Google

Pre-College Computer Science Education: A Survey of the Field

2018

TLTL tltl.stanford.edu Transformative Learning Technologies Lab

Pre-College Computer Science Education: A Survey of the Field

2018

Table of Contents

This report was commissioned by Google LLC. and authored by Paulo Blikstein of the Transformative Learning Technologies Lab, Stanford University Graduate School of Education.

Suggested citation: Blikstein, P. (2018). Pre-College Computer Science Education: A Survey of the Field. Mountain View, CA: Google LLC. Retrieved from https://goo.gl/gmS1Vm.

Additional reports from Google's Computer Science Education Research are available at g.co/cseduresearch.

1. Acknowledgments

The author wishes to thank the 14 interviewees for their willingness to share a few centuries of accumulated knowledge about computer science education (CSEd), as well as their invaluable insight and vision for the future: Matthew Berland (University of Wisconsin—Madison), Leah Buechley (Rural Digital), Michael Clancy (University of California, Berkeley), Andrea “Andy” diSessa (University of California, Berkeley), Sally Fincher (University of Kent), Shuchi Grover (Formerly SRI International), Mark Guzdial (Georgia Institute of Technology), Mike Horn (Northwestern University), Jane Margolis (University of California, Los Angeles), Mitchel Resnick (Massachusetts Institute of Technology), Sue Sentance (King’s College London), Ben Shapiro (University of Colorado, Boulder), David Weintrop (University of Maryland), and Pat Yongpradit (Code.org).

The author would like to acknowledge Sepi Hejazi Moghadam (Google) for his exhaustive and comprehensive review of the report, and his insightful suggestions throughout the writing.

2. Preface

There is growing excitement about and commitment to CSEd as governments and organizations move to make computer science courses available to all students. While this movement is driven by multiple rationales, there can be no doubt that all parties are motivated by the desire to ensure students have the skills and opportunities they need to thrive in a world where computing is ubiquitous and its impact is felt in all areas of study and work. Along with this commitment is the responsibility to ensure that CSEd implementation in formal education is grounded in a solid understanding of what students need to learn, when they are ready to learn, and how they can best be taught.

Computer science is a dynamic field in which change is a constant. It is also a young field, and as such, it lacks the extensive and comprehensive body of educational research that other academic disciplines possess. As a result, we are playing catch up to disciplines such as mathematics and science that have been part of the educational canon since the earliest days of schooling.

This paper provides two distinct perspectives into CSEd research. First, it provides a review of the current state of that research, outlining the current knowns and unknowns. Second, it shares the views of the highly-regarded researchers and practitioners who generously provided interviews for this paper. We believe that this combination provides a rich perspective on where we are now and where we need to go.

At Google, we believe in education and opportunity. We also believe in making decisions based on rigorous research. We hope this paper will help the CSEd community better understand what we know now and what we still need to learn. We further hope this knowledge can guide our efforts to support and contribute to new research efforts.

Google is proud to have supported Dr. Paulo Blikstein’s work and hopes that this paper will generate much discussion in the CSEd practitioner and research communities.

Chris Stephenson
Head of Computer Science Education Strategy
Google

3. Executive Summary

In 1967, Seymour Papert, Cynthia Solomon, and Wally Feurzeig created the LOGO computer language, the first ever designed for children—an event widely considered as the beginning of CSEd. It has taken a few decades to enter the educational mainstream, but the largest and most ambitious implementations of CSEd have now started to roll out. With this widening acceptance of CSEd comes an overwhelming demand from school systems for research-based knowledge and implementation guidelines, especially for K–8 schools. To help meet this demand, Google commissioned this report. We aim to better understand what we know—and what we don't know—about how children learn to program, the ways in which CSEd furthers the aims of public education, and how to chart a path to address imminent challenges. We have examined current literature and conducted interviews with 14 leading researchers in the field.[1] Our literature review reveals that the evidence and perspectives on what we know about how children learn to program is promising, but still limited:

- CS learning has the potential to be transformative. It includes algorithms, design, data, making, creativity, and personal expression. It also boosts the potential for productive collaboration and project-based learning in the classroom, connects to personally meaningful aspects of students' lives, allows for new types of knowledge and assessments to be valued in schools, and opens up innovative ways to organize learning environments.
- Developing new pedagogies and approaches for learning a discipline as new as CS is a challenge. First, CSEd requires students to have a well-developed mental model of what computers are and how they run code, and how to interpret, trace, and debug programs. Second, programming tools and content are always changing, often leading to new pedagogies that are harder to orchestrate (e.g., project-based learning, students conducting group projects). Third, acquiring and assessing expertise in CS might not follow the same patterns of traditional school disciplines. Despite all this (or perhaps because of it), CSEd can provide a powerful and authentic context for learning computing concepts and also the content of other disciplines. In this way, it can serve as a new foundational literacy and an expressive, creative medium to allow young learners to share ideas in socially and culturally relevant ways.
- Paradoxically, simplified programming languages and activities can complicate future learning if they are not carefully designed. Designers are not always aware of how their simplifications can lead students to form misconceptions regarding core CS ideas that might limit their future development. There is a need for novice-friendly programming environments and activities that provide robust pathways for transitioning into more complex projects or languages.
- CSEd implementations and tool development must be informed by well-researched age-related differences in what students can accomplish. Current research from science, technology, engineering, and mathematics (STEM) education and beyond should be used in the development of CSEd and the knowledge base should be continually expanded.
- New technologies and research methods are needed to help CSEd implementation, by creating tools to help teachers manage and assess complex student projects, and by providing researchers with new types of data and insights. Students' usage of programming tools can be instrumented to collect data, potentially bringing unprecedented insights into student learning. But the interpretation of these data is still a challenge, and determining what representations of these data are useful to developers, teachers, and students is an open research question.

Addressing the known gaps requires attention to perspectives around how to implement CSEd equitably in schools. To do this, we must align different points of view about why CS should be in schools, decide on the kind of preparation and development teachers receive, increase our understanding about how learners develop key CSEd concepts and the appropriate practices within and across grades, and improve CSEd through research. In our analysis of interview data, we found that:

1 While most of the interviewees were from the U.S. and are grounded in the U.S. educational experience, the author hopes that international readers may be able to draw useful parallels to their own systems and research needs.

- Important differences remain as to why CS should be in schools. We found varying rationales for CS in schools, but also some similarities that suggest the possibility of finding common ground to advance the field.
- Many believe that transforming CSEd will require special attention to equitable participation and integrated systems of teacher development. Equity and inclusiveness are seen as critical to advancing CSEd and are imperative to teacher development.
- Many expressed the need for comprehensive rollouts that consider the creation of state-level standards, develop curricula and assessments, use appropriate pedagogies for various grade levels (especially for K–8), focus on teacher preparation and certification, provide appropriate software/hardware infrastructure, and incentivize research and evaluation. Otherwise, concern was raised about partial or selective rollouts as having the potential to further exacerbate social disparities and educational inequalities by favoring affluent schools or districts.
- Although there is much to celebrate and there are many success stories, our review found that the unintended consequences of the success of “CS exposure” projects are that they might lead to less focus on sustained activities because they generate a false sense of how much it takes to teach CSEd more broadly and deeply.
- Despite the existence of CS standards, we found no comprehensive K–12 CS curriculum. Many believe that this curriculum should be attuned to cultural differences and made meaningful to increasingly diverse populations of students. Some expressed that excessive formalization and standardization of the CS curriculum might undercut the purpose of CSEd and diminish its potential for cross-disciplinary, creative, and innovative work.
- Research across grade levels reveals that students’ mental models about what a computer does when it executes programs predicts how well they learn to program, so the learning of such models should be a major focus in CSEd curricula.
- The number of researchers and research programs in CSEd appears to be insufficient to deal with new large-scale rollout programs. Interviewees expressly stated that CSEd research must become more rigorous and connect more with new and established knowledge in cognitive science, education, learning sciences, and data mining. New funding pathways are also viewed as necessary for sustaining basic and applied research.

3.1 Recommendations

To fulfill the vision of a meaningful and sustained CSEd field that meets the needs of all students, policymakers, educators, and the research community should consider improving the key areas identified by this review:

Create clarity around the different visions of CSEd

- Create clarity and alignment around the core rationales that varied stakeholders use to advance CSEd, highlighting their synergies, differences, and consequences for classroom instruction.
- As CSEd grows, it should maintain some of its key transformational and innovative elements, such as the focus on student project-based work and alignment with learner interests and ways of expression.

Make participation equitable

- National roll-outs of CSEd must prioritize and evaluate their impact on improving the equitable participation of all students regardless of backgrounds, motivations, preparations, and abilities.
- CSEd should be a mandatory content area in public schools in order to overcome biases and structural inequalities that prevent equitable participation.

Ensure teachers are prepared and supported

- Develop integrated systems of teacher certification, training programs, and professional incentives, with special attention to the pre-service pipeline for underrepresented communities.
- Provide high-quality teacher preparation and induction models focused on inclusive CS pedagogical content knowledge.

Create continuity and coherence around learning progressions

- Describe recommended sequences for CS knowledge and skills that can build on one another as students learn new topics over time.
- Develop robust and developmentally-appropriate programming tools for multiple age groups, especially for K–8, and domains that also provide additional insights into student learning.

Commit to ongoing and thorough research

- CSEd research funders, researchers, practitioners, and policymakers should develop a strategic plan for CSEd research, making it a stable, academically valued, and well-funded enterprise for years to come.

