates for not fixating on programming and the computer. In *Changing Minds*, diSessa (2001) emphasized both the "cognitive" and "social" aspects of computational literacy. In more recent writing, Li, diSessa, and colleagues argue for making CT more about thinking than computing because "the reconceptualization of CT [beyond Wing's articulation], as a model of thinking, makes its integration in all education a possibility" (Li et al., 2020).

In the same vein as diSessa, Weintrop, Beheshti, Horn, Orton, Jona, Trouille, and Wilensky (2016) operationalized the computational modeling view of CT based on two decades of research on computer-based systems modeling led by Uri Wilensky and his research group. They presented a taxonomy that defined "computational thinking in mathematics and science" as including data practices, modeling and simulation practices, computational problem-solving practices (including programming), and systems thinking practices. Several projects, especially those in science contexts that involve modeling and simulation and/or a focus on data practices benefit from this taxonomy. Netlogo, created by Uri Wilensky, (along with its variants

such as StarLogo) was among the first computing environments that embodied a pedagogy aimed at engaging in the disciplinary STEM practice of developing computational models and simulations, and offered learners tools to think in terms of computational models and engage in the practices outlined by the taxonomy. Curricular interventions such as Project GUTS leveraged these tools to include modeling and simulation in middle school science and social studies program.

Denning's recent writings on CT highlight that the sciences are where CT was born, and that computational science has been the shining example and the best argument for CT as a valuable way of bringing CS and computers into non-CS disciplines (Also See Chapter 1 for a discussion on historical roots of CT). He channels Aho (2012) to argue that computational modeling – a practice integral to computational science – be called out as a key part of CT. Although the value of computational modeling in professional fields from economics to finance to medicine is beyond question, many in K-12 settings would likely take issue with Denning's formulation. It not only ignores the validity of unplugged approaches to CT, but it also sidesteps a view of CT as a disciplinary thinking skill for deeper engagement in CS classrooms. In contrast, Sengupta et al. (2020) suggest that there is value in *framing* coding as a modeling in STEM classrooms and they reasonably argue that embodied modeling and non-computational materials should be viewed as representational and cognitive amplifications of computational code. The following section showcases the plurality of approaches for integrating CT.

Approaches for CT Integration in Other Disciplines

Not surprisingly, science and mathematics classrooms have provided the most prolific contexts for CT integration. In addition to Project Growing Up Thinking Scientifically, or GUTS, (projectguts.com/) and CT-STEM (ct-stem.northwestern.edu/), several new efforts have pursued the goal of integrating CT and computing in STEM domains for secondary students (though not all draw on Weintrop et al., 2016). These include Bootstrapworld (www.bootstrapworld.org/), the STEM Coding project (u.osu.edu/stemcoding/), C2STEM (c2stem.org), EcoMOD (Jeon et al., 2020), among others. The C2STEM curriculum and environment uses domain-specific blocks specially designed in Snap! programming, along with guidance from both Grover and Pea (2013) and Weintrop et al. (2016) to design for and measure synergistic learning of CT and science (Hutchins et al., 2020a). Other efforts, including Dominguez, Grover and Vahey (2020), Sengupta, Dickes, and Farris (2020), Lavigne, Orr, and Wolsky, (2018), Benton, Hoyles, Kalas, and Noss (2016), (2018), Moore, Brophy, Tank, Lopez, Johnston, Hynes, and Gajdzik (2020), Waterman, Goldsmith, and Pasquale (2020), Yadav, Larimore, Rich, and Schwarz (2019), have attempted to integrate coding and CT into science and math in primary and pre-primary school classrooms.

Not all STEM integration efforts rely on modeling and simulation to guide integration into STEM learning. Wendell et al. (2020) designed an interdisciplinary school curriculum in which students design a search and rescue robot (using physical computing) that can dig through the rubble. They accomplish this through exploring animal structure-function relationships using CT practices like decomposition and abstraction and then combining those findings with engineering design practices to build robots using Hummingbird controllers. Hadad et al. (2020) also used a physical computing approach to CT integration in physics. Other examples favor a data-driven approach to STEM and CT integration (e.g., Krishnamurthi et al., 2020; Wilkerson & Fenwick, 2017). Yet another approach taken by some researchers is to leverage the overlap between systems thinking and CT, to integrate these thinking skills in STEM learning (Damelin et al., 2017; Puttick & Tucker-Raymond, 2018).

There are compelling examples of CT integration in non-STEM subjects, such as social studies, as well. Cannell, Tofel-Grehl, and Searle (2020) describe how upper elementary students worked with first-hand data sources, engaged in data analysis and transformations, used visualization tools and physical computing with Circuit Playground Express to explain population shifts & explore weighty and difficult-to-discuss topics such as the great migration of Blacks from the American south during the Jim Crow era. Similarly, the Bootstrap: Data Science curriculum centers data for answering historical questions in ways students could not otherwise, for example, *what caused the fall of the Mayan civilization?* Or *how is the history of food production in the US tied to immigrants' backgrounds?* Through choosing appropriate narratives and stories, the curriculum helps set the context for data collection, analysis, and visualization using spreadsheets.

