Exploring Computational Thinking Concepts, Practices, and Dispositions in K-12 Computer Science and Engineering

By

Amanda M. Bell

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Learning, Teaching, and Diversity

May 31, 2018

Nashville, Tennessee

Approved:

Melissa S. Gresalfi, Ph.D.

Douglas B. Clark, Ph.D.

Corey E. Brady, Ph.D.

TABLE OF CONTENTS

Introduction

Increasingly, educators and policymakers value computer science (CS) education for its ability to prepare students for the growing number of jobs in computing fields and for its potential to equip learners with problem-solving skills and technological knowledge. While the traditional method of learning CS through programming teaches students about programming languages and algorithms, students should also have access to the concepts and practices computer scientists use to solve problems, referred to as *computational thinking (CT)*. CT empowers learners to use programming as a tool to generate innovative solutions to problems, to become thoughtful technology users in their everyday lives, to apply logical thinking to a variety of situations, and to prepare for jobs using technology across a variety of fields.

New initiatives in CS education call for the integration of CT into K-12 schools. President Obama's Computer Science for All initiative articulated the need to expose all students to "computational thinking skills that are relevant to many disciplines and careers" (Smith, 2016). CT was one of the five main conceptual strands in the 2011 K-12 CS standards developed by the Computer Science Teachers Association (CSTA). Now CSTA, along with the Association for Computing Machinery (ACM), are updating the CS standards and integrating CT throughout the concepts and practices (CSTA & ACM, 2016). At the same time, CSTA and the International Society of Technology in Education (ISTE) created a toolkit to help teachers and school leaders advocate for and develop a CT curriculum (ISTE & CSTA, 2011). These initiatives demonstrate educators' and policymakers' focus on CT as an important part of CS education for all students.

Taking CT learning a step further, many researchers are beginning to explore the integration of CT with STEM (science, technology, engineering, and mathematics) fields and other disciplines. The Next Generation Science Standards (NGSS) also include examples of CT in their science and engineering practices for grades K-12 (NGSS, 2013). By incorporating CT into existing school subjects, researchers hope to address several factors that hinder "CT for all" efforts: (1) all students would be exposed to CT if it is part of their core subjects rather than an elective computing course; (2) it would save time to teach CT during an existing subject rather than adding a course to busy school schedules; (3) it may be easier to train teachers if CT is incorporated as part of what they already do; and (4) not all schools have access to advanced technology to support CS classes (Horn et al., 2014; Hu, 2011; Sneider, Stephenson, Schafer, & Flick, 2014; Weintrop et al., 2016; Wilensky, Brady, & Horn, 2014). In fact, a study examining school-wide integration of computing at the elementary level found that classroom teachers could only teach computing by integrating it into their content areas because the pre-existing curriculum was too time-consuming to introduce computing on its own (Israel et al., 2015).

Despite these initiatives in K-12 education, researchers still do not have a strong consensus about what CT is and how it is applied in different contexts. A better understanding of how people use CT in a variety of fields is necessary to meaningfully integrate CT across school subjects (Grover & Pea, 2018). Furthermore, in order to engage all students in CT, not just those predisposed to computing, researchers need to understand how students can legitimately participate in a CT learning community through richly varied experiences. Therefore, my questions in this paper are: **what does CT look like in different contexts, and how do learners engage in CT in these different K-12 learning communities?** In the section titled "Contexts of Focus" below, I discuss why I chose to explore CT in the context of computer science and engineering in this paper. My goal is to expand ideas of what it means to be a competent computational thinker by identifying elements of CT from CS then discussing how people use CT skills in other disciplines outside CS.

Computational Thinking in Society

Computers are no longer a specialized tool but are pervasive in our society. Therefore, "the ability to extend the power of human thought with computers and other digital tools has become an essential part of our everyday lives and work" (Barr, Harrison, & Conery, 2011, p. 23). Computer scientists employ CT skills to ask questions and solve problems across disciplines using computers. At a workshop organized by the National Research Council in 2009 to discuss the scope and nature of CT, attendees argued that CT is "comparable in importance and significance to the mathematical, linguistic, and logical reasoning that society today agrees should be taught to all children" (National Research Council, 2010, p. 3).

This emphasis on the importance of computing in our society is not new. For decades, there have been appeals for widespread integration of computing skills into all levels and types of education (Weintrop et al., 2016). In 1962, Perlis, the first recipient of the ACM Turing Award, claimed that all undergraduates should learn programming (Guzdial, 2008). Papert (1980) later argued for introducing a literacy of computing to children, and he used CT to describe the ability of computing to empower ideas (Papert, 1996). diSessa (2000) called for a new form of computational literacy that changes the way we all communicate, learn, and live with technology. More recently, there has been a resurgence in interest in CT from the perspective of 21st century skills preparing students for a job market that increasingly involves technology creation and use (Grover & Pea, 2013; Wing, 2006).

However, participation in computing fields remains low in the U.S. By 2024, there could be a predicted 1.1 million jobs in computing fields (National Center for Women and Information Technology, 2017), but less than 17,000 people graduated with computer science or programming degrees in 2015, including fewer than 3,000 women (Snyder, 2016). Exposing students to computer science before they enter college is essential for increasing the number and variety of computing majors, as students are 8 times more likely to major in computer science after taking an AP CS course in high school (Mattern, Shaw, & Ewing, 2011). Furthermore, jobs in computing tend to be intellectually rewarding and high paying; the median annual wage was over $80,000 in 2015, much higher than the overall median annual wage for all jobs of $36,000 (Bureau of Labor Statistics, 2017). But the vast majority of those jobs are held by men. The proportion of women in computing jobs has actually decreased since 1990, down to 25% in 2015 (NCWIT, 2017). Just 7% of workers in computing in 2014 identified as Black and 7% as Hispanic (Beckhusen, 2016). The high wages of computing jobs highlights the value our society places on that work, but the diversity of participation in those jobs is limited. "If the population of people creating software is more closely matched to the population using software, the software designed and released will probably better match users needs" (Kelleher & Pausch, 2005, p. 131). Computer scientists and engineers design tools that are integral to lives across the world, so it makes little sense that the vast majority of those designers only represent one type of person. The challenge we face today is both to increase engagement in technological creation and to ensure the field is representative of our diverse population.

Stereotypes of what CS jobs entail and the image of loner, nerdy, male programmers still perpetuate and undermine diversity in computing fields. Stereotypes often serve as gatekeepers for women in particular, hindering learning (Cheryan, Master, & Meltzoff, 2015) and decreasing sense of belonging (Master, Cheryan, & Meltzoff, 2016). K-12 schools play an important role in introducing a variety of students to CS and potentially changing their perceptions of what CS is and who can participate in it.

CT vs. CS vs. Programming

The lines between CT, CS, and programming can sometimes be blurred in educational contexts. The best articulation of the interconnections among the three I have found is a blog post written by software engineer, author, and startup co-founder Yevgeniy Brikman. Rather than learning programming, Brikman (2014) articulates the importance of learning to think. First, programming is just what it sounds like: writing code, whether on paper or on the computer. Programming involves speaking a particular language to a computer to get the computer to do something. It could involve creating a piece of software, an app, a website, or it could be as simple as using a computer to calculate a multiplication problem. Programming is a tool to help solve a problem or perform a computation. It is a common tool used by computer scientists, but programming does not define what computer science is. The brain is also a tool for computation; it can think logically, solve problems, and run calculations.

Computer science, on the other hand, is a broad discipline that includes software engineering, algorithm design, problem solving, computational theory, artificial intelligence, information theory, logic, and more. Every discipline involves particular ways of thinking, and part of learning to participate in a field involves learning those ways to think. For instance, physics gives us a particular way of thinking about the physical world through calculus and in terms of matter, motion, and forces. Chemistry gives us a way of thinking about matter in terms of structure, properties, and reactions. Computational thinking refers to the ways of thinking computer scientists use. The idea of CT is to articulate how computer scientists think and solve problems so others can learn to think in a similar way. Programming is just one tool for exercising that way of thinking, in the same way chemists perform experiments using test tubes (Brikman, 2014).

While chemistry may involve its own way of thinking about the world, the skills chemists' use also leak into other fields of study, like physics, biology, medicine, and engineering. For instance, biochemistry is a hybrid field that takes the ability to think about chemical processes and applies it to living creatures. According to the UK Biochemical Society, this cross-application has contributed significant advances in the areas of health, disease, technology, and more (Biochemical Society, 2017), demonstrating that applying disciplinary thinking skills across contexts creates innovative solutions and technological advances. In much the same way, the ability to think logically and computationally can leak outside the CS field and help solve problems in other endeavors. Regardless of the subject matter, the constant increase in technology across different cultures strengthens the call for an understanding of how people use CT to think with and solve problems through technology.

Computer literacy is another term people use around computing professions, but it often can refer to the knowledge of specific programs, like Word or Adobe Photoshop, and components of computers, like using a mouse and keyboard. Some researchers use the term computational literacy, which is closer to CT but not exactly the same. While reading and writing are fundamental to education, diSessa (2000) argues that computational literacy is also crucial to creating knowledgeable people. In much the same way the invention of the printing press changed mainstream literacy, the proliferation of computers changes our current structures of literacy and what it means to be a literate person. Computational literacy is a "material intelligence" that is mediated by materials such as symbols and representations, it involves the ways we think in the presence of those materials, and it is a social endeavor, developing with other individuals across time and space (diSessa, 2000). The first pillar of computational literacy -- materials -- is similar to the notion of programming as a tool and programming languages as an inscription or symbol system of a computer-based literacy. The second pillar -- mental processes -- includes the ways in which people think and solve problems with computers and programming tools, which captures the essence of CT. The third pillar -- social aspects -- goes beyond the collaborative processes central to CT to describe the ways in which knowledge builds over long periods of time and with the help of many different people. Therefore, computational literacy includes CT but has the more ambitious goal of fundamentally changing modern education, which goes beyond the scope of this paper.

However, educators are increasingly focused on teaching CT rather than just programming. Whereas learning to program gives students tools for exercising CT, learning how to think like a computer scientist helps students adapt to constantly changing technological innovations and computing problems. While researchers are still working to build a clear operational definition of CT (Grover & Pea, 2013), many agree with the idea of CT as "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing, 2006, p. 33). CT is a way of thinking that allows people to create algorithms or solutions to problems in such a way that they can be carried out by a computational agent, whether that is a computer or a human (Brennan & Resnick, 2012). Computational participation is another term used to emphasize the social and creative practices at the core of what computer scientists do (Kafai, 2016). Computational participation is also a form of diSessa's (2000) third pillar, with its focus on the social aspects of computing. Ultimately, as students learn CT, they develop ways of participating in communities of learners and computational thinkers (National Research Council, 2010).

Contexts of Focus: Computer Science and Engineering

With initiatives like CS for All and the inclusion of CT in computing, science, and engineering education standards, educators need ways to get all students, not just those predisposed to computers, engaged in CT skills. Many public and private funding agencies support STEM and STEAM (integrating arts into STEM subjects) projects, highlighting the public's interest in engaging students in multidisciplinary experiences that focus on not just computing but also integrating those skills across science, mathematics, engineering, and the arts (Feldman, 2015; Handelsman & Smith, 2016; Jagodzinski, 2016). Research aimed at better understanding how these subjects intersect is essential for creating learning environments that truly embody interdisciplinary mindsets and allow students to apply problem solving strategies across disciplines. While CT is closely tied to the T (Technology) in STEAM through its roots in computing, in this paper, I seek to improve understandings of

CT while advancing its integration with other disciplines by exploring connections to other STEAM subjects, namely, engineering.

To explore applications of CT in technology, I focus on computer programming as a tool for engaging in CT skills. Programming is a common educational tool for CT. It provides many problem-solving activities and empowers students to learn through creating their own artifacts that can be shared with others (Papert, 1980). In other words, programming can introduce students to the tools used by professional computer scientists, but it can also empower students to use CT skills to solve a variety of computational problems. Designers, researchers, and educators have developed numerous tools for learning programming, both with and without computers.

I chose to explore engineering (the E in STEAM) as the second disciplinary context for defining CT for several reasons. Given the recent increase in engineering education in U.S. classrooms (Catterall, 2013; National Academy of Engineering and National Research Council, 2009; Robelen, 2013), it is important to understand how engineering connects to other school subjects, including CS. Engineering can improve student learning in other STEAM contexts by connecting disciplinary concepts and practices to real-world problems (Catterall, 2013; NAE & NRC, 2009). At the same time, engineering "can provide a way to integrate the STEM disciplines meaningfully" (Moore et al., 2014, p. 2) by applying skills from mathematics, science, and CS to create solutions to problems. Thus, engineering may be a powerful context for learning and applying CT concepts and practices. One branch of engineering, software engineering, is part of the computer science field and involves significant programming work. But beyond software engineering, recent NGSS standards illustrated connections between CT and broader engineering practices (NGSS, 2013). CT and CS education have roots in constructionism, which emphasizes learning through the design, building and discovery practices at the core of engineering (Lucas, Hanson, Claxton, & Centre for Real World Learning, 2014; Papert, 1980). It then follows that CT would have deep connections to engineering processes, making the engineering field a strong candidate for exploring what CT is and how it overlaps with different disciplines.