Underlying these findings and recommendations are the social, economic, and cultural barriers surrounding computing. Experts agree that if CSEd programs are not implemented with an eye toward equity, they will deepen educational inequalities that already exist and defeat the purpose of CSEd as a force for youth empowerment, democratic labor market access, and social justice. Much remains to be learned about the scalability, external validity, and optimal design of CSEd implementations. Given the scope and complexity of demands placed on them, interdisciplinary and inter-sector partnerships between public schools, universities, researchers, and industry will play a pivotal role in meeting the aforementioned objectives.

4. Introduction

In 1967, Seymour Papert, Cynthia Solomon, and Wally Feurzeig created the LOGO computer language, the first designed for children (Papert, 1980)—an event widely considered as the beginning of CSEd.[2] In a time when computers cost millions of dollars and occupied entire rooms, teaching CS for children, while visionary, was a hard sell for school systems and policymakers. From the mid-1970s to the early 1990s, CSEd slowly penetrated schools worldwide. Despite a decade of popularity in the 1980s, it never reached as deeply into the educational mainstream as Papert and his colleagues wished . Since the mid-2000s, however, there has been a pronounced shift in the focus on STEM education, and CSEd is at the forefront of this process (National Research Council, 2012). As computational technologies have become inexpensive and pervasive in our lives, so has the demand for an educated and technologically literate labor force (Noonan, 2017; U.S. Department of Labor, 2007). The need for children to become future producers of technology, fluent in the medium of our time, instead of merely consumers has become a major focus for policymakers and researchers. Today, educators and CSEd advocates are pushing ahead with plans to add CS to the list of topics that all students should study (K–12 Computer Science Framework Steering Committee, 2016).

Other catalysts to the mainstream acceptance of CSEd include the launch of the Scratch, Blockly, and Alice programming environments; the launch of CS teacher organizations such as the Computer Science Teachers Association (CSTA) (an international body founded by the Association for Computing Machinery [ACM]) the rise of the maker movement and fablabs; the creation of organizations providing CS learning opportunities such as Code.org, Black Girls Code, Girls Who Code, and others;[3] and the rollout of national programs such as CS4All. As a result, there is an almost overwhelming demand from school systems worldwide for research and implementation guidelines, one which the relatively small CSEd education research community is simply not able to meet (Guzdial, 2017).

The newness of the discipline is also an important factor. For example, while the U.S. National Council of Teachers of Mathematics (NCTM) was founded in 1920, and its science counterpart, the National Science Teachers Association (NSTA) was formed in 1944, CSTA was not launched until 2004. When NCTM and NSTA were formed, school infrastructure was already in place for these disciplines, thousands of mathematics and science teachers were teaching in schools across the U.S., and teachers colleges supported a strong pipeline for more. *CSEd does not have those advantages today*. The current focus on CSEd has also generated much discourse regarding its purpose. Is the rationale for CSEd to fulfill job market needs, promote personal empowerment, teach children to code, develop students' fluency in a new literacy, address historical educational inequalities, or some combination of all of the above? **See a timeline for CSEd in Appendix C.**

Google commissioned this work to better understand the knowns and unknowns with regard to the state of the CSEd field in relation to our understanding of student learning and the research opportunities that exist or that might be created to ensure fruitful and sustained advancement for all students. With this goal in mind, this report summarizes an examination of literature reviews and articles and interviews conducted with a number of leading researchers in the field.

2 John Kemeny and Thomas Kurtz (Dartmouth College) created the BASIC programming language in 1964, but LOGO is used as a landmark because of its comprehensive focus on all segments and age levels of education, especially children.

3 There is a large number of such organizations, many focusing on underserved populations: Black Girls Code, Girls Who Code, Girls Code it, CoderDojo, Technovation, Yes We Code.

5. Methods

We utilized three major data sources for this report: (1) a review of all foundational works in the field, and existing literature reviews, (2) interviews, and (3) analysis of papers and resources recommended by interviewees **(see Appendix A for full details on the methods used and for the full list of interviewees).**

For the interviews, we selected leaders in the field from various universities and institutions, trying to balance intellectual traditions, academic backgrounds, and expertise. The final group of interviewees consisted of 14 practitioners, researchers, and scholars. We used a semi-structured protocol for the interviews that included questions about the relevance and importance of teaching CS, the main research findings in the field, and research, policy, and implementation agendas for the next year **(see Appendix B for interview protocols).**

We used the literature to add a layer of peer-reviewed research to the themes extracted from the interviews, and triangulated research findings across interviews and the literature. We chose this hybrid format (interviews and reviews) to simultaneously capture well-established facts and findings from seminal and contemporary literature, and novel information that has not yet been published in the field. Also, some of the important challenges and issues in CSEd do not show up in peer-reviewed publications because many active members of the community are tool developers rather than researchers—so their work is less likely to be be captured in a traditional literature review. This combined use of interviews and literature gave us a more comprehensive view of the state of the young and dynamic field of CSEd.

After the first complete draft was finished, all 14 interviewees were given the opportunity to fully review the text and suggest further changes, which were individually considered for the final version.

6. Rationales for Justifying CS Education

Support for CSEd is strong, but the reasons why often vary. Similar to a recent study by Vogel, Santo, and Ching (2017), we found that the interdisciplinary nature of CS brings together very different stakeholders and views. CSEd includes professionals from different academic cultures and professional allegiances: university professors, K–12 educators, CEOs of technology companies, entrepreneurs, government officials, and diversity and equity advocates. Not surprisingly, the data from the interviews and literature revealed many different justifications for why CS should be taught in public education systems (e.g., diSessa, 2000; Wing, 2006). These rationales can be expressed as four distinct positions:

- The labor market rationale,
- The computational thinking rationale,
- The computational literacy rationale, and
- The equity of participation rationale.

Making these four rationales explicit is important because they drive the way we write curricula, train teachers, and implement CSEd in schools. Interviewees pointed out that the public's lack of awareness about these different viewpoints—and the ways they are similar, dissimilar, complementary, and compatible—must be addressed (e.g., Buechley, 2017; Resnick, 2017).

6.1 The labor market rationale

Labor market changes and the need to sustain a competitive economy are the main driving forces for this rationale. Some consider that CS knowledge will be useful in a variety of 21st century non-technical jobs, so it will be universally valuable for all professions.

Changes in the U.S. labor market have been a major driver of the efforts to teach CS in the nation's schools. This rationale is primarily related to the demands for more workers with new skill sets and is frequently championed by industry leaders and policy makers. The labor market argument comes in two chief forms. The first cites the hundreds of thousands of open jobs in CS (Google LLC & Gallup Inc., 2016; Grover & Pea, 2013), and notes that this number will increase in years to come, with data science and artificial intelligence becoming mainstream fields relevant across many industries. Similarly, it is argued that the economic productivity or contributions of a country will be determined by its capacity to generate more scientists and engineers. CSEd can presumably contribute to this vision by fixing the "leaky" STEM pipeline and driving more students to pursue CS careers. However, Grover and Horn point out that in grades K–8 especially, this concern with jobs might be misplaced:

> In elementary school, students and teachers are definitely not thinking about jobs. It is about what are the foundational knowledge and skills that children should have? At the middle school level, even though it is not a jobs argument, I think there is an identity argument there. This is especially relevant to computing because there are so many stereotypes associated with it. (Grover, 2017)
>
> We have gone a little too far on the commercial end of the spectrum, we have become preoccupied with training the next generation of engineers, these economic motivations are outweighing the computational literacy ideas. (Horn, 2017)

The second form the labor market argument takes is a subtler one. It argues for more CS knowledge embedded in all careers, instead of simply training more programmers. Several of the interviewees mentioned that while professional programmers will be necessary, the need could be restricted to a relatively small number of positions that are highly specialized (Guzdial, 2017; Resnick, 2017; Shapiro, 2017). Some reports suggest that only about six percent of the workforce will need to do coding with the scope and specialization of professional programmers (Noonan, 2017). The greatest demand would not be for professional programmers, but for other professionals who will have to use CS and programming for automating spreadsheets, programming queries, accessing online databases, using data mining software tools, and operating physical computing devices in interactive art or home automation.

6.2 The computational thinking rationale

The argument for "computational thinking" is that computer scientists' ways of thinking, heuristics, and problem-solving strategies are universally important, would transfer to a variety of knowledge domains and to the solution of everyday problems, and would support the development of students' higher-order thinking skills.