Recently Caplan et al. (2020) suggested moving beyond a mere definition of what students needed to learn in integration settings (as defined by Weintrop et al., 2016). Their work extends our understanding of what discipline-based CT looks like for "competent outsiders" belonging to other disciplines, understanding ways in which students make sense of CT in integration settings, and what a learning progression might look like.

CT integration is often driven by the need to marry data, representations, and algorithms with concepts from the host domain. Programming these representations and algorithms affords engagement with – and sense-making of – those concepts, however, engaging in designing those data organizations and data representations can be done outside of coding. Gu et al. (2020) demonstrate the value of engaging in data organization in helping learners wrestle with data reasoning aspects of CT. Similarly, Grover (2018) presents an example that draws on the computational literary analysis of Shakespeare's Hamlet by Stanford professor Franco Moretti, to demonstrate how representations from social network analysis can be combined with literary analysis through the use of graphs (closed networks with nodes and edges) for CT integration

in Language Arts. The exercise of converting a story into a network could be attempted computationally, but it is a valuable cognitive exercise that could also be done manually. Extending Moretti's example, Grover demonstrates how students can subsequently transform graphs into mathematical abstractions (adjacency matrices) that can be coded as two-dimensional arrays or data structures in a programming environment. Thus, an unplugged activity could be extended through a plugged one; however, both versions can afford learners novel experiences into modeling and analyses of literary narratives and the relationships of the characters.

The Special Issue of the Journal of Science Education and Technology Computational Thinking from a Disciplinary Perspective (Lee et al., 2020) as well as recent symposia on STEM integration with CT (Grover et al., 2020; Grover, Fisler, Lee, & Yadav, 2020) share many other efforts and exemplary works in STEM and CT integration.

Toward Coherence to Support CT Integration into School Subjects

Recently, scholars have attempted to provide more coherence around the various definitions and approaches to teaching CT to emerge with frameworks that can help teachers better understand how to integrate CT in programming and across other subjects. Here we describe Malyn-Smith et al. (2018)'s articulation of CT from a disciplinary perspective, Grover (2020)'s CT Integration framework, and PRADA (Dong et al., 2019) – an acronym for Pattern Recognition, Abstraction, Decomposition, and Algorithms. Additionally, see Chapter 3 in this volume by Azeka and Yadav on a CT integration model and Chapter 5 by Tannert, Lorentzen, and Berthelsen on CT in K-12 as an independent subject vs. integrating it in school subjects.

Malyn-Smith et al. (2018) attempted to examine and articulate what CT looks like from within a discipline in an effort to move away from the CS-centric view of CT and better guide integration efforts. They examined the work of 'Computational X' professionals and examined descriptions of curricular units that integrated CT provided by teachers and researchers (as part of a 2-part NSF-funded workshop series). They concluded that individual CT components were rarely deployed in discrete capacity in professional work. Instead, "CT integration practices" came into play that included: understanding complex systems; innovating with computational representations; designing solutions that leverage computational power and resources; engaging in collective sense-making around data; and (making predictions) and understanding potential consequences of actions. In my view, these practices are perhaps the best articulation of the "why" or benefits of integrating CT and computing into other subjects.

More recently, Grover (2020) further built on Malyn-Smith et al. (2018) and drew on several CT integration projects to synthesize the interconnected

components of integration to articulate *CTIntegration* (Grover, 2021a), a comprehensive framework for integrating computing and CT in disciplinary learning in school. The framework draws on Shulman (1987)'s seminal ideas on teacher's pedagogical content knowledge (PCK) related to a domain and is inspired by Mishra and Koehler (2006)'s Technological Pedagogical Content Knowledge (TPCK) framework to integrate technology into school learning. It provides guidance on how researchers and teachers can attend to three critical intersections of the host domain or discipline (content, concepts, and practices), CT (concepts and practices), and pedagogy (see Figure 2.1). By foregrounding domain PCK, CT PCK (including examination of appropriate computing tools), and the intersection of discipline and CT (with guidance on how to identify productive points of intersection), the framework shows that CT integration can serve many purposes in teaching various subjects, including the creation of computational artifacts, sense-making through data and modeling, as well as the critical examination of phenomena. The framework is intended to support not only curriculum and assessment design for CT integration, but also research and teacher preparation efforts. Grover (2021a) also examines and evaluates existing integration efforts through the lens of the framework, and highlights that the success of curricula that do a good job of integrating CT depends on how well the three intersections are designed. Figure 2.1(B) presents a view of the framework that acknowledges the growing emphasis on, and opportunities provided by data as the linchpin for integration. Using the framework, Grover underscores the need to attend to issues of tool choice for integration including task-specific versus general-purpose languages, issues of cognitive load, and designing for bridging and transfer, and the importance of framing in CT integration.