Theoretical Framework

By looking at how CT is re-contextualized in CS and engineering, I seek to understand how students can engage in CT in rich and meaningful ways. Both students who like programming and those with interests outside computing should be able to participate in the practices of CT and take on legitimate roles in the CT learning community. I am curious about how students can participate in CT in rich and meaningful ways and how participation relates to students' interests in, experiences with, and beliefs about computing. Therefore, in the next sections, I draw on perspectives of situated learning and legitimate peripheral participation (Lave & Wenger, 1991) to understand what it means to be a computational thinker in different contexts.

Historically, research in education focused on understanding the knowledge students acquire and how they learn it. The development of situative theories demonstrates that knowledge and practices are tied together, and the activities people engage in are central to and make up the knowledge being developed (Cobb & Bowers, 1999). In situative perspectives, learning involves a change in participation with a set of resources or activities in addition to changes in ways of thinking (Gresalfi, Martin, Hand, & Greeno, 2009; Hand & Gresalfi, 2015; Lave & Wenger, 1991; Nasir & Cooks, 2009). Essential to this framework is the idea that productive learning happens through *legitimate peripheral participation* (LPP), that is, when people have access to the core practices of a community of practitioners and opportunities to participate more fully over time (Lave & Wenger, 1991). LPP asks what kinds of social situations provide a context for learning, rather than what cognitive processes take place in the individual. LPP describes engagement in social practice as distributed among co-participants in a learning context, with a focus on participation in the social world. In this way, learning occurs through interactions in communities of practice both inside and outside the formal classroom.

Communities of practice (CoP) are groups of people engaging in a similar craft or profession (Lave & Wenger, 1991; Wenger, 2010). Through sharing information and experiences, members learn from each other and develop their identities in relation to a broader community. People in a CoP are active practitioners, as opposed to a community of interest that involves people with a shared interest without dependence on expertise or practice (Wenger, 2010). CoPs involve relations among people, activities, and the world across time and space. Different communities overlap with one another, and people are members of multiple communities at a time.

During the learning process, the community itself changes. It is not just the novice who learns, but the expert and the skill itself also change in some way. LPP moves learners toward fuller participation in a community. People start by participating in tasks that are meaningful and productive to the community but have minimal risk. Through this work, novices slowly become familiar with the languages, tasks, and practices of a community. As they continue to participate over time, they take on more aspects and responsibilities, and their participation becomes more central to the functioning of the community. The legitimate peripheral nature implies that learners can change perspectives in the community as part of learning and developing identities, and there is no single core to the community. There are different ways of being full participants.

As an example of learning through LPP, we can think about the trajectory of a doctoral student. First year graduate students working as research assistants may start by reading papers written by their advisors and correcting typos or filling in missing references in the bibliography. This work is necessary and important, which allows the newcomer to contribute to the academic community even in a peripheral way, but it is not too costly if mistakes are made. Over time, the graduate student may contribute to writing a paper with an advisor, collect data with a research team, then eventually conduct their own research and publish their own papers. In their first year of study, a student may not feel like an important member of the community, but over years of becoming a fuller participant, they may develop a closer relationship and sense of belonging with the academic community. In this way, the graduate students are learning, changing their forms of participation and roles within the community, and constructing identities in relation to the community of academic practitioners in their field. It may not always feel like it to graduate students, but as long as they have access to expert mentors and authentic practices, they are slowly transforming their identities and learning to become real researchers and full members of the field. Thus, learning involves becoming a member of a community and results in a change in relationship with respect to a community. A newcomer's "changing knowledge, skill, and discourse are part of a developing identity" (Lave & Wenger, 1991, p. 122).

Many scholars point to issues of identity as critical to learning and engagement (e.g. Boaler & Greeno, 2000; Hand & Gresalfi, 2015; Holland, Lachicotte, Skinner, & Cain, 1998; Wenger, 1998). In mathematics education, researchers have argued for the importance of understanding how students develop views of themselves as members in the discipline and as capable of learning and doing mathematics (Martin, 2000; Nasir, 2002; Nasir & de Royston, 2013). How learners view themselves affects how they engage in learning contexts, and their engagement is shaped by the opportunities afforded for participation (Nasir & de Royston, 2013).

In LPP, identity and learning are inextricably tied to changes in participation with resources and activities in a particular social context (Greeno & Gresalfi, 2008; Hand & Gresalfi, 2015; Lave & Wenger, 1991; Nasir & Cooks, 2009). In other words, both learning and identity are the result of participation in communities of practice (Wenger, 1998). Identities form through participation, and learning involves becoming a legitimate participant and member of the community (Lave & Wenger, 1991). Learning involves a shift in participation with artifacts or resources, while the ways in which resources are used and participation occurs depend on the learner's identity (Nasir & Cooks, 2009; Gresalfi et al., 2009). Learning involves changes in ways of acting in relation to the norms and resources of a community, and at the same time, identity affects what people learn, how they engage, and what they choose to pursue (Bishop, 2012).

Identities "play a fundamental role in enhancing (or detracting from) our attitudes, dispositions, emotional development, and general sense of self" (Bishop, 2012, pp. 34–35). Rather than a single, static sense of self, identity is a mixture of changing representations negotiated based on how people view themselves, how they are positioned by others, their engagement with norms, practices, cultural tools, past experiences, ways of participating, feelings and beliefs, and the particular social context (Bishop, 2012; Gee, 2000; Hand & Gresalfi, 2015; Holland et al., 1998; Wenger, 1998). Any individual has many identities across communities that are negotiated and inform each other (Holland et al., 1998). Individuals can control their identities in some ways but they are limited by relations of structure and power in the broader context of participation (Brickhouse & Potter, 2001). A student may claim an identity but their interactions with others may adjust the strength or form of that identity as it is enacted over time and space.

CS learning contexts, such as the classroom or informal club, are particular communities of practice where learning occurs through changes in participation in relation to the norms and practices of the community. At the same time, identities develop according to "who students are, who they can be, and who they want to be, as sanctioned by the norms of the classroom" (Tan, Calabrese Barton, Kang, & O'Neill, 2013, p. 1145). Identity involves how people see themselves in relation to the community, but also how others see them and how they are allowed to participate and contribute to the community. Both individual and shared identities are continuously negotiated through interaction with others and through engaging in the practices of a community. Students' engagement, persistence, and goals mediate both identity and learning (Nasir & Cooks, 2009).

Nasir & Hand (2008) use the term "practice-linked identities" to refer to this notion of identity as a sense of self tied to activity. These "are the identities that people come to take on, construct, and embrace that are linked to participation in particular social and cultural practices" (p. 147). This view of identity is tied to the notion of engagement. Various practices afford different types of engagement, which support practice-linked identity development in different ways. For instance, when an individual feels a close connection between their developing sense of self and the practices of a community, the individual is more likely to be engaged and to participate intensely (Nasir & Hand, 2008). Hence, some settings support engagement for particular individuals better than others do. At the same time, some activities may afford more ways of participating than others, allowing students to engage in different ways. This in turn supports students to see themselves as capable of participating in those practices and developing a productive sense of self in relation to the particular community.

As work on identity construction reveals, membership in a community is mediated by the possible forms of participation people have access to, including physical and social tools. Learners must have opportunities to use the tools and participate in the activities of a community to develop understanding and a sense of belonging. For instance, if novices can directly observe expert practice, then they have a better understanding of the organization of the community and where their work fits in. But if newcomers have little access to the tools and broader community, they can stagnate in the same role over many years and never achieve fuller participation (Lave & Wenger, 1991). Thus, access to resources and opportunities to learn are essential. Therefore, we must first understand what the community's tools and activities are before we can design for learning. In the case of integrating CT into K-12 schools, researchers must first identify the central characteristics of the professional computing community, including the domain of knowledge, the practices or activities, and the ways of approaching activity within the community.

In the LPP framework, changes in participation occur in three different ways. First, the tools or resources people leverage to solve problems change as they develop new knowledge and better understandings. Instead of tools, I use the term *concepts* in this paper to reflect the language most researchers use to classify what people learn when they engage in CT (e.g. Barr & Stephenson, 2011; Brennan & Resnick, 2012; Weintrop et al., 2016). Learning also involves engaging in disciplinary *practices*. That is, learners make use of the core activities or processes used by the community. Finally, over time, learners develop identities towards the disciplinary community. The activities children experience in formal and informal learning contexts help form their identities in relation to different disciplines, which affects the ways in which students use their knowledge and approach new problems (Boaler, 2002). Boaler (2002) calls this type of identity a "disciplinary relationship." These identities describe relationships to others, ways of being, or views of oneself in relation to the discipline. In this paper, I use the term *dispositions* to connect this type of disciplinary identity with work on productive dispositions. Research on productive dispositions claims that students need to develop thinking skills along with the appropriate dispositions to use those skills (Gresalfi & Cobb, 2006; McLeod, 1992; Schoenfeld, 1992). Making use of knowledge depends on the practices students have engaged in and whether they have a productive relationship or disposition towards the discipline (Boaler, 2002). My use of the term disposition also corresponds with Brennan and Resnick's (2012) focus on dispositions in their assessments of CT skills.

As mentioned briefly, this framework of concepts, practices, and dispositions corresponds with how many researchers now talk about CT in educational contexts (Barr & Stephenson, 2011; Brennan & Resnick, 2012; Shute, Sun, & Asbell-Clarke, 2017; Weintrop et al., 2016). CT concepts are notions or ideas used as tools in the construction of algorithms and problem solutions. CT practices include processes of constructing algorithms or solutions. While concepts describe knowledge and understandings, practices describe how people participate and use concepts while creating solution processes. At the same time, learners' dispositions towards CT affect their engagement and ways of participating in activities involving CT. Learners need not only the skills and knowledge of a community but also the inclination to recognize when skills are useful and the willingness to use them (Halpern, 1999). Brennan and Resnick (2012) refer to these understandings of oneself and relationships with the discipline of CT as "perspectives." However, in this paper I use the more familiar term "dispositions" as it connects to work on productive engagement and productive disciplinary relationships in the broader field of education.

I use these categories of concepts, practices, and dispositions to describe what it means to know and participate in CT in different contexts. In other words, participating in CT involves using one or more of the CT practices, and knowing CT involves understanding CT concepts or being able to describe and engage in CT practices. At the same time, CT dispositions affect whether people can productively engage in those practices and develop understandings of CT concepts.

In this paper, I draw on the framework of LPP with the goal of broadening ideas of what it means to be competent or legitimate computational thinkers. First, I describe the ways in which researchers define CT in CS contexts, and then I look to engineering to see how people in the field legitimately use CT practices when designing solutions to problems. By highlighting the overlaps between CT and other disciplines, I illustrate how people can legitimately engage in CT and act as competent computational thinkers outside the traditional boundaries of computing. While I separate learning and identity in the sections below on CT in engineering and CS to better structure and clarify the literature review, learning and identity are intricately tied together in LPP. They influence one another, and both change as a consequence of participation.

Literature Search

I first wanted to look at how researchers and policy makers characterize CT and CT learning in CS and in engineering. For the CT in CS track, I started with highly cited anchor readings, including Wing (2006; 2008) and Grover & Pea (2013), and I collected articles based on their citations as well as articles citing them. I continued looking at those subsequent articles' citations to find further work. Much of what I found, especially for empirical work, came from conference proceedings. I then searched for CT education standards, and I looked for national CS standards with CT embedded in them.

For CT in engineering, there were no anchor papers to base the search on, and I found a dearth of work on learning engineering in K-12 in general. Therefore, I extended my search beyond CT specifically to articles that talk about general engineering practices. I also searched for national engineering education standards that incorporate CT and focus on design, since design is where the overlap with CT applies across engineering fields.

I did not find many papers in the CS or CT fields that talked about identity in the way I conceptualize it here. Instead, I found papers looking at beliefs, motivation, and interest, so I include those constructs in my literature review on CT identity below. On the other hand, there is a significant amount of literature about identity in engineering fields. However, almost all the work focuses on adults (college students and professionals) and primarily around belief and identification, which is different from the conceptualization based on participation that I use in this paper. I use the existing literature to highlight potential findings related to CT identity from work on beliefs, interest, and motivation in CS and engineering, and I point to opportunities for further exploration.

Computational Thinking in Computer Science

Following the idea that learning to program also helps people learn how to think, this section aims to identify some of the concepts and practices involved in learning to think computationally through computer programming. CT and programming are deeply intertwined here, as efforts to define CT have started by looking at the skills programmers use to solve problems and formulate solutions. In this context, “solution” refers to the process of coming to an answer rather than the answer itself. While algorithms or programs are not answers on their own, they are solutions that can be carried out by a computational agent to produce an answer to a question or problem.