The second argument for teaching CS derives from the concept of "computational thinking," (CT) as put forth in a position paper written by Jeanette Wing (Wing, 2006). Wing proposed that computer scientists' ways of thinking, heuristics, and problem-solving strategies are universally important for both applying computing ideas to do work in other disciplines, and for applying computing ideas in everyday life. Examples are the ability to use abstractions and pattern recognition to represent problems in new ways, to break down problems into smaller parts, and to employ algorithmic thinking. With 3,000 citations (according to Google Scholar as of October 2017), the position put forward by Wing has played a critical role in shaping the world of CSEd. Her paper and her influential position as a National Science Foundation (NSF) officer helped reinvigorate the field. Some researchers, however, are skeptical about how well students transfer CS knowledge to everyday life and general problem-solving. diSessa (2017) mentions that there have been several attempts over the last 100 years to teach children transferable problem-solving or higher-order thinking skills (HOTS) using mathematics, Latin, or Greek, but these endeavors often failed. Guzdial (2017) mentions several studies on the transfer of CSEd knowledge and points out that generally "students fail to apply even simple computing ideas to fairly simple problems." Yongpradit further notes that:

> CSEd is not immune to the misconceptions about high-level transfer. I know that there are advocates… saying that computer science can improve general critical thinking skills. That's not supported by research. It will not magically improve your math scores. (Yongpradit, 2017)

Because Wing's original ideas are still influential in the field, the lack of empirical evidence and the absence of a more definitive unpacking of the term CT are considered to be major gaps in CSEd. But the definition of CT has been evolving over the last few years, as Grover notes:

> The definition of CT has been evolving since Wing, and in its evolution it has broadened to encompass aspects of CT concepts, practices, as well as learners' dispositions and perspectives, perhaps fueled by a genuine desire to broaden participation, thus including aspects such as creativity, collaboration, and communication in practices of CT. (Grover, 2017)

6.3 The computational literacy rationale

Computational literacy is not a new skill or a class of problem-solving strategies, but a set of material, cognitive, and social elements that generate new ways of thinking and learning. It enables new types of mental operations and knowledge representations, creates new kinds of "literatures," makes it possible for people to express themselves in new ways, and changes how people accomplish cognitive tasks.

With more than 1,000 citations (according to Google Scholar as of October 2017), Andrea diSessa's book *Changing Minds* is the most established account of the idea of "computational literacy" (diSessa, 2000). In the book, and in recent publications (diSessa, in press), he explains how different computational literacy is from the original definition of "computational thinking" (a similar discussion appears in Wilensky & Papert, 2010).

> Learning to use a new medium takes effort. The printing press was a huge leap in human history, but that leap did not happen until many more people became literate. A printing press is not of much use unless authors know how to write and your audience knows how to read. Achieving computational literacy in society means that people can read and write with computation, which includes an ability to read and write computer programs. (diSessa, 2000)

> I view computation as, potentially, providing a new, deep, and profoundly influential literacy—computational literacy—that will impact all STEM disciplines at their very core, but most especially in terms of learning. (diSessa, in press)

diSessa claims that computational literacy is not simply a new job skill or generic CS-inspired problem-solving strategy, but a set of material, cognitive, and social elements that generate a new way of knowing, thinking, learning, and representing knowledge. A new literacy makes new types of mental operations and knowledge representations possible, creates new kinds of previously nonexistent "literatures", and changes how people interact with each other and use computers and digital devices when they are accomplishing cognitive tasks. He also mentions that there is a semantic confusion between computational literacy versus terms like digital literacy, computer literacy, or Information Communication and Technology (ICT) literacy. These latter terms refer to the competent use of different computational devices and technologies. Computational literacy, conversely, is concerned with how computational media can change the way we know, learn, and think (in contrast with the focus on problem-solving or higher-order thinking skills).

diSessa also argues that concepts in science and mathematics can be made simpler using computational representations. For example, velocity and acceleration are simpler to understand algorithmically but unnecessarily complex to learn using traditional algebraic representations. Chemical processes such as diffusion, given their probabilistic nature, are convoluted when represented in algebraic terms, but very simple to learn using computational tools such as agent-based models (e.g, NetLogo, Wilensky, 1999), in which students can program the behavior of atoms. The argument for computational literacy extends beyond the need for teaching programming languages. It makes the claim that several disciplines could be fundamentally transformed if taught using computational tools, in the same way that text literacy changed the teaching of so many disciplines centuries ago.[4] Sentance, Resnick, and Horn also stress that

4 Text literacy fundamentally changed how we accomplish cognitive operations—for example, it acts as external memory, it is shareable, and permanent. diSessa and others claim that computational literacy could have the same revolutionary consequences.

computational literacy is multi-faceted, and more than just learning computational thinking or programming concepts:

> I think computational thinking skills exist…I think we just have to be careful about thinking that computer science is only computational thinking. CS…involves modeling and design and creativity, more than just the cognitive elemental thinking skills. That is what we need to teach in K–8. We need to teach the whole subject and be cautious of being too narrow in what we are offering in the curriculum in school. (Sentance, 2017)

> Gaining a literacy is a matter of developing your thinking, your voice, and your identity…The reason for learning to write is not just for doing practical things but being able to express your ideas to others. Computation is a new way of expressing ourselves and it's important for everyone to learn…If you want to feel like a full participant in the culture, you need to be a contributor with the media of the times. (Resnick, 2017)

> It is about supporting computation in many different genres or niches. As a poet, the way you use computation might be very different than a journalist, a researcher, or somebody who works in government. Just like we have different forms of literacy, we might have different forms of computational literacy. (Horn, 2017)

However, as diSessa states, discussions about the role and importance of CSEd are far from over and these views should all be earnestly considered with their implicit contradictions:

> The labor market view and the computational thinking view contain at least implicit criticisms of the computational literacy view. The former might think that immediate and practical economic effects are more important, and the latter suggests that computational literacy is diffuse, hard to implement, and might insist that high-order thinking skills do exist, so these perspectives should not be ignored. (diSessa, in press)

Some interviewees pointed out that the boundaries between CT and computational literacy are not well-defined. While Grover (2017) states that new definitions of CT have been evolving to include, for example, creativity and collaboration, formerly mostly associated with computational literacy, Guzdial (2017) worries that these new CT definitions "are going too broad," and Resnick notes that the definition of CT "out in the field" is still very much connected to the original one as stated in Wing's 2006 paper.

6.4 The equity of participation rationale

CS knowledge will be required for the best and most creative jobs, for civic participation, and for understanding the impact of computation on society. Additionally, since our cognitive capabilities will be limited by our ability to utilize computation, equity of participation in CSEd becomes the central concern, and is one of the most significant gaps in research and implementation.

Several interviewees mentioned equity as their central concern in CSEd, arguing that it has traditionally been a side issue in the field and one of the most significant gaps in research and implementation. There are two main issues related to the topic: 1. Understanding the impact of computation on society, and 2. Ensuring equity and diversity in participation.

The K–12 Computer Science Framework (K–12 Computer Science Framework Steering Committee, 2016) also recognized equity and broadening participation as one of the core issues in CSEd.

Students excluded from CSEd may struggle to fully participate in 21st century society along multiple dimensions. Not only will the best and most creative jobs require CS knowledge, but our cognitive capabilities to solve problems will be limited by our inability to utilize computation fully. Even traditional forms of civic participation will require an understanding of CS. As Buechley stated:

> We live in a computationally mediated world, and it is important for people to have an understanding of how computational systems work and the role that they play in those systems, how those systems impact their lives, our democracy, the economy, and the way we socialize and interact with people. (Buechley, 2017)

Several interviewees gave examples of how computer science will become increasingly crucial for civic participation and informed decision making. These examples include knowing what algorithms are, how computational tools can manipulate social media, how to participate in a social discourse mediated by algorithms, and how to make sense of job displacement due to automation. It is also important to be aware of the presence and consequences of technologies such as machine learning (ML) and artificial intelligence (AI) in a number of everyday devices and experiences; understanding how much information we divulge (sometimes unknowingly) about ourselves; and being aware of the ways in which bias can get built into technologies that influence critical decisions such as prison sentencing, mortgage allocation, and the deployment of neighborhood policing resources (O'Neill, 2016; Shapiro, 2017). The comprehension of the rapidly evolving landscape of devices and tools that are key for active participation in modern society is also central to this argument. Students who do not fully understand these issues risk being more easily manipulated as consumers, voters, and citizens, and more vulnerable to cybercrime. They also are less likely to have access to leadership positions and high-status jobs, and are more likely to be on the sidelines of future societal change.

The interviewees also noted that CS drives innovation across many disciplines and industries and that the resulting changes have had both an economic and sociological impact. Some also said that allowing students to explore their social and cultural concerns using computing helps motivate and engage them and make CS relevant to their lives, especially in diverse populations (Margolis, 2017). Buechley (2017) adds that when you put computing in contexts that can be compelling and exciting to different groups of people, "you get diverse populations to show up and participate," and stresses the importance of making conscious, deliberate space for that to happen. Many interviewees noted that private and more affluent schools will most certainly be able to offer CSEd programs with high complexity, while less affluent or public school systems will only offer very simplified versions:

> Private schools do not do just generic education. They have kids working on portfolios. They have children doing internships. They have kids doing projects and making it relevant to them…Standardized education which has no connection to kids' lives is what is often given to poor kids. (Margolis, 2017)

> [I was] working first in informal settings and then in recent years, I have moved more in the formal space. I saw it as being more relevant because that is now seen as a way to level the playing field and make sure that all children get it, not just those that happen to be fortunate to get it through after-school experiences. (Grover, 2017)

Grover noted that the Obama administration's naming of the national CSEd effort as "Computer Science for All" when it was announced in January 2016 supported this perspective:

> This of course came as a result of notions the community grew to accept over the previous 5 years... CSForAll is now a well-used term that captures this "equity of participation" notion. (Grover, 2017)

Sentance (2017) stresses the importance of making CS mandatory in all schools, for all students, not as mere "exposure," but as a way to avoid self-selection. The interviewees also noted that the lack of a diverse CS workforce results in the design of products and services that cater to a very narrow range of people and problems, thus perpetuating inequality. Researchers concerned with the equity argument also posit that we could see a much worse version of the "digital divide" in the years to come if immediate and intentional actions are not taken to address these inequities while we are still in early design stages of CSEd. diSessa believes that there is agreement in the community about the topic:

> I don't think there's any reasonable dissent on the importance of social context and diversity concerns. Only strategic differences. (diSessa, in press)

7. Implementation Considerations

This section highlights perspectives on key components needed across the CSEd system to support wider and more effective implementation. The "system" we define includes the various interrelated institutions and mechanisms that shape and support CSEd teaching and learning in the classroom.