Dong et al. (2019) proposed PRADA "as a practical and understandable way of introducing the core ideas of CT to non-computing teachers in order to support them in infusing CT into their curricula". Their goal was to create a definition of CT that could be used for integration into existing curricula and generalizable to any discipline and be easily understood by primary and secondary teachers (the majority of whom have limited CS knowledge). They used definitions of CT that were popular in literature and practice to find common ground (Figure 2.2). The PRADA acronym stands for Pattern Recognition (observing and identifying patterns, trends, and regularities in data, processes, or problems), Abstraction (identifying the general principles and properties that are important and relevant to the problem), Decomposition (breaking down data, processes, or problems into meaningful smaller, manageable parts), and Algorithms developing step by step instructions for solving [a problem] and similar problems. They argue that data practices and modeling and simulation are also captured within the elements of the PRADA framework.

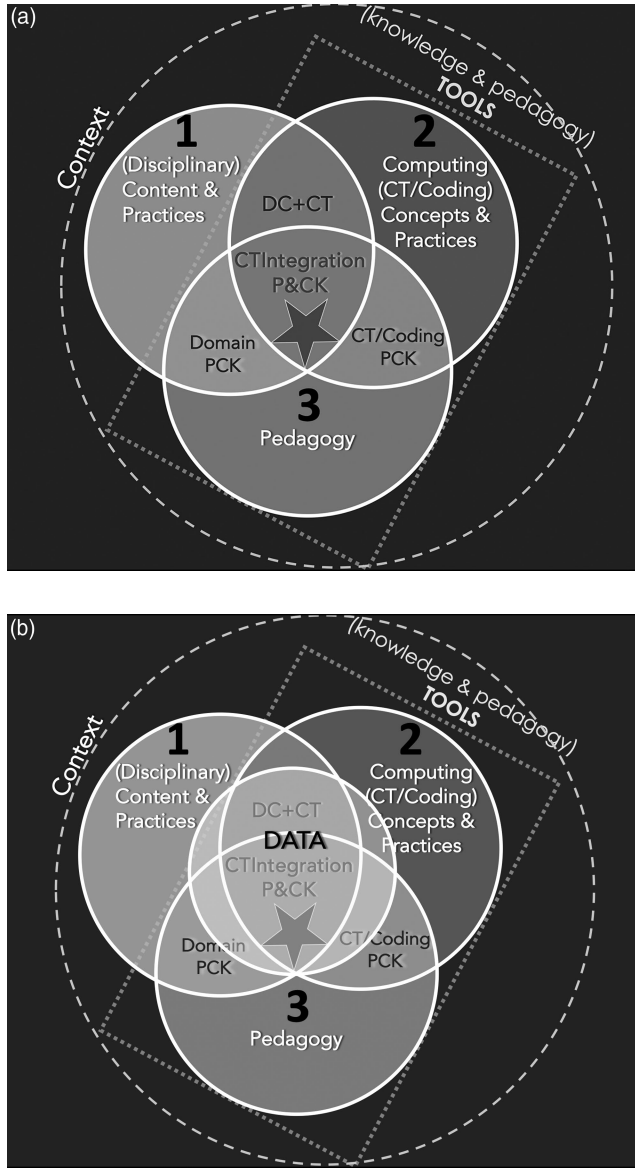

FIGURE 2.1 (A and B) CTIntegration conceptual framework for aiding design, evaluation, and research on CT Integration Grover (2021a).

Shuchi Grover.

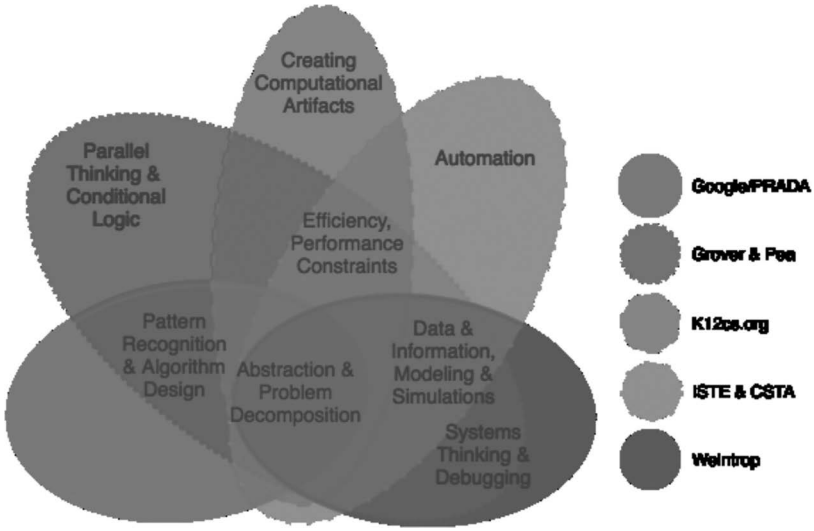

FIGURE 2.2 PRADA framework derived from extant definitions for CT integration.

Dong et al. (2019).

Open Questions and Need for Further Empirical Inquiry

This section outlines open questions that the education community continues to wrestle with even as work on CT and its implementation continues to expand. Despite the enormous progress for CT learning in various forms and various contexts in K–12 school education, as well as synthesis efforts such as those described above that help in instructional design, teaching, and research, there remain open questions that can and must be answered through empirical inquiry.