Concepts

In essence, “an algorithm is an abstraction of a step-by-step procedure for taking input and producing some desired output” (Wing, 2008, p. 3718). Since CT helps solve problems by creating algorithms, abstraction is at its core (Wing, 2006; 2008). Abstraction involves deciding what to pay attention to and what to ignore in representing and processing data (Weintrop et al., 2016). For example, when you want to print something from your computer, you only need to worry about finding the “print” button; you do not need to think about or understand the mechanics behind how a printer works, how data is sent to the printer, etc. In essence, much of the printing process is in a “black box” that users can ignore; you need only see the “print” button on your computer screen. Therefore, “the print command is an abstraction that shows the user only what he or she needs to see” (National Research Council, 2004, p. 16).

While abstraction is central to CT, it is still a broad concept and does not help educators understand how to implement CT and what to pay attention to. Some researchers have suggested that CT is similar to notions of procedural thinking developed by Seymour Papert (National Research Council, 2010). Procedural thinking involves thinking in and about procedures for performing actions. These could be everyday actions like giving someone directions or more complex tasks like developing programming algorithms. Procedural thinking helps people break down complex tasks into smaller components and debug errors in the solution processes (Papert, 1980).

Using Scratch, a block-based programming language developed based on principles from Papert's Logo language, Brennan and Resnick (2012) specified the connections between CT and procedural thinking. Specifically, CT concepts that programmers use to think procedurally and create algorithms include: (i) creating and following sequences of instructions; (ii) parallel instructions (executing multiple sets of instructions at the same time); (iii) using and organizing data; (iv) operating on data; and (v) elements of control flow like looping sets of instructions, conditionals (if this is true then do this), and events (when this happens then do this) (Brennan & Resnick, 2012; CSTA, 2017; Grover & Pea, 2013). An analysis of programmers with years of experience in the Scratch online community (https://scratch.mit.edu) showed that they make use of a wide variety of Scratch blocks that involve all of the concepts listed above (Brennan & Resnick, 2012). Furthermore, Bers and colleagues (2014) demonstrated that children as young as four can engage in these CT concepts, like sequencing instructions and control flow, through tangible programming activities. Thus, even novice programmers can legitimately participate in the CT community by accessing these core concepts through CS education.

Practices

Beyond understanding programming concepts, CT is ultimately a way of thinking that describes "processes of construction" (Brennan & Resnick, 2012, p. 6) used to solve problems. Solving problems using procedural thinking from programming involves actively developing, representing, testing, and debugging procedures or algorithms (Papert, 1980). These procedural thinking practices map onto similar CT practices. First, developing procedures or algorithms in CT involves the practice of being incremental and iterative (Brennan & Resnick, 2012; CSTA, 2017; Grover & Pea, 2018; Shute, Sun, & Asbell-Clarke, 2017). For example, designing an algorithm is not a consecutive process but involves adapting plans and going through cycles of brainstorming and creating. Even experienced programmers are likely to make errors when they first write new procedures, so revision is expected and not an indication of someone's lack of programming ability. Instead, both novice and expert programmers code a little bit, try it out, and adjust it or move forward based on what they find and the new ideas they generate (Brennan & Resnick, 2012). Hence, the process involves iterating on solution ideas.

Another practice many programmers use when developing procedures involves reusing or remixing solutions from others (Brennan & Resnick, 2012; CSTA, 2017. "Remixing" involves starting with a procedure someone else has written and changing it in some way to achieve a new goal. New technologies allow programmers to easily exchange ideas, access each other's work, and engage in reusing and remixing practices. Stack Overflow (http://stackoverflow.com/) is just one example of a popular online community where programmers of all levels help each other solve problems and share samples of code. Scratch also has its own online community (https://scratch.mit.edu/) for programmers to view, comment on, and remix each other's projects. These communities give learners access to the knowledge, skills, and work of more experienced members of the community. Newcomers and oldtimers can exchange thoughts, and newcomers can develop an understanding of the kinds of participation they are moving towards as they see examples of oldtimers' work and the kinds of interactions that are considered legitimate in the larger community of practitioners. Hence, these resources support learning and progression toward fuller participation in the programming and CT community (Lave & Wenger, 1991).

During the iterative development process, programmers test and debug to refine their solutions (Papert, 1980). "Debugging" is a process of finding and fixing errors (Bers et al., 2014; Brennan & Resnick, 2012; CSTA, 2017; Grover & Pea, 2013; 2018; Shute, Sun, Asbell-Clarke, 2017). Debugging starts by recognizing that something is not working as expected, then involves choosing to continue working towards the original goal or changing the desired goal. If the programmer decides to fix the problem, they will develop conjectures about what caused the problem, then finally attempt to solve the problem. This four-step debugging procedure can even be used by kindergarten-aged children (Bers et al., 2014), showing that this core practice is accessible to novice computational thinkers.

Along with developing solutions, representing procedures or solutions so they can be carried out by a computational agent is a core part of CT. In programming, the computational agent is usually a mechanical computer. Programmers use several related skills in this process. One skill involves reworking problems so they can be solved by a computer (ISTE & CSTA, 2011; Wing, 2006). Even after developing an idea that solves a problem, procedures must be specific, clear, and written in a particular way depending on the programming language and computational agent. In many cases, efficiency of solutions is an important consideration. The practice of working towards efficient solutions involves addressing constraints such as time it takes a solution to compute, the space the program takes to run and store, and even the simplicity of instructions so they can be reused and understood by others (Barr, Harrison, & Conery, 2011; Grover & Pea, 2013; Wing, 2008). These core CT practices fundamentally depend on the computational agent being used to carry out the solution, so programmers must consider and use them in various ways across programming contexts. Thus, part of learning to participate in the programming community involves developing the ability to adapt practices of design in different situations. This is especially true as learners gain expertise and move beyond a single programming language or environment.

Because expert programmers must adapt to many different environments, especially to keep up with the changing technologies and languages used to create programs, it is helpful if their solutions can transfer across a variety of problems. Hence, some researchers now identify the practice of generalizing a solution into a problem solving process as an important part of CT (Barr, Harrison, & Conery, 2011; Hu, 2011; Shute, Sun, Asbell-Clarke, 2017). Instead of a specific solution that only applies to a particular computational agent or programming language, these general processes are less formal and specific so they can be adapted to different environments. For instance, over the years, programmers have developed general algorithms for sorting lists of numbers. Programmers learn these algorithms and explain how they work using pseudo-code, images and diagrams, or paragraphs of text. With that general understanding of the algorithms for sorting numbers, programmers can then consider the environment and constraints they have, choose which algorithm is most appropriate, and code it in the specific programming language they need. For novices then, it does not make sense to memorize the procedures for sorting numbers in many different languages. Instead, learners should develop knowledge of general sorting algorithms and engage in the practice of reformulating and specifying those algorithms across contexts.

This practice of generalizing solutions relates to the idea of modularizing code. In modularizing, programmers break the problem down into simpler tasks and group lines of code according to the functions they perform (CSTA, 2017). These groups are often called "functions" or "methods" in object-oriented programming. As an example of modularization in a novice programming context, the Scratch learning environment allows learners to engage in this practice of modularization by creating separate stacks of code that run in response to an event that occurs in the larger program. For instance, Brennan and Resnick (2012) illustrate how a learner in the Scratch community uses modularization to split her code into three stacks. The first stack controls an object's movement, the second stack control's its visual appearance, and the third stack controls other events that occur in response to the object, like resetting a level in the game when the object collides with something. In this example, the learner also uses the concept of parallel procedures as all three of her stacks are set to start when they receive the same event command.

Modularization is a type of abstraction in which programmers build something complicated by combining smaller parts together (Brennan & Resnick, 2012; CSTA, 2017). The practice of representing and using data also draws on abstraction (Barr, Harrison, & Conery, 2011; CSTA, 2017; Grover & Pea, 2013; Hu, 2011; NRC, 2010; Wing, 2006; ISTE & CSTA, 2011; Shute, Sun, & Asbell-Clarke, 2017). "All information stored and processed by a computing device is referred to as data… As students use software to complete tasks on a computing device, they will be manipulating data" (CSTA, 2017, p. 2). Two ways programmers commonly work with different representations of data is by using different data types and structures, like arrays and lists, and by transforming data to make it more usable (CSTA, 2017). While newcomers do not typically start out by learning data structures, children can work with data in the form of variables. For instance, in Scratch, learners can explicitly use variables to store information and perform calculations. Variables can store data in the form of numbers or text. Scratch users can also choose to display the data in different ways, using a simple bar showing the value of the variable, having characters say the value of a variable, or by changing a visual or auditory output in response to a data value. In this way, novices can participate in a legitimate but peripheral form of the data use practice until they develop fuller understandings of data in computational contexts.

This section discussed several CT practices from programming, namely: being incremental and iterative, reusing or remixing solutions from others, testing and debugging, representing procedures so they can be carried out by a computational agent, generalizing a solution into a problem solving process, modularizing, and representing data. The next section dives into the dispositions CT researchers are beginning to recognize from studying programming.

Dispositions

CT dispositions affect learners' views of themselves, their ways of participating, their attitudes towards technology, and their perspectives on CT. Productive dispositions help learners engage and make progress in their learning as they shift towards fuller participation in the community. Drawing on programming and computer science at a meeting to develop an operational definition of CT, researchers and educators called attention to several relevant dispositions. These dispositions include dealing with complexity, persisting on difficult problems, dealing with ambiguity and open-ended problems, and collaborating with others on a shared goal (Barr, Harrison, & Conery, 2011; Barr & Stephenson, 2011; ISTE & CSTA, 2011). Brennan and Resnick (2012) used examples from programming in Scratch to describe two additional CT dispositions: using computation for self-expression and questioning the world about and with technologies.

First, it is not clear exactly what the differences are between the dispositions labeled as dealing with complexity, dealing with open-ended problems, and persisting on difficult problems. They all seem to overlap, and they are all mentioned together several times in the CSTA K-12 Computer Science Standards (CSTA, 2013). They all involve characteristics like patience, adaptability, accepting challenges, and ability to tinker or try things out (CSTA, 2013). They describe recognized ways of approaching problems and characteristics of successful problem solvers from other fields, including metacognitive skills and beliefs (Lester, 1994; Mayer, 1998). Therefore, these three CT dispositions can be combined to describe productive characteristics for approaching open-ended and complex problems in programming (CSTA, 2013). More research is needed to understand what these dispositions entail, how they affect learning and engagement, and how to foster them productively in educational programming environments.

The next disposition highlights the importance of collaboration in programming and CT. Collaboration occurs in both K-12 classrooms, such as through pair programming and group projects, and in the workplace, also in pair programming and through divisions of labor (Grover & Pea, 2018). In educational contexts, collaboration gives students access to others in the community as a resource for learning through asking questions, observing practices, and developing a broad overview of other roles and ways of participating in CT (Lave & Wenger, 1991). For young learners, these connections can occur when creating projects both with others and for others (Brennan & Resnick, 2012). For instance, collaborating with others allows novice programmers to ask questions of their peers, reuse others' code, and create lasting partnerships. By creating projects for others to use, learners must engage in new skills and concepts involved in understanding their audience, defining their goals, and disseminating their work. As another example, the Scratch online community supports similar collaborations among participants of all levels (Brennan & Resnick, 2012; Resnick et al., 2009). In order to collaborate in these ways, participants must be able to give each other feedback, make use of feedback in their work, understand different perspectives, and create both social and working relationships with other members of the programming community (CSTA, 2013; 2017). These dispositions help learners take on new roles as they productively engage with other programmers and computational thinkers.

While learning to program, students should also develop a disposition towards expressing ideas with technology. Rather than just consuming existing technologies, like browsing the Internet or texting friends, programmers can actually create and adapt technologies to solve problems in new ways (CSTA, 2017; Grover & Pea, 2018). For programmers, "computation is something they can use for design and self-expression. A computational thinker sees computation as a medium" for creativity and exploration (Brennan & Resnick, 2012, p. 10). Experienced programmers with well-developed knowledge and skills can create many different types of projects depending on their interests, professional work, and confidence in their abilities. But even novices can express themselves through programming in Scratch by creating simple stories and by importing their own content like music, images, and voice recordings (Resnick et al., 2009).

Finally, programmers have the ability to ask questions about technology and with technology. Rather than taking technology for granted, programmers can use computation to make sense of how technologies work, their limitations, and how to improve them in response to real-world situations (Brennan & Resnick, 2012). For novices, this can start out as a disposition towards wondering how things work. Or it can develop over time as programmers realize their abilities to adapt technologies for their own and others' needs. Questioning empowers computational thinkers to modify technologies, consider the affordances and constraints of existing tools, and discuss the impacts of technology on the world (CSTA, 2013; 2017).