The key components of CSEd that we review in this section are curriculum, instruction, and teacher development. It's difficult to focus on any particular component without considering how it is influenced by—and how it in turn influences—the other components. For example, what students learn is clearly related to what they are taught, which itself depends on many elements: the instructional materials available in the market; the curriculum adopted locally; teachers' content and pedagogical knowledge; how teachers elect to use the curriculum; the kinds of resources, time, and space that teachers have for their practice; what the community values regarding student learning; and how local, state, and national standards and assessments influence instructional practice.

We are not attempting to provide a full discussion of all possible influences on CSEd; rather, we focus on the themes that emerged from our review and how they might contribute to a more coherent and inclusive implementation of CSEd.

7.1 Systemic obstacles

Equity should be a priority, but rushing products to market can harm efforts to attract underrepresented students. As CSEd scales up, excessive formalization and standardization might undercut its very purpose and hinder development of creative solutions and uses of CSEd.

The interviewees highlighted the importance of systemic obstacles to consider when scaling up CSEd efforts and programs. The following sections explore the barriers they identified.

The need to broaden equitable participation. Several interviewees mentioned broadening participation in and changing perceptions of CS as perhaps the most important challenges for our community. Berland, Buechley, Margolis, Sentance, and others stressed the striking contrast between what happens in CS classrooms in affluent schools and less affluent schools. Almost all of the interviewees expressed concern with the unequal presence of CS in public schools, the quality of instruction, and the unconscious bias of some educators and counselors regarding who is "suited" to take the CS classes. They also noted that while affluent schools are more likely to offer comprehensive CS programs for their students, most public districts are ill-equipped to offer anything more than very brief, standardized experiences which they fear could give school administrators and teachers an incorrect metric for CS adoption and distract them from implementing more robust CS programs in their schools. The interviewees also worry that the reach numbers advertised by nonprofits and industry providers give the impression that the "mission has been accomplished," whereas most agree that we are still very far from providing CSEd to all students. At least three researchers also noted that funding currently provided to large national organizations would be better directed to research institutions or smaller, more local nonprofits. Yongpradit (2017) noted that national organizations can be a channel for funding to smaller organizations: "Code.org supports local implementation through...more than 60 regional partners, most of which are local nonprofits."

It is essential to examine how to broaden participation in CSEd. Most interviewees favored programs that make learning CS more attractive by focusing on personal expression and creativity, especially at K–8 level. They also agreed on the importance of culturally relevant curricula that support diverse ways of approaching CS and diverse ways of expressing one's knowledge. Buechley (2017), for example, mentioned that computer scientists and engineers tend to discount culture and cultural relevance as key factors in learning and in CS educational tool design. In her work, she instead focuses on creating *new types of clubhouses* and computing cultures that speak to these diverse practices. Michael Clancy also advocated for CSEd that incentivizes meaningful engagement:

> Students will be more motivated to work if the assignments allow creativity, and allow the student to relate to his or her experience. Part of that would be

> more flexible tools that allow a student to make better use of his or her experience. What I would like to see is some way to have a broader scope and interest of activities (Clancy, 2017).

Some identified the need to make CSEd mandatory for all students as a means of ensuring equitable participation. Sentance (2017), for example, argued that "if we don't make computer science mandatory, we know from previous experience that self-selecting groups of people will choose computing…so we have a responsibility to offer that to all children and to reach everybody." Yongpradit (2017) stated that schools should at least be required to offer CSEd, and that we should make CS and CS-related courses available permanently for students in public schools. Guzdial (2017) expressed concern that some states are trying to implement "CS4All" without an explicit focus on underserved groups. He points out that affluent schools will be able to move quickly to provide CS for their students while less affluent schools will struggle with financial limitations, further exacerbating the "coding divide." Margolis also noted that, while the CS for California campaign has an equity agenda,

> The rush to scale and the pressure to put curriculum and teacher professional development (PD) online will possibly have dangerous unintended consequences for the issue of equity...The learning partnership of teachers and of researchers needs to become part of a dynamic iterative cycle for continuous improvement… For programs to sustain themselves, to change the culture of the schools so that teachers are supported to have active, engaged, inclusive classrooms, for programs to be fully embraced by the districts themselves. It is the slow work of relationship building and learning together that is required. For this to happen there also needs to be a holistic awareness of all the educational issues in schools that continue to threaten equity. CS in schools does not exist on isolated islands. All of the large issues impacting education, such as the move for privatization, de-professionalizing teachers, and school tracking will affect our broadening participation in the computing mission. (Margolis, 2017)

Different approaches for the scaling and assessment of CSEd. Buechley, Shapiro, Berland and other interviewees expressed concern about traditional forms of school reform taking over the implementation of CSEd. Specifically, they noted that fixed curricula, standardized assessments, and inflexible teacher training programs do not foster real scientific or mathematical thinking in students (National Research Council, 2006; 2012) and have a questionable track record for motivating students to pursue STEM careers (Maltese & Tai, 2011). For Buechley (2017), one dominant narrative around CSEd is that "we need to figure out the concepts, and teach them in the right way in a fixed curriculum." She disagreed with this narrative, however, and instead advocated for a perspective in which motivation, engagement, personally relevant projects, and culturally aware curriculum design take precedence. diSessa (2017) stressed that "this is quite consistent with the computational literacy perspective, which emphasizes use over mere technical proficiency." According to Buechley, CS lends itself especially well to projects and interdisciplinary work that connect CS to art, design, biology, or mathematics:

> Connecting computation and computing to different practices, which sometimes coincide with really different ways of approaching and making sense of the world, is the most powerful way that you can engage different kinds of people in computing…As one example, I have been connecting computation to textile crafts, textile design, and fashion design, and I have found that through doing that, you can dramatically change the gender participation ratios. You can get lots of young women to engage enthusiastically with computing in a way that they just do not do in more traditional computer science contexts.
>
> Computer science is a fundamentally creative discipline. You construct things when you write a computer program. And in that sense, it's really distinct from mathematics or science. That is a distinction that is not fully appreciated and made sense of, but is very powerful and important. (Buechley, 2017)

Berland expressed a similar concern:

> There are very few subjects in which students feel like they can make a change in the world and they can express their independent selves. I think their ability to make their own games, make their own art, make them in ways that are shareable with code, is really powerful. [Instead of giving students the right answer] it is better to create safe spaces to fail, to play, to tinker…This is where you get the bang for the buck. That's where the learning happens. Another truism of education is that things are driven by the ways that they are assessed. If you assess people for knowing this or that keyword in C++[5], then that's what you're going to get and that's not particularly valuable, but if you assess people on their ability to teach each other complex concepts, that's what you're going to get. (Berland, 2017)

Fincher (2017) cited the UK's Project Quantum[6] as an example of an explicitly research-based project that combines scholarly work, practical utility, curriculum scaffolds, and teacher PD.

The persistent lack of resources, rush to release low-quality programs, and reliance on surface-level solutions. Margolis expressed concerns about the speed at which solutions are being developed and put into classrooms, and argued that this approach has unintended educational consequences, especially for members of underrepresented groups:

> The idea of many programs is] ship it out. Get it out there and we will see if there are bugs in it, right? That has some real potential dangers in education because you put something online and the school district says, "Okay, we're going to do computer science online," and then all of a sudden the girls and a lot of the students of color don't do well, and then the principal says, "See? Our kids are not up for computer science. They didn't do well. They're not interested." In fact, they just experienced horrible instruction, and so they get turned off, but in their minds they're not cut out for it, and in the minds of the principals they're not cut out for it. (Margolis, 2017)

Grover voiced a similar concern. She has been observing and researching citywide implementations in the U.S. and examining the quality and depth of the projects. She noted the simplicity of the projects she observed and the need to more deeply engage groups historically underrepresented in CS:

> Almost no one uses Boolean logic. They use variables but just as a count or a score. You barely ever see expressions with variables being used or you will rarely see a loop with a terminating condition that is controlled by a Boolean expression with variables. Also, I read this paper from Yasmin Kafai and Deborah Fields where they analyzed the Scratch community projects [in 2012].[7] Most children stayed at the shallow end, they used the simplest constructs. (Grover, 2017)

Shapiro (2017) voiced concerns about the concentration of resources in just a few CSEd organizations, which could lead to "very homogeneous curricula/programs which would move us in the opposite direction" from many of the progressive approaches discussed in the CSEd community. Similar concerns have been voiced by many prominent educators in light of large-scale implementations in many U.S. cities. As those implementations roll out, the quality of instruction has often been criticized as superficial, stifled, and insufficient to create fluency. Gary Stager observed:

> I wish I had 1 cent for every educator who has told me that her students "do a little Scratch." I always want to respond, "Call me when your students have done a lot of Scratch." The epistemological benefit of programming computers comes from long intense thinking. Fluency should be the goal. (Stager, 2017)

Changing perceptions of CS and exploring new domains and tools. Interviewees discussed the importance of different ways of doing CS, in terms of tools, programming

5 C++ is a very popular professional programming language.
6 http://community.computingatschool.org.uk/resources/4382/single

7 *The paper examined data from a subset of about 5,000 users in January 2012.*

languages, developmental levels, and approaches to organizing one's practice. In 1990, Sherry Turkle and Seymour Papert published an influential paper on *Epistemological Pluralism,* in which they described a study where children engaged in programming in a variety of ways that were all ultimately successful (Turkle & Papert, 1990). Even though some children were violating the canons of traditional programming practice (the "bricoleurs"), they were doing so in a personally meaningful way that allowed them to create a strong connection with programming. Echoes of this influential paper were heard in almost all the interviews, and the principle of epistemological pluralism appears to have taken hold in CSEd at the K–8 level. Grover (2017), however, pointed out that the epistemological pluralism approach needs to be combined with the teaching of some agreed-upon concepts and programming practices. When Resnick (2017) pointed out the need to keep pushing for epistemological pluralism, he noted that some systems only reward students for standard ways of doing coding (i.e., the smallest number of blocks when solving a puzzle), and some automated assessment programs still grade students solely based on the number and types of programming blocks they use. The interviewees also expressed the belief that traditional professional or college-level practices should not be automatically used in K–8 environments, since nontraditional approaches to programming (such as bricolage) may make sense only for younger students, even if advanced programmers might sometimes make use of these techniques as well (Berland, Martin, & Benton, 2013; Blikstein, 2011; Blikstein et al., 2014; Brennan, 2013; Graham, 2004).