Despite the growing popularity of unplugged approaches to teaching CT, we lack a robust understanding of the affordances of unplugged approaches to CT. Huang and Looi (2020) also point out that many unplugged activities and CT assessments are heavily oriented toward a coding view of CT (see Grover, 2017; Tang et al., 2020). Among many other open questions related to unplugged engagement with CT, they ask, *how do we assess CT in unplugged activities without using code representations?* Our work with pre-kindergarten learners forced us to think in non-coding terms, resulting in a scenario-based assessment (Grover, Dominguez, Kamdar, Leones, Vahey, & Gracely (2021)). Much work remains to also be done for the assessment of learning in contexts where CT is taught through integration in other subjects. Researchers and teachers routinely seek assessments of CT that are not tethered to a specific coding environment, and few such exist. For STEM integration projects, it appears that every context requires its own ways of measuring CT learning because when integration is done well, CT is tightly intertwined with the domain and appropriate CT tools (Grover, 2020). This is a space that continues to need more work.

Transfer of learning (or lack thereof) has often been offered (in my view) as a strawman argument by critics of CT. As explained in Grover and Pea (2018),

the learning sciences advocate that learning designs consciously attend to transfer; transfer needs to be mediated through empirically established techniques that call for, among other things, making explicit connections between the original and transfer learning contexts. My own work, although focused narrowly on algorithmic elements and coding, demonstrated transfer from block to text-based programming. In recent work, Hutchins et al. (2020a) demonstrated that students used a discrete time-step approach on a novel physics problem after an intervention involving designing computational simulations in Snap! that involved using the "delta-t" time-step variable. Hutchins et al. (2020b) demonstrated that another group of students was able to reason about Netlogo simulations after the same intervention. They also show evidence of improvement in students' modeling skills as the intervention progressed through various units, demonstrating the notion of ***dosing***, and CT skills-building over time. We need more robust studies aimed at studying the transfer of CT in interventions that are explicitly designed to mediate for transfer of these problem-solving skills. We need to better understand how to bridge problem-solving in new contexts with earlier (perhaps unplugged) experiences. We also need to understand how much dosing learners need in order to be able to apply CT skills to new (near- and far-) transfer contexts.

More work is needed in the context of integration of CT with other subjects emerges as a preferred approach to teaching CT. Although there has been work done to define learning trajectories of elements of CT (Rich et al., 2017, 2018, 2019), these are theoretical (based on prior literature) and need to be empirically validated, and may not always translate to CT integration settings. Grover (2018) proposed a theoretical progression of CT integration activities from simple to complex based on her work in Pre-K through high school integration projects, but this too needs to be fleshed out further with additional empirical inquiry. Teacher preparation, especially in integration settings in secondary grades, also needs more attention. While there is a growing body of literature on teacher training for CT integration in elementary school settings much work needs to be done for integration in secondary grades where integration is more complex and often includes coding, modeling, and simulations, (Grover, 2018, 2020).

Finally, we need to push on a few directions hitherto explored by only a few. There is an urgency to understand the role of culture and phenomenology in CS and CT pedagogy. The importance of appropriate framing of CT integration in school subjects cannot be overemphasized. It impacts learners' relationship with computing as well as subject learning through computing. For example, Pierson, Brady, and Clark (2020) framed computational models as interactive participants in a sixth grade STEM classroom and Sengupta et al. (2020) provide evidence that phenomenological approaches can be useful for framing CT in K12 STEM classrooms – for example, framing programming as designing mathematical measures of change, and framing coding as designing for authentic use. Such framing, as Sengupta et al. (2020) contend, can also help make CT learning more meaningful in classrooms with minoritized populations. Barring a few efforts (e.g., Madkins et al., 2019), the field has not engaged deeply in how culturally relevant pedagogy can be more widely employed in developing CT skills

in various contexts. Relevant to this is also the role of learners' natural language, and how it can and should be leveraged for authentic engagement with CT as demonstrated in the trans-languaging work of Vogel et al. (2019).

Ultimately, we also need to continue to examine why we teach CT and CT's relationship with the teaching of CS and coding in order to be guided on the what and how of developing this skill among learners at various ages and stages. Coding is no doubt a new and valuable skill for K-12 classrooms. It is worth bearing in mind that we teach computing to develop informed citizens in a fast-changing and increasingly pernicious world driven by computing where biases are being replicated in algorithms, and the loss of privacy or reputation may be but a few clicks away. We need to teach computational thinking and problem-solving in ways that transcend programming language (which may not be used in professional settings today, or may be obsolete tomorrow if in use today) and can survive the impending onslaught of artificial intelligence systems with the capability to generate much of the code written by humans today. What learners must take forward are habits of mind, analytical thinking and problem-solving strategies, and sensibilities of design and data gleaned in CS and non-CS classrooms.