Learning CT Concepts and Practices in K-12 Computer Science

In this paper, I am conceptualizing learning as a change in participation in a community of practice that occurs through interactions with the tools, practices, and participants in a community. But that conceptualization is not how researchers in CS always talk about learning. Early ideas of CT came from suggestions that while learning to program computers, students also learn powerful thinking skills applicable to broader problems (Papert, 1980; Nickerson, 1983). In the 1980s, many researchers interested in programming education claimed to also engage children in general problem-solving skills, supported by qualitative analyses and case studies (e.g. Gorman, Jr. & Bourne, Jr., 1983; Papert, 1980; Soloway, 1986; Yelland, 1995). However, quantitative studies on the cognitive effects of learning programming languages, including problem solving, creativity, logical reasoning, and more, showed mixed results (Gorman, Jr. & Bourne, Jr., 1983; Kalelioglu & Gülbahar, 2014; Pea, 1983; Pea & Kurland, 1984; Pea, Kurland, & Hawkins, 1985; Swan, 1989). Most of the changes to students' thinking skills appeared when the skills were assessed in a near transfer task or were closely related to the specific programming language students learned (Clements & Gullo, 1984; Mayer, Dyck, & Vilberg, 1986; Midian Kurland, Pea, Clement, & Mawby, 1986). This is not surprising from a situative perspective in which knowledge and understanding are fundamentally tied to the context and practices in which people participate. Furthermore, much of this work studied the effects of learning a specific programming language, and results suggest that general problem solving and thinking practices may not spontaneously arise from learning a programming language on its own (Pea, 1983). This finding confirms work in mathematics education that specifies that skills (in this case, being able to use a particular programming language) and practices (in this case, CT practices) are not the same thing (Greeno, 1991).

In contrast, recent work on CT favors instruction on general thinking skills in the context of programming or other disciplines in order to develop computational thinkers who can solve problems in different contexts. In other words, CT emphasizes the importance of learning practices while solving problems in different environments rather than learning a programming language and hoping the practices arise. In this sense, the act of programming is a useful tool for supporting engagement in CT (Grover & Pea, 2013). Programming affords opportunities for children to think about their own thinking because they "must make processes explicit in order to teach the computer how to perform a given task" (Cejka, Rogers, & Portsmore, 2006, p. 712). For example, research on creating digital games through programming demonstrated positive effects on motivation, creativity, problem solving, and critical thinking (e.g. Carolyn Yang & Chang, 2013).

The value of learning programming, beyond preparing a diverse workforce in computing, comes from empowering children to create their own solutions and use CT skills to solve personally meaningful problems. This sounds promising in theory, but what do students actually learn by engaging in CT through programming? And what do we know about students' identity development in relation to CT when engaged in the context of programming? Much of the work on CT has focused on designing learning environments to support CT concepts and practices. This paper asks how these designs can support the apprenticeship of people into the CT community using LPP.

Studies of educational programming environments have shown that even young newcomers to the CT community can engage in CT practices. Tangible programming environments, which use physical (rather than digital) blocks to code, have shown promise for preschool and kindergarten-aged children. These environments allow learners to easily create a functioning project with little introduction time (Bers, 2010; Horn & Jacob, 2007; Kelleher & Pausch, 2005; Wang, Wang, & Liu, 2014; Wyeth & Purchase, 2002). Programming languages with simple syntactical structures give newcomers immediate access to legitimate concepts and skills (Resnick et al., 2009), creating a space for learning through LPP.

For example, 5-9 year old children using T-Maze, a maze-building and puzzle-solving programming environment using physical blocks, were able to use the CT practices of abstraction, automation, and problem decomposition (Wang, Wang, & Liu, 2014). Children understood the relationship between physical blocks of code and virtual squares in the maze on the screen, a form of abstracting information. From evidence in students' talk, researchers concluded that students realized the computer automated their programs in the virtual space by executing the instructions they created using the physical blocks. In terms of concepts, students learned to create sequential instructions, and researchers later introduced the concept of loops to children. Additionally, there was some evidence of problem decomposition, such as when a student separated the maze problem into two sub-problems: moving forward and turning. While tangible programming is promising for introducing CT to young children, it is not clear how learners who start with tangible programming move towards becoming fuller participants in the community over time. We need to consider how to connect CT skills across different educational tools so students can build on their CT learning as they use more advanced programming environments, like professional text-based languages.

Studies with middle school students using visual programming tools (digital block-based environments) have also demonstrated some successful CT learning. For example, Storytelling Alice is a mixture of block-based and text-based programming that allows learners to create detailed stories and games. As early as fifth grade, students using Storytelling Alice can apply the CT concepts of loops, conditionals, sequences of instructions, variables, and data types (Kelleher & Pausch, 2007). Even within two hours of using the tool in one study, all students were able to create a working sequential program, and some used loops and variables (Kelleher & Pausch, 2007).

Other research on Scratch, a block-based programming tool, has shown that learners of all ages and experience levels can engage in almost all of the CT concepts and practices discussed above (Brennan & Resnick, 2012). Using the Scratch online community (https://scratch.mit.edu/), novices can develop a broad picture of what the community does and where their learning might take them as they participate more fully through LPP. These different roles and opportunities to engage leave open questions about how the variations affect student learning. In what ways do learners use CT in different roles within the programming community (e.g. people who remix work might learn different concepts and skills than those who always make their own projects)? One study looked at amount of participation in the Scratch online community and found no correlation between level of involvement (including a mixture of downloading projects, commenting, remixing, and friending other users) and types of CT concepts used in the users' projects (Fields, Giang, &

Kafai, 2014). Not much, if any, work has been done on a closer level to understand the relationship between different ways of participating in the community and students' CT learning.

Remixing also plays a role in learning. Through remixing others' projects, newcomers are exposed to different ways of solving problems and can see strategies used by old-timers in the community. Learners can also practice their own skills in a low-stakes environment by starting with projects that already work and building off them to add new code. In Scratch, the remixed projects are automatically saved in a new file, so any changes do not affect the original creator's work, which lowers the pressure for newcomers to produce accurate and efficient work. Not surprisingly, the more learners remix others' code in Scratch, the more CT concepts they use in their own projects (Dasgupta et al., 2016).

At the same time, a study of over 5,000 users in the Scratch online community found that length of membership in the programming community does not always predict the amount of programming concepts used (Fields, Giang, & Kafai, 2014). In other words, some people with less than one year of experience with Scratch used just as many CT concepts, like loops, conditionals, variables, and Booleans, as old-timers with years of experience in Scratch. However, most girls in the study remained at the beginner level in Scratch, only using simple loops in their projects, while more boys created projects using several different CT concepts.

To summarize, work on learning CT in programming contexts demonstrates that children as young as five can learn CT concepts, and visual programming tools work well to introduce CT concepts to newcomers in middle school grades and above. Visual tools allow newcomers to quickly participate in legitimate ways by creating new projects without memorizing complicated syntax. Online communities, like Scratch, support a variety of types of participation. Within those communities, learners can participate on the periphery by remixing existing projects, or they can engage in other legitimate types of participation by commenting, sharing, and "friending" other users. Regardless of the level of participation, learners in Scratch use many CT concepts. In fact, Scratch has been shown to support a variety of learners to participate in CT concepts, practices, and dispositions.

The studies reviewed above, focusing on learning CT through CS, include a mixture of in-school and out-of-school contexts. However, the in-school studies do not explain how the activities were integrated into the classroom system. To implement CT instruction in ways that encourage diversity and meaningful learning, we need a better understanding of the role of the teacher, the integration of CT activities into the discipline of the overall course, and classroom norms. Additionally, most of this work on CT learning has focused on CT concepts rather than practices or dispositions. The next section discusses some connections to CT dispositions and identity development, but more research needs to explore how learners participate in CT practices and how their participation changes over time as they become fuller members of the community (or, in many cases, choose to distance themselves from the CS community).

From research on learning CT through programming, we see that there are different ways learners can legitimately engage in CT practices and use CT concepts. There are many roles for computational thinkers within the programming community, including creating algorithms, debugging projects, and managing others' work. These variations in roles and ways of participating in the CT community invite questions about identity development. For instance, how do different types of programming projects affect students' identity development in relation to CT, especially in relation to students' prior interests and experiences? Some scholars have used storytelling to capture girls' interests in programming (e.g. Kelleher, 2009; Pinkard, Erete, Martin, & McKiney de Royston, 2017), but it is not clear whether and how those initial learning experiences lead to long-term identity development and productive relationships with computing. Additionally, it is not clear how participating in different roles while learning CT through programming supports identity and persistence in the field. Can we change students' views of what computational thinkers do and broaden participation in computing by exposing students to the variety of meaningful and legitimate ways to participate? This is an open question in the field. Maybe someone who dislikes creating procedures but enjoys testing and looking for errors in others' work will be surprised to learn that finding errors still involves CT skills, and then learn to recognize themselves as a computational thinker.

CT Identity Development and Dispositions in K-12 Computer Science

I conceptualize identity in this paper as the development of dispositions or regularities in the ways people participate in practices and their views of those practices and themselves in relation to a community. To date, little research has focused on identity towards CT as it is conceptualized here, but some studies have explored students' interests and motivations in CT learning environments. I draw on the literature on interest and motivation here because it influences people's views of the discipline and their views of themselves, both part of the framework of identity from LPP. Specifically, interest and motivation have been shown to lead to meaningful engagement and increased persistence, which are related to feelings of competence, productive disciplinary relationships, and productive sense of self in relation to the discipline (Kaplan & Flum, 2009; McCaslin, 2009; Potvin & Hasni, 2014; Renninger, 2009; Waterman, 2004; Wigfield & Wagner, 2005). Having a reason for learning CT and a social context for using it are important for motivating students and addressing sociological barriers to learning (Kelleher & Pausch, 2005). By seeing CT as a tool for accomplishing their own goals, students have agency over what they create and how they engage with computers in and beyond the classroom. The goal is to create positive experiences and to support students to feel a sense of belonging in the community by exploring factors related to identity and dispositions.

Several design characteristics of CS learning environments have been shown to support the development of productive elements of identity, including communities, mentors and role models, collaborative work, and programming contexts like stories or games. First, as an example of community, the Scratch online community is a place where people can engage in CT in many different ways. It is a community of learners and practitioners of all different levels of experience, making it an important resource for learning and identity development through LPP. When users create projects, post online, comment on others' work, or remix existing projects, they're engaging in a form of apprenticeship learning by participating in the community and interacting with other, more experienced practitioners. Members of the Scratch community can take on different roles and participate in different ways, depending on what they are interested in and where their prior experiences take them. These choices for participation give students some agency over their own engagement and allow students to define their roles while still acting as legitimate members of the community. However, more research is needed to understand how these different roles affect what students learn. Someone who spends more time commenting on and critiquing projects would most likely develop different skills and understandings from someone who creates projects but doesn't engage in commenting. It seems likely that K-12 educators would want students to explore all the roles to support the development of different concepts and practices, but it is also important that educators value students' interests and preferences for participation so students can participate in legitimate roles in the learning community.

Highlighting the social nature of computing in environments like Scratch, with its online community, can create positive, gender-inclusive educational experiences for newcomers (Mark, 1992; Resnick et al., 2009). Even interactions with fictional characters can support positive disciplinary relationships. For instance, in Digital Youth Divas, researchers designed characters to imitate actual middle school girls with a variety of interests, body types, and stories (Pinkard, Erete, Martin, & McKiney de Royston, 2017). The relatable characters and situations offer ideational resources to support girls' identification with CT. Researchers found the narratives motivated girls to work on projects, while the characters provided a community of relatable (but fictional) girls interested in STEM, igniting and confirming students' own interests in STEM fields (Pinkard et al., 2017). The study did not report on participants' use of CT concepts and practices. But the results did point to several factors related to identity and dispositions, including girls' increased interests in working on the projects, opportunities to exercise agency in design challenges, and changes in views of themselves in relation to STEM.

Teaching CT through programming stories, dances, and games are other popular ways to connect to students' interests and create gender-inclusive learning environments. One study using the language Alice to program characters to perform a dance showed increased motivation for some girls in the class, but it's not clear what they learned about programming or CT (Daily et al., 2014). Other research has shown that storytelling is a particularly interesting context for girls learning to program and can increase the time they spend persisting on a programming project (Kelleher, 2009; Kelleher & Pausch, 2007). Programming stories also allows learners to develop productive CT dispositions. Students can express themselves through their stories, developing a disposition towards using computational tools for expression. Students can also use computational tools to connect with others, by sharing their stories with friends, family, and their classroom peers, to develop productive dispositions for collaboration.

Along with opportunities to share projects, the Digital Youth Divas program structures in-person mentorship and conversations with peers into the curriculum (Pinkard et al., 2017). Students talk informally while working on their projects, but they also participate in structured check-ins with mentors at the beginning of each lesson. Mentors share cultural connections with students and encourage engagement, goal setting, and communication. Digital materials built into the narrative environment mediate discussions and relationship building. Conversations with mentors and fictional characters in the online narrative encourage students to reflect on their STEM experiences and racial and social issues related to STEM. Researchers point to a potential link between these on- and offline conversations with a sense of connection and positive engagement within a classroom STEM learning community (Pinkard et al., 2017). Future research should explore the role of this personal mentorship in CT for K-12 students. How can teachers incorporate mentors into their classrooms? It might be difficult for some teachers, particularly those in rural communities, to have access to mentors and role models with professional experience in CT. In those instances, mentors' close social and cultural connections with students would be particularly important. Virtual mentorships or even pen pals might be a way of supporting students who don't have access to in-person role models. How long do mentorship experiences need to last, and how can they be implemented in ways that truly affect students' views of themselves and interests in persisting in CT?