Government officials need support in scaling efforts. Guzdial (2017) is currently helping many states conduct landscape surveys[8] to determine the state of CSEd in different parts of the country. He contends that policy decisions and coordination between different stakeholders would be much easier if landscape surveys were standard operating practice, as they allow states to gauge the growth of CS offerings, PD programs, and enrollments. Yongpradit (2017) also noted that federal and state-level organizations urgently need technical assistance around creating certifications, growing the CSEd teacher pipeline, and implementing curricula. Because CSEd is such a new field, there are too few trained professionals and specialized organizations that can offer those services. Yongpradit also expressed concern with current funding levels, noting that CSEd requires more PD, standards development, and support for task forces to create implementation plans.

7.2 Curriculum and instructional materials

There are a variety of curricula and instructional strategies that have been explored in CSEd. Among the recommendations we have seen are for CS curricula to be culturally relevant and meaningful to students, for STEM subjects and CS activities to be integrated, and for CS courses to be designed based on code production rather than specific languages.

Curriculum refers to the knowledge and practices that teachers teach and that students are supposed to learn in a subject. A curriculum generally consists of a scope, or breadth of content, and a sequence of concepts and activities for learning. The production of a quality curriculum and curricular materials is, for many interviewees, a key component for successful CSEd implementations at scale. The interviewees noted that this is an area of significant and ongoing challenge despite efforts such as the K–12 CS Framework (K–12 Computer Science Framework Steering Committee, 2016):

> No one yet has written out a full, coherent K–12 curriculum built around a foundational framework. The K–12 CS Framework and the CSTA standards have laid out concepts, practices, and performance expectations but how do these things get manifested in curriculum and activities and experiences in K–12? That is a huge problem in computer science right now that directly affects implementation. (Yongpradit, 2017)

Creating comprehensive curriculum materials is especially challenging because there is a natural tension between uniformity and the potential for customization to the learners' interests. Many interviewees noted the need to design culturally and personally relevant curricula that would cater to diverse populations (Buechley, 2017; Margolis, 2017; Resnick, 2017; Shapiro, 2017):

8 http://ecepalliance.org/resources/landscape-reports

> The most important challenge is relating computer science to [students'] culture and their identity. If you can get someone excited about something and engaged, they are incredibly motivated to learn. (Buechley, 2017)

Media Computation is a well researched college-level curricular approach that dramatically increases student interest and performance. Guzdial's introductory CS course focused on the design of relevant computational artifacts. This has doubled success rates and the impact was especially strong for female students (Guzdial, 2013, 2014). Even though Media Computation is mostly used in higher education, its curricular design principles are widely applicable. Guzdial argues that productive CS curriculum building requires four steps:

1. Figure out what has to be learned.
2. Understand the learner's motivations and goals, and make a significant effort to know what they are interested in and what communities of practice inspire them.
3. Find a context in which you can teach what has to be learned while respecting the learner's motivations and goals.
4. Assess the results and refine.

However, research has also unearthed other important principles for curriculum design in CS, as we review in the next sections.

Curriculum building principles. Researchers have been uncovering and deconstructing the typical assumptions that underlie the design of CS courses, to try to reveal hidden degrees of freedom in instructional design. A crucial dimension of design is how students will come into contact with the material. Pears' et al. (2007) review of the literature found three major approaches in how most CS courses are designed: (a) focus on generic problem solving, (b) focus on learning a particular programming language, and (c) focus on code production, or project-oriented CS courses. As we discussed previously, the focus on higher order problem-solving skills is problematic. Palumbo's (1990) review examined transfer between learning to program and problem-solving and concluded that more advanced forms of transfer (far or generalized transfer) should not be expected in introductory courses in CS, since typically there is no time to develop such skills. In other words, if curricula aim for the transfer of problem-solving skills to other domains, explicit time and effort should be put into it. The second approach, based on the learning of specific programming languages, is by far the most common. Textbooks, lesson plans, and assessments are designed based on the constructs of a particular programming language. This focus, common in introductory college courses and Advanced Placement (AP) classes in the U.S., has been criticized by several interviewees as being too limited and too vocational. Buechley, for example, praised new initiatives (such as the new Advanced Placement Computer Science Principles course) that are moving AP classes away from the "one language" model:

> So [Computer Science Principles] is a class that provides a different model of engaging with computer science than the traditional computer science AP class did. And a model that is much more focused on foundational concepts and big ideas as opposed to the nuts and bolts of programming in a particular language. And because of that, it has the potential to provide more accessible pathways to more diverse kids, which is really important. (Buechley, 2017)

The third approach is code production or project-oriented learning. Instead of small assignments and tasks based on language constructs, or more general problem-solving training, students learn to create more complex systems to accomplish a task through projects. Even though this approach is harder to structure and assess, it seems to be more aligned with the approaches advocated by most interviewees. Resnick, for example, advocated for a project-oriented approach rather than small puzzles or language-based activities:

> There are a lot of schools where they do something with coding but it is done very superficially, just learning a few tricks of how to put some blocks together…but not really connecting in a deep way. [CS should not be] just puzzles for kids learning to solve a problem, but a platform for expressing yourself. (Resnick, 2017)

Affinities between computer science and other disciplines. Early research in computer programming found that there are natural affinities between some topics in mathematics and programming but that not all mathematical topics can be successfully integrated into CS. Researchers found that the benefit of computer programming on traditional arithmetic skills is small (Butler & Close, 1989), but when lesson plans are redesigned to use programming as a way to explore rich mathematical practices, they can help students understand basic number sense, such as relationships between size of numbers and length of a line (Bowman, 1985). There is also evidence that the use of programming can help students understand variables and algebra (Carmichael, 1985), ratio and proportion (Hoyles & Noss, 1989), and Newtonian physics (Sherin, 2001). For diSessa (2017), the basis of the computational literacy argument is finding ways to unite subject matter and computational approaches (as in Turtle Geometry), rather than a creating a "forced marriage" between a given topic and the use of computation.

On the issue of CS learning supporting the development of general problem-solving and higher-order thinking skills, research has produced mixed (and mostly negative) results. In general, scholarship has shown that positive results in these areas require a high involvement from teachers and well-developed theoretical foundations (Clements, 1990; De Corte & Verschaffel, 1989), as well as considerable time investment. In one study, 150 hours of experience were needed to generate positive learning gains in problem-solving (Liu, 1997). Guzdial noted that this issue of programming and transfer is far from resolved, especially when the affinities and the unity of content and computation are not clear:

> Most people don't teach programming for transfer, and if they did, they would not be able to cover as much of programming. I think it is a zero sum game: Teach for programming fluency or teach for transferable problem-solving skills. You cannot get both in the same time. (Guzdial, 2017)

CS-inspired mathematics and science practices. Science and mathematics as professional practices have been deeply transformed by computation, both in terms of the core disciplines themselves and the creation of entirely new fields such as bioinformatics, computational statistics, chemometrics, and neuroinformatics. Efforts to improve and modernize the teaching of science and mathematics should include computation as a core curricular component. Skills that can be developed through CS-infused science and mathematics include the ability to deal with open-ended problems, the creation of abstractions, recognizing and addressing ambiguity in algorithms, manipulating and analyzing data, and creating models and simulations (Weintrop et al., 2016).

Most of the interviewees also identified infusing mathematics and science curricula with computation as a productive way to bring CS to classrooms. diSessa (2017) highlighted that "there are people deeply enmeshed in non-CS disciplines, yet sufficiently expert with CS ideas and practices, to really get this agenda accomplished now." And Grover stated:

> It is very synergistic...computation makes the science and the math more real, authentic, and engaging. Students see aspects of the discipline that they would not see in the static form of learning from a textbook. Conversely, computation becomes alive because of the context in which it is used. (Grover, 2017)

Some interviewees expressed skepticism as to whether there are a sufficient number of available CS teachers and whether it is possible to carve out space in the busy K–8 curriculum for a brand-new discipline. As a result, the interviewees noted that retraining science and mathematics teachers to add CS to their teaching and generating new accompanying CS-infused lesson plans might be a more sustainable approach to CSEd. Yongpradit (2017) also suggested that enabling teachers to receive dual certification in mathematics (or science) and CS might be a positive alternative approach for addressing the current CSEd teacher shortage.