Closing Remarks

As this chapter has described, CT is a thinking skill that can be used for problem solving in various domains, for the creation of artifacts and creative expression – for oneself or one's broader communities, and for sense-making (increasingly with data) in various contexts. As such, it has a place in all school subjects (including CS). It can be learned in both CS and non-CS classrooms, and it bears the potential to enrich learning in all settings. Teaching CT ensures deeper learning of CS disciplinary practices and skills (programming among others) in CS classrooms and leveraging CS problem-solving approaches (representations, programming, and computational modeling perhaps being the most valuable) in non-CS settings. Ideally, learners should experience CT often and in diverse settings, and in increasingly sophisticated ways, over the course of their K-12 journey. In the spirit of pedagogical plurality and epistemological pluralism, experiencing and applying CT in different ways (in coding contexts, unplugged activities, as well as games and digital interactives), and for different purposes (for sense-making, innovation, and problem-solving, for creative expression, and for social participation and social justice through computing) – helps diverse learners at different ages and stages engage meaningfully with CT. As with any competency, skills are built over time. Problem-solving skills too are honed through multiple applications, in multiple forms, and multiple contexts. And so it is with CT. The idea of dosing in educational interventions suggests that the more learners engage in CT in various settings, the deeper they understand abstraction and representation, and algorithms (through coding and non-coding activities). There remain questions about teaching and learning CT that must be answered through robust empirical inquiry. Judging by the revelations in section "Introduction: From Academia to Mainstream", however, the idea of CT is thriving and growing in acceptance. CT is, indeed, here to stay.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Araujo, R. C., Floyd, L., & Gadanidis, G. (2019). Teacher candidates' key understandings about computational thinking in mathematics and science education. *Journal of Computers in Mathematics and Science Teaching*, *38*(3), 205–229.

Armoni, M. (2013). On teaching abstraction in computer science to novices. *Journal of Computers in Mathematics and Science Teaching, 32*(3), 265–284.

Bagge, P., & Grover, S. (2020). Before You Program, Plan! In Shuchi Grover (Ed.), Computer science in K-12: An A-to-Z handbook on teaching programming (pp. 99–112). Ed"nity, Palo Alto, CA.

Bell, T., & Vahrenhold, J. (2018). CS unplugged—How is it used, and does it work?. In Dennis Komm, and Walter Unger (Eds.), *Adventures between lower bounds and higher altitudes* (pp. 497–521). Springer, Cham, Switzerland.

Bell, T., Witten, I.H., Fellows, M., McKenzie, J., & Adams, R. (2002). Computer Science Unplugged: An Enrichment and Extension Programme for Primary-Aged Children. Retrieved from http://csunplugged.org

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2016, February). *Building mathematical knowledge with programming: Insights from the ScratchMaths project*. Suksapattana Foundation, Bangkok, Thailand.

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., et al. (2016). *Developing computational thinking in compulsory education-implications for policy and practice*. Join Research Center (European Commission), Seville. Retrieved from http://publications.jrc.ec.europa.eu/repository/bitstream/JRC104188/jrc104188_computhinkreport.pdf.

Boyer, K., Buffum, P. S., Culbertson, K., Frankosky, M., Lester, J., Martinez-Arocho, A., … Wiebe, E. (2015, February). ENGAGE: A game-based learning environment for middle school computational thinking. In *Proceedings of the 46th ACM technical symposium on computer science education* (pp. 440–440). ACM, New York.

Cannell, C., Tofel-Grehl, C., & Searle, K. (2020). Using circuit playground express and maps to visualize population migration data. In Proceedings of the 14th international conference of the learning sciences (ICLS) 2020, Nashville, Tennessee.

Caplan, B., Covitt, B., Love, G., Berkowitz, A. R., Gunckel, K. L., McClure, C., & Moore, J. C. (2021). Using computational thinking and modeling to build water and watershed literacy. Connected Science Learning, 3(2). Retrieved from https://www.nsta.org/connected-science-learning/connected-science-learning-march-april-2021/using-computational-thinking.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). Computational thinking – A guide for teachers. Computing at School Community, 2015, 1–18. Retrieved from https://community.computingatschool.org.uk/ resources/2324.

Curzon, P., Bell, T., Waite, J., & Dorling, M. (2020). Computational thinking. In Sally A. Fincher and Anthony Robbins (Eds.), *Cambridge handbook of computer science education research*. Cambridge University Press, Cambridge, UK.

Curzon, P., & Grover, S. (2020). Guided exploration and unplugged activities. In Shuchi Grover (Ed.), *Computer science in K-12: An A-to-Z handbook on teaching programmin*g (pp. 63–74). Edfinity, Palo Alto, CA.

Curzon, P., & McOwan, P. W. (2017). *The power of computational thinking: Games, magic and puzzles to help you become a computational thinker.* World Scientific, Hackensack, NJ.

Damelin, D., Krajcik, J. S., Mcintyre, C., & Bielik, T. (2017). Students making systems models. *Science Scope, 40*(5), 78–83.

Denning, P. J. (2000). Computer science: The discipline. *Encyclopedia of Computer Science, 32*(1), 9–23.

DiSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy.* MIT Press, Cambridge, MA.

Dominguez, X., Grover, S., Kamdar, D., Leones, T., & Vahey, P (2021). Preschool problem solvers: Developing assessment tasks to measure young children's learning of computational thinking skills and practices. *Computer Science Education.*

Dominguez, X., Grover, S., & Vahey, P. (2020). Enriching mathematics and science with computational thinking: Co-designing preschool activities with educators and parents. In Session 'integrating STEM & computing in PK-12: Operationalizing computational thinking for STEM learning & assessment. In *Proceedings of the 14th international conference of the learning sciences.* ISLS, Nashville, TN.