Besides mentorship, pair programming is another common in-person collaborative programming task, building the CT disposition for collaboration and communication. Pair programming involves two people working at a computer at the same time. One person acts as the "driver" by typing at the computer while the other acts as the "navigator" by observing and critiquing the driver. Both participants collaborate on solving problems and often switch roles (Williams et al., 2002). While making programming a more collaborative experience, pair programming also gives authority to students as they solve their own problems. This type of interaction allows students to learn from differences in each other's knowledge and experiences. Rather than asking an expert for the answer, students can work with their peers to solve a problem or search for answers from other sources, like online forums. Both are valuable skills in professional computational work, so it legitimizes students' roles as computational problem solvers. Pair programming in classroom settings apprentices students into the act of pair programming that also occurs in professional settings. By working together to develop solutions, students can see themselves as capable of participating in the CT community. This is in contrast with expecting an answer from an expert TA, which positions students as less competent and less qualified to participate legitimately in the community of practitioners. Most pair programming research focuses on college-level courses, but how successful is it for middle or high school students? Intentionally pairing students with different kinds of knowledge or experiences in programming might help them collaborate and learn from each other, but what happens in an introductory course when students have no prior computing experience? What is the role of the instructor in supporting productive collaborations, and what should the instructor do when students struggle? These are all important questions to think about if pair programming is going to support a variety of learners in K-12 settings.

Non-stereotypical approaches to programming instruction, like the storytelling and dance examples here, can potentially reduce gender disparities in computational subjects by sparking girls' interests. The Digital Youth Divas program also demonstrated the value of role models, whether real or fictional, for interest and personal identity construction (Pinkard et al., 2017). Additionally, a recent study found that even reducing stereotypical objects in computing classrooms (e.g. replacing Star Wars posters, electronic parts, and tech magazines with art, plants, and general magazines) can increase girls' sense of belonging and interest in a high school computing course without lowering boys' existing interests (Master, Cheryan, & Meltzoff, 2016). Non-stereotypical learning environments, activities, and role models – all designed to minimize and challenge stereotypes - affect students' sense of belonging in CT contexts (Cheryan, Master, & Meltzoff, 2015).

Much of this work focuses on motivation and interest, particularly for girls, but it is not clear how these different approaches affect long-term identity development and persistence in CT. There is also a lack of research on marginalized racial groups in computing. Future work should consider the intersection of race, gender, and other institutional factors that influence students' experiences and identities both inside and outside the classroom. To truly break away from stereotypical views of computational thinkers, researchers need to look even beyond programming to see how learners can engage in CT in other contexts.

Computational Thinking in Engineering

According to the Royal Academy of Engineering, engineering covers many different industries, from buildings to food to medicine, and it involves making things work and designing solutions to meet the needs of society (Brophy, Klein, Portsmore, & Rogers, 2008; Royal Academy of Engineering, 2017). Some researchers see CT as a way of thinking that creates a bridge between computer science and engineering (NRC, 2010). CT "inherently draws on engineering, given that [computer scientists] build systems that interact with the real world" (Wing, 2006, p. 35). Computational thinking and engineering both involve solving problems and making things (Wing, 2008), but engineering is inherently constrained by the physical world in ways that CT is not (Shute, Sun, Asbell-Clarke, 2017; Wing, 2010). Engineering design thinking "focuses on product specification and the requirements imposed by both the human and the environment—i.e., practical problems. CT is not always limited by physical constraints, enabling people to solve theoretical as well as practical problems" (Shute, Sun, Asbell-Clarke, 2017, p. 8). This distinction is meant to highlight the idea that people can think computationally about problems in imaginative ways without being tied to rules of the physical world, while engineers ultimately aim to implement their ideas in the physical world so their work cannot be separated from those constraints. However, when people use CT to create procedures for a computational agent to carry out, whether that agent is a mechanical computer or the human brain, they must consider the capabilities of the agent. As an example of someone considering the limitations and capabilities of a computational agent, a programmer using Scratch could not make a word processing software or a website, but they could make a story or game. If an important part of CT is considering and testing solutions with a computational agent, then it does overlap with engineering in its consideration of the physical and digital world.

Educators and policymakers are starting to recognize these connections between designing in CT and engineering. The NGSS now include a CT progression within their K-12 engineering standards (NGSS, 2013). Additionally, research already shows that engaging students in design is useful for learning. Design-based activities can help learners develop deep conceptual understandings and inquiry skills (Crismond, 2001; Kimmel et al., 2006; Kolodner et al., 2003; Roth, 1995; Sadler, Barab, & Scott, 2007). To better understand the role of CT in design, the next section draws on LPP to explore how engineers use CT concepts, practices, and dispositions in their work. The discussion also mentions some ways in which learners can access those skills through engineering education.

Concepts

In specifying the concepts from CT, researchers looked to programming instead of the broad field of computer science with its many different domains of knowledge. Similarly, it is difficult to list the specific concepts involved in engineering because it encompasses many different sub-fields with their own core concepts. Engineers use concepts from across STEM disciplines, with programming included as one of the areas some engineers may draw on (Brophy, Klein, Portsmore, & Rogers, 2008; Royal Academy of Engineering, 2017). In particular, software engineering involves computer programming in the development and maintenance of computer software. Thus, software engineers use the CT concepts from programming described above, but being an expert engineer may involve knowing many other STEM concepts outside the scope of CT.

However, engineers do use CT concepts related to data collection, organization, and representation. When testing different designs, engineers collect data to determine the best option that meets the constraints of the problem (NRC, 2012). Both engineers and computer scientists use technology to collect and interpret data. They must understand how to collect the data, how to use the appropriate tools, how to appropriately organize the data, and how to interpret the results.

Since this paper focuses on how people use CT in different disciplines, it is outside the scope of this work to spend time describing all the other concepts used by engineers. Instead, the rest of this section looks at broader processes of design and problem solving used across engineering fields to generate an overview of CT practices and dispositions in the context of general engineering skills, particularly those specified in learning standards and curricula.

Practices

Central to the work of engineers is the Engineering Design Process (Haik, Sivaloganathan, & Shahin, 2015). This is the iterative process engineers use to design artifacts based on specific needs or goals (NASA's Best, 2016). It is a cyclical process that includes identifying a problem or asking a question, imagining a solution, designing a prototype, testing the designs, and improving the solution (EiE, 2017a; NASA's Best, 2016; NGSS, 2013). One CT practice from programming involves generalizing solutions into a problem solving process that can be applied to a variety of problems (Barr, Harrison, & Conery, 2011; Hu, 2011), which is what the Engineering Design Process already is. It is a way of solving problems that engineers can draw on in any situation. Ultimately, CT is also about designing solutions to problems, and each of these elements of the Engineering Design Process overlap with other CT practices used in creating computer programs.

First, engineers start by identifying a problem or question they want to address. In both professional and educational environments, the problem may be defined by the engineer or may be assigned by another person, like a manager, funder, or teacher. In any case, the engineer must work to understand the constraints of the situation and learn about how others have approached similar problems (EiE, 2017a). There may be limitations in the materials that can be used, the number of prototypes that can be tested, and the timeframe for completing the project. The goal in this phase is to ask questions of a client and about prior approaches to similar problems to understand the problem in as much detail as possible (NASA's Best, 2016). The same can be true about solving programming problems using CT, although there are usually fewer physical constraints to consider (Shute, Sun, Asbell-Clarke, 2017). Both novice and expert programmers must identify a problem and define their goals at the beginning of the computational problem-solving process. In programming, some of this work may involve reworking the problem into one that can be solved by a computer (ISTE & CSTA, 2011; Wing, 2006), while in engineering, it may involve reworking the problem into one that can be solved with the available materials and within current technological capabilities.

The second and third parts of the design process involve imagining a solution and implementing the solution by creating a model or prototype. These are the processes of building something in engineering or writing a procedure in programming. While creating designs, both programmers and engineers have to consider the efficiency of their solutions, a practice many label as part of CT (Barr, Harrison, & Conery, 2011; Grover & Pea, 2013; Wing, 2008). Engineers have to consider limitations on materials, cost, and time to both build and work efficiently. Another CT practice both engineers and programmers use in design is reusing others' work (Brennan & Resnick, 2012). To develop ideas, engineers can draw on previous attempts to solve the same problem or existing solutions from related problems. To engage in this practice, engineering students can give feedback and suggest ideas to their peers, and they can investigate related designs created by more experienced engineers. For instance, in a bridge-design task, students do not have to start from scratch but can look to real-world bridges for ideas about materials, functionality, and strength.

The final two elements of the Engineering Design Process are testing and refining designs. Engineers may rely on models or simulations when testing designs. Both types of abstractions are also considered part of CT (Grover & Pea, 2013; Hu, 2011; NRC, 2010; Wing, 2006; ISTE & CSTA, 2011). Additionally, engineers may work with data representations to organize the outcomes of their trials, another practice in CT (Barr, Harrison, & Conery, 2011; Grover & Pea, 2013; Hu, 2011; NRC, 2010). But the central component of testing and refining in engineering, like in CT, is the debugging process (Bers et al., 2014; Brennan & Resnick, 2012; Grover & Pea, 2013). Engineers debug their prototypes by finding and fixing errors and preparing them for further testing. This practice drives the iterative nature of the design process. Debugging also offers opportunities for productive struggle and failure, which have shown to help students develop metacognitive skills and perform better on other open-ended problem solving tasks (Bullmaster-day, 2015; Hung, Chen, & Lim, 2009; Kapur, 2008). Specifically, "the steps of testing and improving, which require debugging, are particularly important in establishing a learning environment where failure -- rather than immediate success -- is expected and seen as necessary for learning. With the Engineering Design Process, children are not expected to 'get it right' the first time" (Bers et al., 2014, p. 149). Debugging allows students to get things wrong but still legitimately participate in CT and engineering. In fact, testing solutions gives students a space to tinker by building things on the fringes of professional engineering while also apprenticing into a core practice of the engineering and CT communities. In other words, the processes of testing and refining allow learners to participate both legitimately and peripherally in CT and engineering disciplines (Lave & Wenger, 1991).

Engineers engage in the practice of reusing and remixing others' work when troubleshooting or reverse engineering existing designs. "Troubleshooting and reverse engineering require investigating someone else's designs to either repair it, replicate it, or refine it" (Brophy et al., 2008, p. 375). Engineering students engaging in this process should evaluate the quality of an existing product by analyzing the original designer's intentions and constraints.

To expose learners to the practices of the Engineering Design Process, curricula are usually created to move students systematically through all phases of the process (EiE, 2017a). However, professional engineers may work within a couple of the phases, and then pass their work onto other engineers to continue the process. Thus, the work becomes more specialized as engineers take on different roles within the community. The phases themselves are flexible and can be completed in different orders and in multiple ways. When considering connections between engineering and CT, engineers in different roles will use different CT practices in their work depending on how they use the Engineering Design Process. In other words, it makes sense that engineering students may use some CT practices but not others. Educators want to expose novices to all the core practices used by the engineering community, but they should also consider the different ways of legitimately acting as an engineer. Students can still be competent computational thinkers even if they do not make use of all the CT practices in their work. If a student does not like the debugging or testing process, they should not be discouraged from being an engineer or computational thinker. Instead, educators should demonstrate that there are other ways of legitimately participating. Students could specialize in defining problems or creating solutions and still have important roles as computational thinkers in the engineering (or programming) communities.

Even young children can engage in planning, making, and evaluating their solutions in design-based engineering activities (Fleer, 1999; 2000). In Fleer's study, preschool children were given an open-ended task to design a home for a mythical creature the teacher imagined living in her garden. Young children often begin these activities with an unspecified design goal that emerges as they build things (Brophy et al., 2008; Johnsey, 1995). By second grade however, students who have been engaging in design processes for several years are able to plan their designs by considering materials and constraints of the task (Roden, 1999). This work demonstrates that it is reasonable for novices to engage in making and testing practices first, since “the natural cycle of iterative design places students in a continuous cycle of test and evaluation” (Brophy et al., 2008, p. 373). After gaining some experience with the design cycle, then learners can practice planning their designs and specifying their goals ahead of time. Content knowledge also seems to affect the number of iterations of the design cycle. Experts have more prior knowledge and experiences to draw on when planning their designs, so they are more likely than novices to come up with an accurate plan the first time (Roth, 1996; Wineburg, 1991). However, like with debugging programs, even experts are expected to find errors and make changes through cycles of design.