Programming language choice and learning outcomes. Pears et al. (2007) examined the impact of programming language choice on learning. With the development of block-based languages (such as Alice, Blockly, and Scratch), research has been showing that child-friendly, block-based graphical programming offers many benefits to young learners when compared to text-based languages (e.g., Weintrop & Wilensky, 2015a), particularly in K–8 classrooms. Other approaches, such as CS Unplugged (Hermans & Aivaloglou, 2017), have shown positive results in introductory activities even without the use of computers. The research on Scratch and Alice are consistent with the design principles commonly expressed by Papert (1980) and Resnick (2017). They should be rich enough to introduce the fundamental CS concepts, have a small enough set of constructs and features to be learnable in a few hours or weeks, and allow for a variety of forms and domains of personal expression.

Because richness and simplicity are not easily combined, educators need to carefully consider the trade-offs when choosing a programming language. Fincher (2015) articulated these design principles in more detail, asking a crucial question: "How do we know how to reduce the complexity of programming languages while not curtailing students' future development in CS?" Weintrop and Wilensky (2017) have shown that students may perform better with block-based programming, but they see those languages as further away from "real" programming, so learning with simplified languages could limit their future development in CS. Fincher cited a 1960s experiment in literacy that created a simplified English alphabet to facilitate the learning of spelling in elementary school (the "Initial Teaching Alphabet"). Students did learn how to spell faster with the new alphabet, but could not make the transition to the normal alphabet, and it took them years to unlearn the modified one. Fincher finds a similar conundrum in CSEd: How can we know what to simplify, and how? Despite the work of researchers such as Weintrop and Wilensky, more studies are needed to understand how and under what circumstances learning with block-based or other simplified languages transfer to more traditional programming tools, and put students on a trajectory for more sophisticated experiences in CS. This seems to be one of the main research gaps in CSEd, but Resnick noted that the limitation of block-based languages is not always a problem:

> It is true that students planning to pursue a university degree in CS, or pursue a job as a professional programmer, need to make the transition from block-based languages to text-based languages. But it might be just fine for most other students to continue to use block-based languages. It depends on the goals of CSEd. Research on how to support students in progressively enhancing their fluency with block-based languages might be just as important as research on how to support the transition to text-based languages. (Resnick, 2017)

Another very important issue in programming language choice and design is domain-specificity. For example, the original LOGO language is especially well-suited for "body-syntonic" geometry (Papert, 1980). StarLogo and NetLogo, designed primarily for modeling emergent scientific phenomena in which multiple particles interact through simple rules, is a very good fit for some content areas in physics, chemistry, and biology (Wilensky, 1999, updated 2006, 2017; Wilensky & Reisman, 2006). An increasingly productive path for language designers is to tailor their languages to specific domains and support expression and problem solving within those domains, instead of creating complete languages. This approach has the advantage of reducing the complexity of the language and making it more readily learnable, even if it reduces their application as a general purpose programming environment (see also Wilkerson-Jerde, Wagh, & Wilensky, 2015).

7.3 Teaching and learning

Effective teaching and learning requires instructors to strike a balance between structured-activities and student exploration. Teachers need special training for CS teaching to help students notice connections to disciplinary content and make sure CSEd takes place in an environment conducive to collaborative work.

Teaching and learning refers to methods and the activities used to help students master the content and objectives specified by a curriculum. It encompasses the activities of both teachers and students in terms of pedagogical techniques, sequences of activities, and ordering of topics. In addition to the cognitive and developmental issues, researchers have also focused on identifying productive CS-specific teaching and learning strategies and identified significant differences between CS and other disciplines, in that:

- CSEd requires students to use computers and due to logistical or design issues, often demands that students share the same equipment and collaborate.
- CSEd frequently involves long-term, complex projects that span multiple classes and could take a toll on students' cognitive load.

- At the K–8 level, some researchers warn that teachers must consider the availability of computers in students' homes when assigning homework outside of class. While teachers want to engage students and motivate them to continue working on projects, they need to know if students have the needed infrastructure at home. Consequently, models that rely heavily on instruction during class and independent work at home might not work with CS for all students.

The following sections address some of the findings regarding pedagogical elements.

Pedagogical strategies and the role of teachers in CS classrooms. Some studies suggested that self-guided exposure to computing without purposeful teacher or curricular facilitation results in little learning (Pea, Kurland, & Hawkins, 1987). Clements and Meredith (1993), for example, explain that despite the apparent connection between programming and mathematical thinking, many students tend to rely on purely visual cues given by the computer to infer rules and avoid analytical work (see also Hillel & Kieran, 1987). And while in Clements' study, visual problem solving helped students with math problems in the beginning, over time it prevented them from arriving at mathematical generalizations unless teachers presented them with tasks that required an analytical, mathematical approach (Clements & Meredith, 1993). Grover and Basu (2017) found that exploratory activities in block-based programming environments without competent teacher facilitation do not address misconceptions about basic CS concepts. Other researchers have found that learning mathematics using programming tools does not lead to substantial learning unless there is effort to direct students' attention to mathematical analysis (Hoyles & Noss, 1992). Grover, Pea, and Cooper (2015, 2016) found that even when students go through a designed curriculum in block-based programming environments, they struggle with concepts such as loops (terminating based on a Boolean condition) and variables much more than other concepts. However, with the correct guidance, computer-based exploration in mathematics shows promise compared to work in other media. Hoyles, Sutherlands, and Noss (1991) found that, compared to a paper and pencil or spreadsheet activity, a well-designed collaborative unit with computer programming led students to more frequently use formal mathematical language. However, Margolis (2017) mentioned that there is a significant variance in how well teachers can do this facilitation work, and that the PD programs that foster that kind of competent guidance are often inaccessible for less affluent public schools.

Cooperation, collaboration, and pair programming. Research indicates that students seem to cooperate and collaborate more when working with computers because they often disagree and therefore spend more time discussing their solutions. Most of the disagreements detected by researchers were about *ideas* rather than *social issues.* Therefore, productive, on-topic conflict has been identified as a positive aspect of programming in classrooms (Lehrer & Smith, 1986; Nastasi, Clements, & Battista, 1990). Also, some online programming environments now make it easier for learners to "see inside" their projects, remix, and build upon one another's projects. Some environments, such as NetsBlox, even allow for collaborative programming, where multiple students can edit the same project synchronously. These tools have important implications for research and practice.

Pair programming is a widely-used and well-researched strategy that also builds on collaborative practices. This pedagogical practice derives from the literature on collaboration and more specifically on computer-supported collaborative learning (Suthers, 2006). Collaborative computer programming has drawn considerable attention since research has shown that 70% of programmers' time is spent in collaborative endeavors, while in most CS courses (and their assessments) students work alone. In pair programming, the "driver" types at the computer and the "navigator" completes a variety of tasks. Ideally, there is strong communication between drivers and navigators and roles are constantly switched. Results of studies of pair programming indicate that students working this way produce better quality code, perform better on graded assignments, and experience higher levels of self-reported satisfaction and confidence. The research on pair programming also raises concerns with regard to its use in K–8 environments. First, researchers have tried to incentivize collaboration (students tutoring each other and accomplishing tasks together) but have struggled with supporting cooperation (working on different parts of the assignment then merging the work later). This is a concern because students may specialize in different tasks, lose perspective of the whole, or not venture into more complex parts of the project. Another concern has been the tendency for some individuals to dominate the work. According to Shapiro:

> Research on pair programming in K–12 shows that while it can be beneficial, it can also exacerbate power dynamics that can marginalize students. (Shapiro, 2017)

Finally, studies have shown that pair programming should be explicitly taught in teacher preparation programs because it provides advantages for CSEd instructors by incentivizing students to help each other without necessarily relying on instructors except when necessary (Bevan, Werner, & McDowell, 2002; McDowell, Werner, Bullock, & Fernald, 2002; McDowell, Werner, Bullock, & Fernald, 2006; Ruvalcaba, Werner, & Denner, 2016).

Scaffolding complex programming tasks. Guzdial (1993) showed how different parts of the programming process can be scaffolded and discussed the ways that students chose to use scaffolds. One of the well-known approaches to scaffolding is subgoal labeling. A well-known difficulty in science education (especially when using worked examples) is how to help students focus on the structural features of a problem instead of superficial aspects (Chi, Feltovich, & Glaser, 1981). Anderson, Farrell, and Sauers (1984) found that the same happens in CS instruction when using worked examples: Students might not understand the fundamental and structural characteristics of the task at hand and might get lost in contextual elements. Additionally, the technique of worked examples in CSEd could increase cognitive load (Lister, 2011) because it requires students to simultaneously learn to program, learn a new programming language, try to problem solve, and work in an environment that is different from normal classrooms (Morrison, Margulieux, Ericson, & Guzdial, 2016).

In the more general educational research literature, the technique of using worked examples and breaking them up into subgoals improved students' performance, but only when structural versus superficial information about the task were clearly differentiated (Catrambone, 1998). Researchers have adapted this approach to computer programming with positive results. An influential study compared conventional worked examples and subgoal-labeled work examples. Instead of a simple set of step-by-step instructions (e.g., "click on block A," "drag block A," "connect block A to block B"), the worked examples condition provided a simple label before each group of instructions (e.g., "handle events," "set properties," "create new objects," "set output") alongside information about the purpose of the subtask. Students in the subgoal label condition performed better in every measure of problem-solving performance (Margulieux, Guzdial, & Catrambone, 2012; Morrison et al., 2016).