Dong, Y., Catete, V., Jocius, R., Lytle, N., Barnes, T., Albert, J., … & Andrews, A. (2019, February). PRADA: A practical model for integrating computational thinking in K-12 education. In *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 906–912), Portland, OR.

Fields, D. A., Kafai, Y. B., Morales-Navarro, L., & Walker, J. T. (2021). Debugging by design: A constructionist approach to high school students' crafting and coding of electronic textiles as failure artefacts. British Journal of Educational Technology, 52: 1078–1092.

Futschek, G., & Moschitz, J. (2011, October). Learning algorithmic thinking with tangible objects eases transition to computer programming. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 155–164). Springer, Berlin, Heidelberg.

Gadanidis, G., Cendros, R., Floyd, L., & Namukasa, I. (2017). Computational thinking in mathematics teacher education. *Contemporary Issues in Technology and Teacher Education*, *17*(4), 458–477.

Grover, S. (2013). *Learning to code isn't enough.* EdSurge. Retrieved from https://www.edsurge.com/news/2013-05-28-opinion-learning-to-code-isn-t-enough.

Grover, S. (2017). Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom. In Peter J. Rich and Charles B. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 269–288). Springer, Cham, Switzerland.

Grover, S. (2018a). *Computational modeling: How can we manage cognitive load when students must simultaneously learn to code and code to learn in a STEM classroom?* Retrieved from https://www.shuchigrover.com/what-is-computational-modeling-and-how-can-we-manage-cognitive-load-when-students-must-learn-to-code-and-code-to-learn-simultaneously/.

Grover, S. (2018b). Helping students see Hamlet and Harry Potter in a new light with computational thinking. *edSurge.* Retrieved from https://www.edsurge.com/news/2019-12-19-how-an-unplugged-approach-to-computational-thinking-can-move-schools-to-computer-science.

Grover, S. (2021a). 'CTIntegration': A conceptual framework guiding design and analysis of integration of computing and computational thinking into school subjects. EdArXiv. Retrieved from https://doi.org/10.35542/osf.io/eg8n5

Grover, S. (2021b). Teaching and Assessing for Transfer from block-to-text programming in middle school computer science. In Charles Hohensee and Joanne Lobato (Eds.), *Transfer of learning: Progressive perspectives for mathematics education and related fields* (p. 251). Springer, Cham, Switzerland.

Grover, S., Dominguez, X., Kamdar, D., Vahey, P., Moorthy, S., Rafanan, K., & Gracely, S. (2019, February). Integrating computational thinking in informal and formal science and math activities for preschool learners. In *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 1257–1258). ACM, New York.

Grover, S., Dominguez, X., Kamdar, D., Leones, T., Vahey, P., & Gracely, S. (2021, manuscript under revision). Strengthening Early STEM Learning by Integrating CT into Science and Math Activities at Home. In ACM Special Publication on K-5 Computational Thinking. ACM. New York.

Grover, S., et al. (2020). Integrating STEM & computing in PK-12: Operationalizing computational thinking for STEM learning & assessment. In *Proceedings of the 14th international conference of the learning sciences*. ISLS, Nashville, TN. Presentation slidedeck. Retrieved from https://bit.ly/STEMC-Integration.

Grover, S., Fisler, K., Lee, I., & Yadav, A. (2020). Integrating Computing and Computational Thinking into K-12 STEM Learning. In *Proceedings of the 51st ACM technical symposium on computer science education* (pp. 481–482). ACM, New York

Grover, S., Jackiw, N., & Lundh, P. (2019). Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school. *Computer Science Education*, *29*(2–3), 106–135.

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. (2018). Computational thinking: A competency whose time has come. *Computer Science Education: Perspectives on Teaching and Learning in School*, *19*, 1257–1258.

Grover, S., Pea, R. D., & Cooper, S. (2014). *Expansive framing and preparation for future learning in middle-school computer science*. International Society of the Learning Sciences, Boulder, CO.

Gu, X., Heller, M. A., Li, S., Ren, Y., Fisler, K., & Krishnamurthi, S. (2020, August). Using Design Alternatives to Learn About Data Organizations. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (pp. 248–258). ACM, New York.

Hadad, R., Thomas, K., Kachovska, M., & Yin, Y. (2020). Practicing formative assessment for computational thinking in making environments. *Journal of Science Education and Technology*, *29*(1), 162–173.

Hermans, F., & Aivaloglou, E. (2017). To scratch or not to scratch? A controlled experiment comparing plugged first and unplugged first programming lessons. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 49–56). ACM, New York.

Hoyles, C. (2020). Programming and mathematics: Insights from research in England. Computing education research seminars, Raspberry Pi Foundation. UK. Retrieved from https://www.raspberrypi.org/computing-education-research-online-seminars/previous-seminars/#programming-and-mathematics-insights-from-research-in-england.

Huang, W., & Looi, C. K. (2020). A critical review of literature on "unplugged" pedagogies in K-12 computer science and computational thinking education. *Computer Science Education*, 31(1), 83–111.