Dispositions

One of the dispositions Brennan and Resnick (2012) identified as important to learning CT in programming is the ability to deal with open-ended problems. Similarly, designing solutions to open-ended problems is central to the work of engineering. “Design and troubleshooting represent the types of ill-structured, or open-ended, problems on which engineers enjoy spending intellectual energy” (Brophy et al., 2008, p. 371). Engineers serving different roles in the design cycle have to respond to open-ended problems in different ways. Some may focus on planning and brainstorming solutions, while others may focus on testing and debugging solutions. Like computer scientists, engineers must welcome open-ended problems as a challenge and persist in solving them. But this raises a question about transferability. If students develop the disposition to persist on engineering problems, will they also persist on open-ended problems in CS and other disciplines? The disposition may start out as context-specific, but as it becomes part of learners’ identities over time, they may be able to use similar approaches to problems in different contexts. Longitudinal studies are needed to investigate the construction of dispositions over years of learning and identity development.

Questioning is another CT disposition from programming that overlaps with the core of engineering. The goal of engineering is to address social needs and solve problems through design. Rather than taking existing tools and technologies as given, engineers ask how they can improve and re-conceptualize those tools to solve new problems and improve solutions to old problems (NGSS, 2013). They also use technologies as part of the design process, to model situations and test solutions. Thus, engineers ask questions both about and with technologies. More research is needed to understand whether and how students learning CS and engineering develop these questioning mindsets. Is it a disposition that all students develop when they see they can create new things with technology, or are some students more apt to look at technology in this way than others are? It seems like the latter is more likely, since the disposition aligns with masculine forms of competence and stereotypes of makers that enjoy taking things apart to see how they work. However, this broader view of questioning in CT involves asking not only how technologies work but also what new technologies we can create.

Stereotypes in both engineering and CS include visions of lonely individuals working on their own to solve problems. Earlier I described how the ability and willingness to collaborate with others is actually an important mindset of computational thinkers in CS, and the same can be said in engineering. Learning to collaborate with others is built into K-12 engineering education standards and curricula (EiE, 2017b; NRC, 2012; NGSS, 2013). Collaboration and communication with others are also considered engineering "habits of mind" or attitudes associated with engineering (NAE & NRC, 2009). Multiple engineers often work on the same problem by designing and testing different ideas, then collaborating to choose the most promising solution (NRC, 2012). Engineers must learn to evaluate and compare each other's ideas and formulate arguments based on data and testing. They also need to communicate their ideas clearly so their solutions can be understood by outside clients as well as engineers serving other roles in the design process (NRC, 2012).

Learning CT Practices in K-12 Engineering

Learning engineering in K-12 and its integration with other STEM disciplines is understudied (Moore et al., 2014; Rogers, Wendell, & Foster, 2010), including the idea of learning CT through engineering. K-12 engineering education is still quite new and not widely implemented in the U.S. (NAE & NRC, 2009). In the curricula that have been developed for K-12 engineering, the content centers on design (NAE & NRC, 2009). Likewise, the previous section outlining the connections between CT and engineering demonstrates that most of the overlaps occur in the engineering design process. So what do we know about learning the engineering design process in K-12? We know very little about it, actually. Research in engineering education tends to focus on the presentation of educational tools or curricula, or on identity development. Very few engineering education studies have focused on students' understandings of concepts and practices. Researchers suggest that engineering learning occurs best when students have extended time to design and iterate on projects (Rogers, Wendell, & Foster, 2010) and when tools (e.g. software, computational tools) are meaningfully integrated into problem-solving activities (NAE & NRC, 2009). However, there is little empirical evidence to back those claims.

The idea of learning CT through engineering is a gap in the literature and an important space for future exploration. The small amount of research that exists occurs in the context of e-textiles, and that work is framed as learning CT through craft rather than engineering (Kafai et al., 2010; 2013; 2014; Kafai, Searle, Martinez, & Brayboy, 2014; Fields, Searle, & Kafai, 2016; Lui et al., 2016; Rode et al., 2015; Searle, Fields, Lui, & Kafai, 2014). I touch on the literature briefly here because it is a form of engineering; engineers use science concepts to design solutions to problems, and e-textiles projects draw on circuitry and materials science concepts through design.

E-textiles allow makers to incorporate electronic hardware (e.g. lights, sensors, microcomputers, and buzzers) into fabric designs. One study using e-textiles with high school students showed that students used several CT concepts and practices in their work, including sequences, conditionals, loops, variables, remixing, and debugging (Kafai et al., 2014). However, students in that study programmed e-textile projects using Arduino code. Thus, the CT skills students' employed largely occurred in the context of programming, with the exception of debugging, which students engaged in throughout the design process. Little is known about how students use CT concepts and practices in engineering activities without computer programming and how students' participation changes over time as they become members of the engineering community. Thus, an open question is, how might students engage in CT in ways that are legitimate to the engineering community and thus support students to learn through meaningful participation?

CT Identity Development and Dispositions in K-12 Engineering

While few studies of engineering have deeply considered learning, more have focused on students' identities in relation to engineering. Much of the work on engineering-related identities has studied university level engineering students and their persistence in engineering occupations (e.g. McGee & Martin, 2011; Meyers et al., 2012; Pierrakos et al., 2009; Tate & Linn, 2005), or on professional identities of working engineers (e.g. Anderson et al., 2010; Hatmaker, 2013; Jorgenson, 2002). On the college level, sense of belonging and recognition affect students' identification with engineering (Meyers et al., 2012). Additionally, university students who persist in engineering majors tend to have more knowledge of the profession, greater exposure to engineering (e.g. through family members or friends), and some productive relationships with engineering faculty and peers (Pierrakos et al., 2009). Persistence is also influenced by the intersection of academic and social identities, illustrated in studies focusing on the roles of gender and race in engineering programs (Tate & Linn, 2005). Although this research on university and professional engineers is a helpful starting point in response to pipeline issues, researchers need a better understanding of how K-12 engineering education affects students' views of engineering, development of productive dispositions, and sense of self in relation to engineering.

As an example of how design activities can affect high school students' views and identities, work on learning CT with e-textiles in high school classrooms demonstrates that alternative ways of approaching CT can change students' perceptions of computing and their views of themselves in relation to computing (Kafai et al., 2013). After making projects using programmable e-textiles materials, high school students saw CS as more relevant to their lives, gained confidence in their programming skills, and developed better understandings of what the computing field involves (Kafai et al., 2014). Furthermore, e-textiles activities have been shown to engage all students, regardless of race or gender, in CT (Kafai et al., 2013; 2014). While this work connects to engineering design processes, it still explicitly engages students in CT through computer programming. Questions remain about how students use CT in engineering contexts without programming, and how other engineering activities affect students' perceptions of CT and technology fields.

Other studies in high school and university engineering have drawn explicitly on LPP to study aspects of identity development. In one study using LPP as a framework to look at university engineering students' engagement in industrial vocation work, Jawitz, Case, and Ahmed (2005) found that opportunities to participate legitimately in meaningful activity influenced students' sense of belonging and views of themselves in relation to engineering. Not surprisingly, the mentoring or supervising engineers significantly influenced access to meaningful activities, and they affected each student's sense of self by advocating for or against the student's role as a legitimate participant. In another study looking at mentorship in a high school context, researchers demonstrated that communities of practice are essential for supporting persistence in science and engineering fields through mentorship and role models (Aschbacher, Li, & Roth, 2010). Along with outside mentors, K-12 teachers have significant influence over students' identity development and learning in their roles as mentors and supervisors. Thus, it is imperative that researchers take into account the role of the teacher in facilitating legitimate participation for all learners to develop productive identities as computational thinkers.

While mentors clearly influence learning and identity, more research is needed to understand how to implement mentorship communities that support productive engagement and sense of belonging for students even before they reach high school. In general, few studies have focused on learning and identity development in engineering with elementary and middle school students (Capobianco, Diefes-Dux, Mena, & Weller, 2011), which is not surprising given the lack of emphasis on formal engineering instruction for young children.

Discussion

The goal of this paper was to develop a better understanding of the concepts, practices, and dispositions involved in CT and how people can learn it by looking at how CT is defined in CS then exploring the overlaps with another context, namely engineering. I chose engineering because of its ties to other STEM content areas and the ability to practically apply STEM content, including CS, through engineering design problems, along with the fact that CT is beginning to appear in K-12 engineering education standards. Engineering offers an opportunity to understand how people use CT in connection with other STEM disciplines that do not necessarily involve mechanical computers or computer programming.

From my review of literature defining CT in CS contexts, I identified common CT concepts, practices, and dispositions that overlap with CS. Then I explored how those elements of CT overlap with the literature on design processes in engineering. First, CT concepts that overlap with both CS and engineering include: data collection, organization, and representation (many other concepts from CS and programming are traditionally included in CT but do not necessarily overlap with different engineering fields). Second, CT practices that overlap with both CS and engineering include: (i) generalizing solutions into a problem solving process, (ii) reworking the problem so it can be solved by a computational agent, (iii) considering efficiency and performance constraints, (iv) reusing or remixing others' work, (v) creating and using abstractions, and (vi) debugging and testing solutions. Finally, CT dispositions that overlap with both CS and engineering include: (i) dealing with open-ended problems, (ii) questioning about and with computational tools, and (iii) collaborating and communicating with others.

Much of the overlap between CT as it is defined in CS and its application in engineering can be seen in the engineering design process. So this begs the question, is CT really just design thinking or problem solving? The answer to this question is not completely clear from the current literature on CT. Without a clear, agreed-upon vision of what we want students to learn about CT, it's hard to articulate what those differences really are. The recent introduction of the term "computational making" (Rode et al., 2015), with ties to maker spaces and the maker movement, shifts CT even more in the direction of design and creation. Given the concepts, practices, and dispositions explored here, it seems like CT might be a specific form of problem solving or design, with some specific concepts that come from CS and programming. It could be that CT adds logical thinking and data use concepts to traditional design practices and dispositions. In other words, CT seems to be about logical thinking (many of the concepts from CS) plus design practices. CT may potentially be a useful combination of concepts, practices, and dispositions that prepares students for jobs across fields involving design and problem solving. But these and related claims about CT's ability to empower children to solve problems (e.g. ISTE & CSTA, 2011; Papert, 1980; Wing, 2006) are highly theoretical at this point, until more work can be done to define CT in use and distinguish it from other forms of thinking.

In general, more research on CT learning and identity development in K-12 contexts is needed. Most research on CT in K-12 has occurred in informal education settings (Lye & Koh, 2014). Given the recent development of CT in K-12 educational standards, such as in the CSTA and ISTE Computer Science Standards and in the Next Generation Science Standards, CT is clearly becoming part of formal K-12 education for all, not just an element of select after-school activities. Thus, researchers need to understand how to design for in-school learning environments and to productively incorporate CT into classrooms.

There has been some empirical work on learning CT in programming or CS contexts, but virtually nothing in K-12 engineering. Research in CS demonstrates that young children can learn CT concepts using visual and block-based programming tools, and online communities of practice support different forms of legitimate participation and roles within the community. Studies using Scratch in particular have demonstrated that students in a variety of grade levels can engage in almost all the CT concepts, practices, and dispositions listed in this paper. However, we don't know much about how these learning tools are integrated into K-12 classroom systems and how classrooms can support engagement in meaningful activities that continue to legitimize students' roles in communities that use CT. In the case of engineering, educators are just beginning to incorporate engineering in K-12 classrooms across the U.S., so there are few empirical studies on learning engineering in K-12 classrooms, let alone learning CT through engineering. There is work on learning CT with e-textiles, but those studies are framed as CT in the context of craft rather than engineering. Additionally, that work looks at CT learning through programming in Arduino, so it is still unclear how or what students learn about CT in engineering (or even craft) contexts that do not involve programming mechanical computers.

When it comes to identity development, relevant literature in both CS and engineering highlights the fact that CT education “is not just a matter of quantity but also one of quality of engagement” (Fields, Giang, & Kafai, 2014, p. 8). Researchers have contextualized problems in narratives, dance, and games to motivate students to participate in CT. Some studies have demonstrated students’ productive engagement in CT dispositions when programming stories or games, including expressing ideas, collaborating and communicating with others, and asking questions with computational tools. However, it is not clear how framing CT through these contexts affects long-term persistence, beliefs about CT, and students’ views of themselves in relation to CT.

Mentorship also plays a significant role in identity development through LPP in both CS and engineering contexts. Mentors shape students’ views about what CT is, who can participate in it, and whether they have access to legitimate roles within the community. Given the important role teachers play as mentors and supervisors, we need more research to understand how teachers can implement and be part of successful mentorship communities in K-12 settings to support meaningful participation for all students, not just those already represented by the majority of CS and engineering professionals.

While this paper illustrates that CT overlaps with design processes in disciplines other than CS, engineering is still a male-dominated profession. Only 14% of engineers in 2016 identified as female (The Economics Daily, 2017). To truly expand notions of competence and participation in CT for students who do not already match with the stereotypes in technology fields, this work needs to connect CT with contexts that are dominated by other groups of people. Therefore, I plan to explore CT in traditionally feminine contexts of craft in future work.