7.4 Teacher development

Integrated systems of teacher certification, PD, and incentives should be in place and inclusiveness should be a priority in both pre-service and in-service programs.

Ultimately, the interactions between teachers and students in individual classrooms are a determining factor in whether students learn CS successfully. Thus, it is not surprising that the interviewees expressed the belief that teachers are the linchpin in any effort to implement or change CSEd. To truly support implementation of CSEd, the preparation, effective development, and retention of CSEd teachers will need to be prioritized.

Teacher development was a central concern for most interviewees. Clancy, Margolis, and Yongpradit (2017) highlighted the challenges in building the CSEd teacher workforce for CSEd, and noted the need for teacher certification, training programs based on these certifications, and incentives for teachers to seek these qualifications. Guzdial (2017) highlighted the importance of pre-service teacher development as the most viable way to sustainability.

The need for equity in teacher development was also highlighted, since more affluent schools are more capable of offering high-quality programs. Interviewees noted that it is not enough to expose teachers to CS content. Teachers need time to practice inclusive CS and these pedagogies should be interwoven into the entire teacher preparation program. Margolis (2017) also raised the need to educate teachers regarding biases, so that they can reflect on belief systems and perceptions about which students can excel in computing, and how these beliefs would impact their relationships with students.

In general, there was concern about the rapid scaling of several CS initiatives and the capacity to prepare thousands of teachers adequately in a very short time. The interviewees argued that scaling too quickly disproportionately impacts underserved communities and populations that are historically excluded from STEM.

Margolis was particularly concerned with making equity a core tenet in teacher development, mentioning that in her research she encountered significant variability among teachers in their capacity for guiding deeper cognitive thinking. She found that teaching was particularly productive when teachers identified the specific CS concepts for the students while they were learning them and discussed how they could relate the concepts to other areas of knowledge. The capacity to competently guide students in this way was found to be a predictor of student learning but it varied considerably among teachers. Not surprisingly, teachers in less affluent areas were found to be the least prepared to enact these strategies in the classroom, in part because their districts had less funding for teacher PD. Margolis adds:

> Not only do teachers need to be introduced to the CS content, but they need to have time practicing pedagogies that are aimed at creating an inclusive CS learning environment, building on the assets, interests, and motivations of traditionally underrepresented students. Also, CS teacher PD must have equity and inclusion woven throughout everything that happens in PD, not just isolating this issue to a discrete one-hour discussion. For instance, as teachers are experiencing teaching lessons during PD, the other teachers who are in the roles as students or observers should be reflecting on their own experiences of inclusion (or not), thinking about their own students in their classrooms, and what works (or does not) to ignite the interest of all students. Also, teachers need time, and a safe learning environment, to reflect on all the biased belief systems associated with which students can and cannot excel in computing, to reflect on their own belief systems, and how belief systems impact their relationships with the students in their classrooms. Traditionally CS education has not been a place where these types of discussions or reflections have taken place, but they must if we are to broaden participation in computing. (Margolis, 2017)

Guzdial also emphasized the importance of pre-service teacher development:

> We do not reach sustainability with in-service teacher development, though that is where most efforts are today. Pre-service is the sustainable path to a supply of well-prepared teachers, and it is the path that the rest of K–12 disciplines follow. (Guzdial, 2017)

8. Learning Progressions and Learning Issues

Learning progressions are descriptions of successively more sophisticated ways of thinking and how learners develop understanding of key concepts and practices within and across multiple grades. Learning progressions can be used to help designers build coherent curriculum and align standards, curricula, and assessments across grades and grade bands. This section covers the key findings in the areas of learning, cognition, and learning progressions, discussing mental models, misconceptions, and developmental approaches to CSEd.

8.1 Mental models of what computers do: the "notional machines"

Having an apt mental model of what computers can and cannot do, and how they execute code is a prerequisite for effective CS learning. Educators and designers should therefore be careful with the analogies and metaphors used to explain what computers do. Young learners have difficulty tracking events, variables, and states that are not visible.

In school subjects like mathematics or physics students can superficially solve problems with little conceptual understanding by figuring out the variables of interest and plugging values into well-defined formulas. However, in CS students need to have a well-developed mental model of what a computer does when it executes programs, or "an abstraction of the computer that they can use for thinking about what a computer can and will do" (Guzdial, 2015). Benedict du Boulay (1986) called this abstraction a "notional machine." To understand notional machines is not to simply know what computer hardware is. Notional machines are language-dependent, since each programming language behaves in a different way and demands different reactions from the computer. Guzdial stated that understanding the correct notional machine for the language at hand is a key learning goal within CSEd, and indeed there is considerable evidence that the level of development and correctness of children's notional machines predicts how well they learn to program.

It is difficult for young learners to develop an accurate understanding of notional machines because they are quite removed from everyday experiences (du Boulay, 1986; Guzdial, 2015; Sorva, 2012). As Guzdial explained:

> The notional machine is unnatural for us. The inhumanness of computers makes them harder to understand…The computer is a non-human agent that is doing what was specified, and not what was intended. (Guzdial, 2017)

Researchers have conducted extensive studies on how students form mental models of how computers execute code. They have concluded that these models go awry when students' intuitive understandings about programing go unchecked, or when teachers present students with inadequate metaphors (Ben-Ari, 1998; Perkins, Schwartz, & Simmons, 1988). For example, the intuition that programming is a conversation with a human-like creature—capable of inferring meaning that is not explicit in the code—is a well-known source of problems (Bonar & Soloway, 1983). Pea (1986) found related misconceptions around sequence of execution and parallelism (all lines of code active at the same time), intentionality (the program has goals and can see what is happening to itself), and the notion of a "hidden mind" inside the machine. The lack of understanding about notional machines generates other well-documented difficulties in learning CS, like how instructions are executed in the state created by the previous instructions, or that variables can only have a single value at a time (du Boulay, 1986; Sajaniemi & Kuittinen, 2008; Smith & Webb, 1995; Sorva, 2012).

8.2 Misconceptions and learning challenges in specific programming constructs

Teachers should be aware of CS misconceptions and interactions with other disciplines when they design and deliver instruction. For example, the way students learn about variables in math might affect their understanding of variables in CS. In addition, the accomplishment of simpler CS tasks does not entail the overcoming of programming misconceptions.

du Boulay's work was one of the first systematic attempts to understand the specific issues and misconceptions of CS learning. This work revealed that learning to program was much harder than anticipated by the pioneers of CSEd. Even at the college level, several large scale international studies showed that introductory courses were largely failing to generate the desired learning outcomes (Lister et al., 2004; McCracken et al., 2001). To try to understand the problem, du Boulay (1986) systematized students' difficulties into five overlapping domains:

1. General understanding of what programs are and what can be done with them;
2. Students' model of the computer as it relates to executing programs (notional machines);
3. Notation, syntax, and semantics of programming languages;
4. Structures, schemas, and plans; and
5. Pragmatics, the skills of planning, developing, testing, and debugging.

This categorization was useful because it was found that "the shock of the first few encounters between the learner and the system are compounded by the student's attempt to deal with all these different kinds of difficulties at once" (du Boulay, 1986, p. 284). This signaled to instructors that these multiple dimensions had to be dealt with in order to improve teaching and learning.

This seminal work led researchers to go deeper into these different categories, detecting difficulties involved in learning each of the five core CS domains. For example, Spohrer and Soloway (1986) showed that loops and conditionals generate more bugs than other types of operations (such as input and output). Soloway, Adelson, and Ehrlich (1988) found that novices preferred a "read then process" approach to writing loops rather than a "process then read" one because internal changes in the system are invisible to students. Samurçay (1989) showed that students are better able to update than to initialize variables, and Lewis (2012) found that debugging performance in middle schoolers was more correlated with their understanding of the system's state than with knowledge about how to debug. Other studies looked at assignments, print statements, control flow (Sleeman, Putnam, Baxter, & Kuspa, 1986), parameter passing (Fleury, 1991), and recursion (Bhuiyan, Greer, & McCalla, 1990; Booth, 1992; Kahney, 1983), always discovering new classes and variations of students' misconceptions. Juha Sorva's review of this research actually found as many as 162 programming misconceptions and obstacles (2012). diSessa (1985) generalized the notion of "notional machine" to "structural models," and also identified two other classes of models ("functional" and "distributed") that are important for understanding programming. He connected these considerations with the design of comprehensible and flexible computational systems. The co-presence of multiple models and their interactions provides an alternative to stage-like developmental approaches to learning programming (as described in the next section).

In general, these pioneering studies from the 1980s and 1990s reported that, surprisingly, successful completion of simple programming tasks is not correlated with the understanding of even simple core CS concepts: it is possible to complete these tasks without correct conceptual understanding of key programming constructs—a finding that has been attributed to the lack of appropriate mental models about what computers do (Sorva, 2012).

More recently, researchers have focussed on understanding in detail how students learn basic CS concepts. Stefik and Siebert (2013) studied programming language syntax and how it affects learning. They created a language called "Randomo" that used random symbols and keywords and tested it against a variety of well-established languages. They found that languages using more traditional syntax (such as C or Java) were as hard to learn as Randomo, but that languages with more modern syntax and more intuitive keywords (such as Python and Ruby) were significantly easier to learn. The authors concluded that the choice of keywords (e.g., "repeat" to start a loop instead of the less intuitive "for") is highly consequential for novices (see also Robins, Rountree, & Rountree, 2003).