Hubwieser, P., Giannakos, M. N., Berges, M., Brinda, T., Diethelm, I., Magenheim, J., … & Jasute, E. (2015). A global snapshot of computer science education in K-12 schools. In *Proceedings of the 2015 ITiCSE on working group reports* (pp. 65–83).

Hutchins, N. M., Biswas, G., Maróti, M., Lédeczi, Á., Grover, S., Wolf, R., … McElhaney, K. (2020a). C2STEM: A system for synergistic learning of physics and computational thinking. *Journal of Science Education and Technology*, *29*(1), 83–100.

Hutchins, N. M., Biswas, G., Wolf, R., Chin, D., Grover, S., & Blair, K. (2020b). Computational thinking in support of learning and transfer. In *Proceedings of the international conference of the learning sciences (ICLS)*, Nashville, TN,

Jackiw, N. (2004). The geometer's sketchpad. Retrieved 15 June 2005 from http://www.keypress.com/sketchpad/modules.html

Jeon, S., Metcalf, S., Dickes, A., & Dede, C. (2020, April). Elementary teacher perspectives on a blended computational modeling and ecosystem science curriculum. In *Society for information technology & teacher education international conference* (pp. 46–55). Association for the Advancement of Computing in Education (AACE).

K12cs.org. 2018. Computational thinking. Retrieved from https://k12cs.org/computational-thinking/.

Kafai, Y. B. (2016). From computational thinking to computational participation in K-12 education. *Communications of the ACM*, *59*(8), 26–27.

Kamdar, D., Dominguez, X., Grover, S., Vahey, P., Rafanan, K., Gracely, S., & Leones, T. (2020). Researchers, teachers and families co-design resources linking computational thinking with math and science in preschool. In *Proceedings of the annual meeting of the AERA*, San Francisco, CA.

Ketelhut, D. J., Mills, K., Hestness, E., Cabrera, L., Plane, J., & McGinnis, J. R. (2020). Teacher change following a professional development experience in integrating computational thinking into elementary science. *Journal of Science Education and Technology*, *29*(1), 174–188.

Knuth, D. E. (1981). *Algorithms in modern mathematics and computer science* (pp. 82–99). Springer, Berlin, Heidelberg.

Knuth, D. E., & Pardo, L. T. (1980). The early development of programming languages. In N. Metropolis, J. Howlett, and Gian-Carlo Rota (Eds.), *A history of computing in the twentieth century* (pp. 197–273). Academic Press, New York.

Ko, A. J., Oleson, A., Ryan, N., Register, Y., Xie, B., Tari, M., … Loksa, D. (2020). It is time for more critical CS education. *Communications of the ACM*, *63*(11), 31–33.

Krishnamurthi, S., Schanzer, E., Politz, J. G., Lerner, B. S., Fisler, K., & Dooman, S. (2020). Data science as a route to AI for middle-and high-school students. *arXiv preprint arXiv:2005.01794*.

Lavigne, H., Orr, J., & Wolsky, M. (2018, March). Exploring computational thinking in preschool math learning environments. In *Society for information technology & teacher education international conference* (pp. 38–43). Association for the Advancement of Computing in Education (AACE).

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

Li, Y., Schoenfeld, A. H., Graesser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2020). On computational thinking and STEM Education. *Journal for STEM Education Research, 3*, 147–166.

Madkins, T. C., Martin, A., Ryoo, J., Scott, K. A., Goode, J., Scott, A., & McAlear, F. (2019, February). Culturally relevant computer science pedagogy: From theory to practice. In *2019 research on equity and sustained participation in engineering, computing, and technology (RESPECT)* (pp. 1–4). IEEE, Piscataway, NJ.

Malyn-Smith, J., Lee, I. A., Martin, F., Grover, S., Evans, M. A., & Pillai, S. (2018). Developing a framework for computational thinking from a disciplinary perspective. In *Proceedings of the international conference on computational thinking education* (p. 5). The Education University of Hong Kong, Hong Kong.

Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record*, *108*(6), 1017–1054.

Moore, T. J., Brophy, S. P., Tank, K. M., Lopez, R. D., Johnston, A. C., Hynes, M. M., & Gajdzik, E. (2020). Multiple representations in computational thinking tasks: A clinical study of second-grade students. *Journal of Science Education and Technology, 29*(1), 19–34.

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

Papadakis, S., Kalogiannakis, M., & Zaranis, N. (2016). Developing fundamental programming concepts and computational thinking with ScratchJr in preschool education: A case study. *International Journal of Mobile Learning and Organisation, 10*(3), 187–202.

Papert, S. A. (1972). Teaching children to be mathematicians versus teaching about mathematics. *International Journal of Mathematical Education in Science and Technology*, *3*(3), 249–262.

Papert, S. A. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York.

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism, 36*(2), 1–11.

Phil Vahey Emmott, S., & Rison, S. (2005). Towards 2020 science, Microsoft research. Retrieved from https://www.scienceinparliament.org.uk/wp-content/uploads/2013/09/sip65-4-17.pdf.