Conclusion

Research into CT in STEM and even humanities in K-12 contexts is just beginning to emerge. This is an important area for future work that has the potential to expand access to CT learning opportunities. It will also help refine the definition of CT and improve understandings of what CT looks like in different contexts. The way Deanna Kuhn described scientific thinking helps explain how researchers might expand the role of CT in K-12 education and our lives. Kuhn explained,

> Scientific thinking tends to be compartmentalized, viewed as relevant and accessible only to the narrow segment of the population who pursue scientific careers. If science education is to be successful, it is essential to counter this view and establish the place that scientific thinking has in the lives of all students. A typical approach to this objective has been to try to connect the *content* of science to phenomena familiar in students' everyday lives. An ultimately more powerful approach may be to connect the *process* of science to thinking processes that figure in ordinary people's lives (1993, p. 333).

By connecting scientific processes to the thinking processes in our everyday lives, it highlights the relevance of scientific thinking, points to the need to engage in the practice of thinking to enhance the quality of thinking, and makes social dialogue a place to externalize thinking strategies (Kuhn, 1993). In this view of thinking processes, it is okay, and even ideal, that CT overlaps with other processes, including design thinking, problem solving, critical thinking, systems thinking, and algorithmic thinking, because it connects the ways in which computer scientists think to other thinking processes people use in a variety of contexts. A focus on thinking processes demands work on the nature and role of CT in contexts outside of computing, with a variety of learners, and in everyday processes. It is still unclear what the role of computers in engaging in CT really is, and whether people can legitimately practice CT without mechanical computers (Weintrop et al., 2016). This work connecting CT to other contexts will advance the field towards a richer understanding of the concepts and practices collected under CT, many of which are not yet clearly defined, and the practical utility of CT as a construct within K-12 education.

REFERENCES

Anderson, K. J. B., Courter, S. S., McGlamery, T., Nathans-Kelly, T. M., & Nicometo, C. G. (2010). Understanding engineering work and identity: a cross-case analysis of engineers within six firms. *Engineering Studies*, *2*(3), 153-174.

Aschbacher, P. R., Li, E., & Roth, E. J. (2010). Is science me? High school students' identities, participation and aspirations in science, engineering, and medicine. *Journal of Research in Science Teaching*, *47*(5), 564–582.

Barr, B. D., Harrison, J., & Conery, L. (2011). Computational Thinking : A Digital Age. *Learning & Leading with Technology*, *5191*(March / April), 20–23.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12. *ACM Inroads*, *2*(1), 111–122.

Beckhusen, J. (2016). *Occupations in Information Technology: American Community Survey Reports* (Vol. 1980). Washington, D.C.: U.S. Census Bureau.

Bers, M. U. (2010). The TangibleK robotics program: Applied computational thinking for young children. *Early Childhood Research and Practice*, *12*(2), 1–20.

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

Biochemical Society (2017). *What is biochemistry?* Retrieved from http://www.biochemistry.org/?TabId=456.

Bishop, J. P. (2012). "She's Always Been the Smart One. I've Always Been the Dumb One": Identities in the Mathematics Classroom. *Journal for Research in Mathematics Education*, *43*(1), 34–74.

Boaler, J. (2002). The development of disciplinary relationships: knowledge, practice, and identity in mathematics classrooms. *For the Learning of Mathematics*, *22*(1), 42–47.

Boaler, J., & Greeno, J. G. (2000). Identity, agency, and knowing in mathematics worlds. *Multiple perspectives on mathematics teaching and learning*, 171-200.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *annual AERA meeting, Vancouver, BC, Canada* (pp. 1–25).

Brickhouse, N. W., & Potter, J. T. (2001). Young Women's Scientific Identity Formation in an Urban Context. *Journal of Research in Science Teaching*, *38*(8), 965–980.

Brikman, Y. (2014). *Don't learn to code. Learn to think.* [Blog post]. Retrieved from https://www.ybrikman.com/writing/2014/05/19/dont-learn-to-code-learn-to-think/.

Brophy, S., Klein, S., Portsmore, M., & Rogers, C. (2008). Advancing Engineering Education in P-12 Classrooms. *Journal of Engineering Education*, (July), 369–387.

Bullmaster-Day, M. L. (2015). *Productive Struggle for Deeper Learning*. Triumph Learning.

Bureau of Labor Statistics (2017). *Computer and information technology occupations*. Retrieved from https://www.bls.gov/ooh/computer-and-information-technology/home.htm.

Capobianco, B. M., Diefes-Dux, H. A., Mena, I., & Weller, J. (2011). What is an Engineer? Implications of Elementary School Student Conceptions for Engineering Education. *Journal of Engineering Education*, *100*(2), 304–328.

Carolyn Yang, Y. T., & Chang, C. H. (2013). Empowering students through digital game authorship: Enhancing concentration, critical thinking, and academic achievement. *Computers and Education*, *68*, 334–344.

Catterall, J. (2013). Getting real about the E in STEAM. *The STEAM Journal*, 1(1), Article 6.

Cejka, E., Rogers, C., & Portsmore, M. (2006). Kindergarten Robotics: Using Robotics to Motivate Math, Science, and Engineering Literacy in Elementary School. *International Journal of Engineering Education*, *22*(4), 711–722.

Cheryan, S., Master, A., & Meltzoff, A. N. (2015). Cultural stereotypes as gatekeepers: Increasing girls' interest in computer science and engineering by diversifying stereotypes. *Frontiers in Psychology*, *6*(FEB), 1–8.

Clements, D. H., & Gullo, D. F. (1984). Effects of Computer Programming on Young Children's Cognition. *Journal of Educational Psychology*, *76*(6), 1051–1058.

Cobb, P., & Bowers, J. (1999). Cognitive and situated learning perspectives in theory and practice. *Educational researcher*, *28*(2), 4-15.

Crismond, D. (2001). Learning and using science ideas when doing investigate-and-redesign tasks: A study of naive, novice, and expert designers doing constrained and scaffolded design work. *Journal of Research in Science Teaching,* 38(7), 791–820.

CSTA. (2013). *CSTA K-12 Computer Science Standards: Mapped to Common Core State Standards for Mathematical Practice*. Retrieved from http://www.csta.acm.org/Curriculum/sub/CurrFiles/CSTA_Standards_Mapped_to_CommonCoreStandards.pdf

CSTA. (2017). *CSTA K-12 Computer Science Standards*. Retrieved from http://www.csteachers.org/page/standards

CSTA & ACM (2016). *Interim CSTA K-12 computer science standards*. New York, NY: CSTA & ACM. Retrieved from https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/Docs/Standards/2016StandardsRevision/INTERIM_StandardsFINAL_07222.pdf.

Daily, S. B., Leonard, A. E., Jörg, S., Babu, S., & Gundersen, K. (2014). Dancing Alice: Exploring embodied pedagogical strategies for learning computational thinking. In *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14* (pp. 91–96). Atlanta, GA: ACM.

Dasgupta, S., Hale, W., Monroy- Hernandez, A., & Hill, B. M. (2016). Remixing as a Pathway to Computational Thinking. In *CSCW* (pp. 1438–1449). San Francisco, CA.

diSessa, A. (2000). *Changing Minds: Computers, Learning, and Literacy*. Cambridge, MA: MIT Press.

EiE (Engineering is Elementary) (2017a). *The engineering design process.* Retrieved from https://www.eie.org/overview/engineering-design-process.

EiE (Engineering is Elementary) (2017b). *Trajectories for preschool-middle school engineering activities.* Retrieved from https://eie.org/overview/engineering-trajectories.

Feldman, A. (2015). STEAM rising: Why we need to put the arts into STEM education. *Slate*. Retrieved from http://www.slate.com/articles/technology/future_tense/2015/06/steam_vs_stem_why_we_need_to_put_the_arts_into_stem_education.html.

Fields, D. A., Giang, M., & Kafai, Y. (2014). Programming in the wild. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education on - WiPSCE '14* (pp. 2–11). Berlin, Germany.

Fields, D. A., Searle, K. A., & Kafai, Y. B. (2016). Deconstruction Kits for Learning: Students ' Collaborative Debugging of Electronic Textile Designs. In *FabLearn* (pp. 82–85). Stanford, CA.

Fleer, M. (1999). Children's alternative views: Alternative to what? *International Journal of Science Education,* 21(2), 119–35.

Fleer, M. (2000). Working technologically: Investigations into how young children design and make during technology education. *International Journal of Technology and Design Education,* 10(1), 43–59.

Gee, J. P. (2000). Chapter 3: Identity as an analytic lens for research in education. *Review of research in education*, *25*(1), 99-125.

Gorman, H., & Bourne, L. E. (1983). Learning to think by learning LOGO: Rule learning in third-grade computer programmers. *Bulletin of the Psychonomic Society*, *21*(3), 165–167.

Greeno, J. G. (1991). Number sense as situated knowing in a conceptual domain. *Journal for Research in Mathematics Education*, *22*(3), 170–218.

Greeno, J. G., & Gresalfi, M. S. (2008). Opportunities to learn in practice and identity. In P. A. Moss, D. C. Pullin, J. P. Gee, E. H. Haertel, & L. J. Young (Eds.), *Assessment, Equity, and Opportunity to Learn*. New York: Cambridge University Press.

Gresalfi, M. S., & Cobb, P. (2006). Cultivating students' discipline-specific dispositions as a critical goal for pedagogy and equity. *Pedagogies, 1*(1), 49–57.

Gresalfi, M., Martin, T., Hand, V., & Greeno, J. (2009). Constructing competence: an analysis of student participation in the activity systems of mathematics classrooms. *Education Studies in Mathematics*, *70*(1), 49–70.

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. In S. Sentance, E. Barendsen, & C. Schulte (Eds.), *Computer Science Education: Perspectives on Teaching and Learning*. London: Bloomsbury.

Guzdial, M. (2008). Education: Paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25–27.

Haik, Y., Sivaloganathan, S., and Shahin, T.M. (2015). *Engineering Design Process* (3rd Ed.). Boston, MA: Cengage Learning.

Halpern, D. F. (1999). Teaching for critical thinking: Helping college students develop the skills and dispositions of a critical thinker. *New directions for teaching and learning*, *1999*(80), 69-74.

Hand, V., & Gresalfi, M. (2015). The Joint Accomplishment of Identity. *Educational Psychologist, 50*(3), 190–203.

Handelsman, J. and Smith, M. (2016, February 11). STEM for all [Blog post]. *The White House Blog*. Retrieved from https://obamawhitehouse.archives.gov/blog/2016/02/11/stem-all.

Hatmaker, D. M. (2013). Engineering identity: Gender and professional identity negotiation among women engineers. *Gender, Work & Organization*, *20*(4), 382-396.

Holland, D., Lachicotte, W., Skinner, D., & Cain, C. (1998). *Identity and agency in cultural worlds*. *History*. Cambridge, MA: Harvard University Press.

Horn, M. S., Brady, C., Hjorth, A., Wagh, A., & Wilensky, U. (2014, June). Frog pond: A codefirst learning environment on evolution and natural selection. In *Proceedings of the 2014 conference on Interaction design and children* (pp. 357-360). ACM.

Horn, M. S., & Jacob, R. J. K. (2007). Designing Tangible Programming Languages for Classroom Use. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction* (pp. 159-162). ACM.

Hu, C. (2011). Computational thinking – What it might mean and what we might do about it. *ITiCSE '11: Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (pp. 223–227).

Hung, D., Chen, V., & Lim, S. H. (2009). Unpacking the hidden efficacies of learning in productive failure. *Learning Inquiry*, *3*(1), 1–19.

Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers and Education*, *82*(March), 263–279.

ISTE, & CSTA. (2011). *Operational Definition of Computational Thinking.* Retrieved from https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CompThinkingFlyer.pdf.

Jagodzinski, A. (2016, June 30). STEAM on the rise: The growing importance of arts education [Blog post]. *Art Force*. Retrieved from http://artforce.org/steam-on-the-rise/.

Jawitz, J., Case, J., & Ahmed, N. (2005). Smile nicely, make the tea - But will I ever be taken seriously? Engineering students' experiences of vacation work. *International Journal of Engineering Education*, *21*(1), 134–138.

Johnsey, R. (1995). The place of the process skill making in design and technology: Lessons from research into the way primary children design and make. In *Proceedings of the IDATER95: International Conference on Design and Technology Educational Research and Curriculum Development*, Loughborough, UK.

Jorgenson, J. (2002). Engineering selves: Negotiating gender and identity in technical work. *Management Communication Quarterly*, *15*(3), 350-380.

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

Kafai, Y. B., Lee, E., Searle, K., & Fields, D. (2014). A crafts-oriented approach to computing in high school: Introducing computational concepts, practices, and perspectives with electronic textiles. *ACM Transactions on Computing Education*, *14*(1), 1–20.

Kafai, Y. B., Peppler, K. A., Burke, Q., Moore, M., & Glosson, D. (2010). Fröbel's forgotten gift: Textile construction kits as pathways into play, design and computation. In *Interaction Design and Children*, Barcelona, Spain.

Kafai, Y. B., Searle, K., Martinez, C., & Brayboy, B. (2014). Ethnocomputing with Electronic Textiles: Culturally Responsive Open Design to Broaden Participation in Computing in American Indian Youth and Communities. In *SIGCSE/14*, Atlanta, GA.

Kafai, Y. B., Searle, K., Fields, D. A., Kafai, Y., Searle, K., Fields, D., & Lui, D. (2013). Cupcake cushions, Scooby Doo shirts, and soft boomboxes: E-textiles in high school to promote computational concepts. In *SIGCSE'13* (pp. 311–316), Denver, CO.

Kalelioğlu, F., & Gülbahar, Y. (2014). The effects of teaching programming via Scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, *13*(1), 33–50.

Kaplan, A., & Flum, H. (2009). Motivation and identity: The relations of action and development in educational contexts – An introduction to the special issue. *Educational Psychologist*, 44(2), 73-77.

Kapur, M. (2008). Productive failure. *Cognition and Instruction*, *26*(3), 379–424.

Kelleher, C. (2009). Barriers to programming engagement. *Advances in Gender and Education*, *1*, 5–10.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, *37*(2), 83–137.

Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM*, *50*(7), 58.

Kimmel, H., Carpinelli, J., Alexander, L.B., and Rockland, R. (2006). Bringing engineering into k-12 schools: A problem looking for solutions? In *Proceedings of the American Society for Engineering Education Annual Conference and Exposition*, Chicago, IL.

Kolodner, J. L., Camp, P.J., Crismond, D., Fasse, B., Gray, J., Holbrook, J., Puntambekar, S., and Ryan, M. (2003). Problem-based learning meets case-based reasoning in the middle-school science classroom: Putting learning by design™ into practice. *Journal of the Learning Sciences,* 12(4), 495–547.

Kuhn, D. (1993). Science as Argument : Implications for Teaching and Learning Scientific Thinking. *Science Education*, *77*(3), 319–337.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge, U.K.: Cambridge university press.

Lester, F. K. (1994). Musings about mathematical problem-solving research: 1970-1994. *Journal for research in mathematics education*, *25*(6), 660-675.

Lucas, B., Hanson, J., Claxton, G., and Centre for Real-World Learning (2014). *Thinking like an engineer: Implications for the education system.* UK: Royal Academy of Engineering.

Lui, D., Litts, B. K., Widman, S., Walker, J. T., & Kafai, Y. B. (2016). Collaborative Maker Activities in the Classroom: Case Studies of High School Student Pairs' Interactions in Designing Electronic Textiles. In *FabLearn* (pp. 74–77), Stanford, CA.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12 ? *Computers in Human Behavior*, *41*, 51–61.

Mark, J. (1992). Beyond equal access: Gender equity in learning with computers. *Women's Educational Equity Act Publishing Center Digest*, 1-2-7.

Martin, D. B. (2000). *Mathematics Success and Failure among African-American Youth: The Roles of Sociohistorical Context, Community Forces, School Influence, and Individual Agency*. Mahwah, NJ: Lawrence Erlbaum Associates.

Master, A., Cheryan, S., & Meltzoff, A. N. (2016). Computing whether she belongs: Stereotypes undermine girls' interest and sense of belonging in computer science. *Journal of Educational Psychology*, *108*(3), 424–437.

Mattern, K. D., Shaw, E. J., & Ewing, M. (2011). Advanced Placement® Exam Participation: Is AP® Exam Participation and Performance Related to Choice of College Major? Research Report No. 2011-6. *College Board*.

Mayer, R. E. (1998). Cognitive, metacognitive, and motivational aspects of problem solving. *Instructional Science*, *26*, 49–63.

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: what's the connection? *Communications of the ACM*, *29*(7), 605–610.

McCaslin, M. (2009). Co-Regulation of Student Motivation and Emergent Identity. *Educational Psychologist*, *44*(2), 137–146.

McGee, E. O., & Martin, D. B. (2011). "You Would Not Believe What I Have to Go Through to Prove My Intellectual Value!" Stereotype Management among Academically Successful

Black Mathematics and Engineering Students. *American Educational Research Journal*, *48*(6), 1347–1389.

McLeod, D. B. (1992). Research on affect in mathematics education: A reconceptualization. In D.A. Grouws (Ed.), *Handbook of research on mathematics teaching and learning* (pp. 575-596). Macmillan.

Meyers, K. L., Ohland, M. W., Pawley, A. L., Silliman, S. E., & Smith, K. A. (2012). Factors relating to engineering identity. *Global Journal of Engineering Education*, *14*(1), 119–131.

Midian Kurland, D., Pea, R. D., Clement, C., Mawby, R., & Mawby, R. A. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, *2*(4), 429–458.

Moore, T. J., Glancy, A. W., Tank, K. M., Kersten, J. A., & Smith, K. A. (2014). A Framework for Quality K-12 Engineering Education: Research and Development. *Journal of Pre-College Engineering Education*, *4*(1), Article 2.

NASA's BEST (2016). *The engineering design process*. NASA. Retrieved from www.nasa.gov/foreducators.

Nasir, N. I. S. (2002). Identity, goals, and learning: Mathematics in cultural practice. *Mathematical thinking and learning*, *4*(2-3), 213-247.

Nasir, N. I. S., & Cooks, J. (2009). Becoming a hurdler: How learning settings afford identities. *Anthropology & Education Quarterly*, *40*(1), 41-61.

Nasir, N. S., & de Royston, M. M. (2013). Power, identity, and mathematical practices outside and inside school. *Journal for Research in Mathematics Education*, *44*(1), 264–287.

Nasir, N. S., & Hand, V. (2008). From the court to the classroom: Opportunities for engagement, learning, and identity in basketball and classroom mathematics. *Journal of the Learning Sciences*, *17*(2), 143–179.

National Academy of Engineering (NAE) and National Research Council (NRC). (2009). *Engineering in K-12 Education: Understanding the Status and Improving the Prospects.* Washington, D.C.: The National Academies Press.

National Center for Women and Information Technology (NCWIT). (2017). *By the Numbers*. Retrieved from https://www.ncwit.org/resources/numbers.

National Research Council (NRC). (2004). *Computer Science: Reflections on the Field, Reflections From the Field*. Washington, D.C.: The National Academies Press.

National Research Council (NRC). (2010). *Report of a Workshop on the Scope and Nature of Computational Thinking*. Washington, D.C.: The National Academies Press.

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

NGSS. (2013). *Science and Engineering Practices*. Washington, D.C.: The National Academies Press.

Nickerson, R. S. (1983). Computer programming as a vehicle for teaching thinking skills. *Thinking: The Journal of Philosophy for Children*, *4*(3/4), 42-48.

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

Papert, S. (1996). An Exploration in the Space of Mathematics Educations. *International Journal of Computers for Mathematical Learning*, *1*(1), 95–123.

Pea, R. D. (1983). Logo Programming and Problem Solving. In *American Educational Research Symposium* (pp. 2–10), Montreal, Canada.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*(2), 137–168.

Pea, R. D., Kurland, D. M., & Hawkins, J. (1985). Logo and the Development of Thinking Skills. In M. Chen & W. Paisley (Eds.), *Children and Microcomputers: Research on the Newest Medium* (pp. 193–317). Sage.

Pierrakos, O., Beam, T. K., Constantz, J., Johri, A., & Anderson, R. (2009). On the Development of a Professional Identity: Engineering Persisters Vs Engineering Switchers. In *39th ASEE/IEEE Frontiers in Education Conference* (pp. 1–6), San Antonio, TX.

Pinkard, N., Erete, S., Martin, C. K., & McKinney de Royston, M. (2017). Digital Youth Divas: Exploring Narrative-Driven Curriculum to Spark Middle School Girls' Interest in Computational Activities. *Journal of the Learning Sciences*, *26(3),* 477-516.

Potvin, P., & Hasni, A. (2014). Interest, motivation and attitude towards science and technology at K-12 levels: a systematic review of 12 years of educational research. *Studies in Science Education*, *50*(1), 85–129.

Renninger, K. A. (2009). Interest and Identity Development in Instruction: An Inductive Model. *Educational Psychologist*, *44*(2), 105–118.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM*, *52*, 60–67.

Robelen, E. W. (2013). K-12 Bolsters Ties to Engineering. *Education Week*, *32*(26), 1-18.

Rode, J. A., Weibert, A., Marshall, A., Aal, K., Von Rekowski, T., El Mimoni, H., & Booker, J. (2015). From Computational Thinking to Computational Making. In *UbiComp* (pp. 239–250), Osaka, Japan.

Roden, C. (1999). How children's problem solving strategies develop at key stage 1. *The Journal of Design and Technology Education,* 4(1), 21–27.

Rogers, C. B., Wendell, K., & Foster, J. (2010). A Review of the NAE Report, Engineering in K-12 Education. *Journal of Engineering Education*, (April), 179–181.

Roth, W. M. (1995). From "wiggly structures" to "unshaky towers": Problem framing, solution finding, and negotiation of courses of actions during a civil engineering unit for elementary students. *Research in Science Education*, 25(4), 365–381.

Roth, W. M. (1996). Knowledge diffusion in a grade 4–5 classroom during a unit on civil engineering: An analysis of a classroom community in terms of its changing resources and practices. *Cognition and Instruction,* 14(2), 179–220.

Royal Academy of Engineering (2017). *What is engineering?* Retrieved from http://www.raeng.org.uk/education/what-is-engineering.

Sadler, T., Barab, S., and Scott, B. (2007). What do students gain by engaging in socioscientific inquiry? *Research in Science Education,* 37(4), 371–91.

Schoenfeld, A. H. (1992). Learning to think mathematically: Problem solving, metacognition, and sense making in mathematics. In D. Grouws (Ed.), *Handbook for Research on Mathematics Teaching and Learning* (pp. 334–370). New York, NY: Macmillan.

Searle, K. A., Fields, D. A., Lui, D. A., & Kafai, Y. B. (2014). Diversifying high school students' views about computing with electronic textiles. In *ICER* (pp. 75–82), Glasgow, UK.

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

Smith, M. (2016, January 30). Computer science for all [Blog post]. *The White House Blog*. Retrieved from https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all.

Sneider, C., Stephenson, C., Schafer, B., & Flick, L. (2014). Computational thinking in high school science classrooms. *The Science Teacher*, *81*(5), 53-59.

Snyder, T. D. (2016). *Digest of Education Statistics: 2015*. National Center for Education Statistics.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, *29*(9), 850–858.

Swan, K. (1989). Programming objects to think with: Logo and the teaching and learning of problem solving. In *Annual Meeting of the American Educational Research Association*, San Francisco, CA.

Tan, E., Calabrese Barton, A., Kang, H., & O'Neill, T. (2013). Desiring a career in STEM-related fields: How middle school girls articulate and negotiate identities-in-practice in science. *Journal of Research in Science Teaching*, *50*(10), 1143–1179.

Tate, E. D., & Linn, M. C. (2005). How Does Identity Shape the Experiences of Women of Color Engineering Students? *Journal of Science Education and Technology*, *14*(5/6), 483–493.

The Economics Daily (2017). *Women in architecture and engineering occupations in 2016.* Bureau of Labor Statistics. Retrieved from https://www.bls.gov/opub/ted/2017/women-in-architecture-and-engineering-occupations-in-2016.htm.

Wang, D., Wang, T., & Liu, Z. (2014). A tangible programming tool for children to cultivate computational thinking. *The Scientific World Journal*, 2014.

Waterman, A. S. (2004). Finding Someone to Be: Studies on the Role of Intrinsic Motivation in Identity Formation. *Identity: An International Journal of Theory and Research*, *4*(3), 209–228.

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.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge, UK: Cambridge University Press.

Wenger, E. (2010). Communities of practice and social learning systems: The career of a concept. In C. Blackmore (Ed.), *Social Learning Systems and Communities of Practice* (pp. 179–198). London: Springer.

Wigfield, A., & Wagner, A. L. (2005). Competence, motivation, and identity development during adolescence. In A. J. Elliot & C. S. Dweck (Eds.), *Handbook of Competence and Motivation* (pp. 222–239). New York: Guilford Press.

Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Fostering Computational Literacy in Science Classrooms. *Communications of the ACM*, *57*(8), 24–28.

Wineburg, S. S. (1991). Historical problem solving: A study of the cognitive processes used in the evaluation of documentary and pictorial evidence. *Journal of Educational Psychology*, 83, 73–87.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education*, *12*(3), 197–212.

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

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions the Royal Society of London A*, *366*(1881), 3717–3725.

Wyeth, P., & Purchase, H. C. (2002). Tangible Programming Elements for Young Children. In *CHI'02 extended abstracts on Human factors in computing systems* (pp. 774–775). ACM.

Yelland, N. J. (1995). Encouraging young children's thinking skills with Logo. *Childhood Education*, *71*(3), 152–155.