Pierson, A. E., Brady, C. E., & Clark, D. B. (2020). Balancing the environment: Computational models as interactive participants in a STEM classroom. *Journal of Science Education and Technology*, *29*(1), 101–119.

Puttick, G., & Tucker-Raymond, E. (2018). Building systems from scratch: An exploratory study of students learning about climate change. *Journal of Science Education and Technology, 27*(4), 306–321.

Rich, K. M., Binkowski, T. A., Strickland, C., & Franklin, D. (2018). Decomposition: A k-8 computational thinking learning trajectory. In *Proceedings of the 2018 ACM conference on international computing education research* (pp. 124–132). ACM, New York.

Rich, K. M., Strickland, C., Binkowski, T. A., & Franklin, D. (2019). A k-8 debugging learning trajectory derived from research literature. In *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 745–751). ACM, New York.

Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2017, August). K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In *Proceedings of the 2017 ACM conference on international computing education research* (pp. 182–190). ACM, New York.

Rowe, E., Asbell-Clarke, J., Cunningham, K., & Gasca, S. (2017, October). Assessing implicit computational thinking in zoombinis gameplay: Pizza pass, fleens & bubblewonder abyss. In *Extended abstracts publication of the annual symposium on computer-human interaction in play* (pp. 195–200). ACM, New York.

Salac, J., & Franklin, D. (2020, June). If they build it, will they understand it? Exploring the relationship between student code and performance. In *Proceedings of the 2020 ACM conference on innovation and technology in computer science education* (pp. 473–479), Portland, OR.

Sengupta, P., Dickes, A., & Farris, A. (2018). Toward a phenomenology of computational thinking in STEM education. In *Computational thinking in the STEM disciplines* (pp. 49–72). Springer, Cham, Switzerland.

Shulman, L. (1987). Knowledge and teaching: Foundations of the new reform. *Harvard Educational Review, 57*(1), 1–23.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158.

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798.

Taylor, S., Min, W., Mott, B., Emerson, A., Smith, A., Wiebe, E., & Lester, J. (2019). IntelliBlox: A toolkit for integrating block-based programming into game-based learning environments. In *2019 IEEE blocks and beyond workshop (B&B)* (pp. 55–58). IEEE Computer Society, Piscataway, NJ.

Tissenbaum, M., Sheldon, J., & Abelson, H. (2019). From computational thinking to computational action. *Communications of the ACM, 62*(3), 34–36.

Touretzky, D., Gardner-McCune, C., Martin, F., & Seehorn, D. (2019, July). Envisioning AI for K-12: What should every child know about AI?. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 33, pp. 9795–9799). AAAI Press, Palo Alto, CA.

Vogel, S., Hoadley, C., Ascenzi-Moreno, L., & Menken, K. (2019, February). The role of translanguaging in computational literacies: Documenting middle school bilinguals' practices in computer science integrated units. In *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 1164–1170). ACM, New York.

Waite, J. L., Curzon, P., Marsh, W., Sentance, S., & Hadwen-Bennett, A. (2018). Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. *International Journal of Computer Science Education in Schools, 2*(1), 14–40.

Washington, A. N. (2020). When twice as good isn't enough: The case for cultural competence in computing. In *Proceedings of the 51st ACM technical symposium on computer science education* (pp. 213–219). ACM, New York.

Waterman, K. P., Goldsmith, L., & Pasquale, M. (2020). Integrating computational thinking into elementary science curriculum: An examination of activities that support students' computational thinking in the service of disciplinary learning. *Journal of Science Education and Technology, 29*(1), 53–64.

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.

Weintrop, D., & Grover, S. (2020). JavaScript, python, scratch, or something else? Navigating the bustling world of introductory programming languages. In Shuchi Grover (Ed.), *Computer science in K-12: An A-to-Z handbook on teaching programming* (pp. 99–112). Edfinity, Palo Alto, CA.

Wendell, K., Shaw, F., Kshirsagar, K., Danahy, E., Bernstein, D., Puttick, G., & Cassidy, M. (2020). Navigating Dual Goals of Team Collaboration and Design Concept Development in a Middle School Bio-Inspired Robotics Unit. In Gresalfi, M. and Horn, I. S. (Eds.), The Interdisciplinarity of the Learning Sciences, 14th International Conference of the Learning Sciences (ICLS) 2020, Volume 2 (pp. 895–896). Nashville, Tennessee: International Society of the Learning Sciences.

Wilkerson, M. H. & Fenwick, M. (2017). The practice of using mathematics and computational thinking. In C. V. Schwarz, C. Passmore, & B. J. Reiser (Eds.), Helping Students Make Sense of the World Using Next Generation Science and Engineering Practices. Arlington, VA: National Science Teachers' Association Press. pp. 181–204.

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

Yadav, A., Gretter, S., Good, J., & McLean, T. (2017). Computational thinking in teacher education. In Peter Rich and Charles Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 205–220). Springer, Cham, Switzerland.

Yadav, A., Larimore, R., Rich, K., & Schwarz, C. (2019, March). Integrating computational thinking in elementary classrooms: Introducing a toolkit to support teachers. In *Society for information technology & teacher education international conference* (pp. 347–350). Association for the Advancement of Computing in Education (AACE).