Grover and Basu (2017) examined sixth, seventh, and eighth graders using a block-based language and reported unexpected problems related to variables. They found that a significant number of students did not understand that a variable name could be longer than one character, so when they encountered long variable names, some of them believed that the name of the variable was a command. The authors attributed this misconception to an interaction between the way students learn math and CS in school, since in math, variables are always represented as a single letter and stand for an "unknown."

Franklin and collaborators (2017) confirmed that there is a mismatch between programming environments and prior mathematics knowledge regarding the inclusion of negative numbers and decimals for upper elementary learners. They also found that there can be considerable differences in preparedness for learning CS between the fourth and sixth graders they studied. Specifically, younger students (fourth and fifth grade) found it challenging to initialize variables, and sixth graders were significantly more precise at navigating in two dimensions than their younger counterparts. There is also a growing body of research showing that visual block-based tools could be more effective and engaging for students. However, there is still no agreement on exactly how to disambiguate the benefits of block-based languages in terms of how they address problems with syntax, semantics, and mental models. Some researchers mention that students "can see blocks as inauthentic, which can be demotivating if one's goal is to develop an identity. On the other hand, the syntax benefits can reduce frustration, which can support engagement and motivation" (Shapiro, 2017). Researchers have been working on ways to scaffold the subsequent transition from blocks to text, which was shown to be problematic unless the block to text transfer is explicitly mediated for (Dann, Cosgrove, Slater, Culyba, & Cooper, 2012; Grover, Pea, & Cooper, 2014). As a result, researchers are successfully experimenting with hybrid environments to ensure a smooth transition (Weintrop & Wilensky, 2015a, 2015b, 2017). These types of studies, which examine very specific misconceptions related to age groups within K–8 education, are becoming more common in CSEd conferences. Researchers, however, are focused on not just detecting those misconceptions, but also looking for ways to design instructional strategies to overcome them, as we will see in subsequent sections.

8.3 Schema building and developmental approaches to CSEd

In K–8 CSEd, students' developmental stage is a determinant of learning outcomes, and teachers must help students transition between stages. The acquisition of expert-like behavior for CS problem solving involves exploring many programming problems and cases and building one's arsenal of schemas.

Several researchers have examined student misconceptions and how best to overcome them and have concluded that most successful approaches make use of the vast literature on cognitive development and learning sciences. Lister, for example, proposed using developmental psychology as a template for this exploration:

> Piaget's crucial observation was that children do not simply know less than adults, or that children believe things that are wrong. Instead, children think differently from adults… Adults (including academics) inexperienced in teaching children, communicate their knowledge in ways that children are not yet ready to understand. (Lister, 2016)

Inspired by Piaget's stages, Lister devised a developmental trajectory for CSEd with four levels (2016):

1. *Sensorimotor or pre-tracing stage:* The novice programmer has an incoherent understanding of program execution.
2. *Preoperational or tracing stage:* The novice can manually execute ("trace") multiple lines of code.
3. *Concrete operational or abstract tracing stage:* The novice programmer reasons about code deductively. Students show a purposeful approach to writing programs.
4. *Formal operational:* The expert performs at this level. Students can reason logically, consistently, and systematically.

This approach is a good example of how findings in education and human cognition can guide CS teaching and learning. Based on extensive empirical work in K–8, Lister (2016) offered some valuable hints for classroom situations—most of which would sound counterintuitive for CS instructors unfamiliar with developmental psychology and learning research:

- "As the novice programmer learns, there are periods of time where the novice maintains their existing mix of stages, even when the novice is taught something new." They "swap between conceptions, correct or otherwise, based on superficial aspects of the code that happen to be in view at the time."
- "Teachers should understand that pre-tracing students might have wildly different understandings of basic CS concepts, and that it is part of a normal developmental trajectory. For example, "Why should a sensorimotor programmer believe the '=' sign always means the same thing when some symbols in programming (e.g., '*') change meaning between contexts?"
- "Using a strategy of "repeat-trace-patch-until-success," preoperational students may eventually succeed in producing correct solutions to small programming problems, but they will only do so after considerable time." Lister noted that "preoperational programmers should only write code when closely supervised."

The developmental approach surfaces another important question in CSEd: How do young students acquire expert-like behaviors, and what are those behaviors? To answer that question, researchers have analyzed expert programmers and tried to distill productive behaviors. Results were counterintuitive: whereas some expected that experts would always use top-down, systematic approaches to problem solving when programming, studies found that they use both top-down and bottom-up approaches (Visser, 1987) and transition between systematic and exploratory behaviors. Adelson and Soloway (1985) and Rist (2004) tried to explain this finding using the idea of schemas, or templates, for recognizing and solving problems. Rist claimed that experts, having knowledge of more kinds of problems (or richer schemas) can easily pick the right strategy for a given problem, doing a breadth-first mental search. For beginners, "programming problems will be unfamiliar and involve slow, difficult, and error-prone… depth-first development. From an educational point of view, the growth of expertise is marked not by adding top-down strategies to one's arsenal, but by being able to use top-down strategies as a result of growing familiarity with problem types and their solutions." (Sorva, 2012)

9. Advancing CS Education through Research

The interview data and literature pointed to important research directions for CSEd. The following sections describe these directions and their justifications. They also assess the current CSEd research capacity and inform recommendations to strengthen research paradigms and the research base itself.

9.1 The current CSEd research base

Important next steps in CSEd research include determining how to systematically overcome its challenges and gaps, growing the field to match the challenges of large-scale implementations, attracting more researchers, making their work sustainable in universities, and making research deeper, more productive, and faster.

The need for more research to enable successful implementation. Guzdial, Shapiro, Margolis, Fincher, Sentance, and others were adamant that more research funding is needed for CSEd. Fincher also noted that it is challenging to convince funding agencies to finance the work because of the uniqueness of CSEd research:

> In the U.K., CSEd research is not seen as "science," so the science funding councils are not appropriate and the social science funding councils say scientists don't have the appropriate methodologies, so they won't touch it. (Fincher, 2015)

According to Guzdial (2017), NSF funding for CSEd is mostly provided for curriculum or broadening participation rather than for fundamental research on how people learn CS: "The kind of work that I have been doing is to look for educational psychology principles, and how they apply to CS learning. There is no program at NSF that will explicitly fund that kind of work." The interviewees also agreed on the need for more stable funding sources and programs to support research on the large-scale implementations currently underway.

CS is a young and rapidly developing field, which makes CSEd unique among traditional school disciplines. CSEd is the first new major subject to roll out in many school districts for decades. The data revealed little agreement about what CS topics are important in K–12 education or how to address them. There was also no clear consensus on how these topics should be introduced to students (via either concepts, ways of thinking, or the details of specific programming languages). In addition, because CS languages and tools are constantly changing, it is difficult to accumulate relevant research results. For example, although there was extensive research on LOGO programming in the 1980s and 1990s, it is unclear if these results can still be applied today given the different programming environments now used in schools. Many researchers are trying to make their claims and research questions more generalizable and less focused on one language but this is still a nascent effort. Fisler, for example, had students solve the same problem in different languages, trying to determine what results were language-bound and which were invariant (Fisler, 2014).

In some cases there are instructional strategies that are becoming standard across languages and tools due to an accumulation of evidence as to their efficacy in different circumstances and contexts (Guzdial, 2015). The strategies, however, are not abundant and interviewees were unanimous in their contention that we are still far from having a solid corpus of research in CSEd (especially in K–8) or anything comparable to what exists in mathematics or science education.

The interviews also revealed that there is much more research on CSEd in high school and college environments than in K–8. This is one effect of the history of disparity in funding and challenging research logistics:

- Large-scale CSEd in K–8 is much more recent than in high school and higher education.
- There is a bias towards research in higher education as opposed to K–12 schools as most university professors in CS departments find it more convenient to research their own students. Universities also more readily fund studies to improve their undergraduate courses.
- Obtaining approval from institutional review boards and school districts is challenging, especially if there is any form of electronic tracking of students' work, which is typical in many modalities of CS research.
- K–8 schools and classrooms tend to be smaller and thus comparative studies are more difficult to design and have less statistical power.

- Despite recent advances, access to computers and broadband (crucial for CSEd experiments) in K–8 public schools is still uneven.
- Disciplines such as mathematics and science, with their long history in K–8 education, have a substantial contingent of trained teachers, teacher training programs and materials, and a vast infrastructure in schools and districts. Compared to these disciplines, the funding and infrastructure for K–8 CSEd and research is still exceedingly small.
- CS teacher development might require a different set of strategies. It is commonly assumed that CS teachers need to be proficient at writing software, but Shapiro (2017) noted that "that may not be the case. Mathematics and science teachers are required to have degrees in the discipline. That may not be necessary for CS: What constitutes sufficient CS training may be different than what has historically been required for those disciplines."

Fortunately, CSEd conferences are now beginning to focus more on K–8 education uncovering a new class of misconceptions, pedagogies, tools, and learning issues (the following sections review several of these papers).

Research into CS concepts. Unlike physics and biology, CS does not yet have stable concept inventories with agreed upon concepts and age-appropriate metrics or a sufficient repository of language-neutral assessment (Taylor et al., 2014; Tew & Dorn, 2013). As a result, some types of research designs are very challenging in CS. In addition, CS is not a natural science like physics and so it is more difficult to create concept inventories in CS given that there is no consensus regarding the lists of concepts. The most well-established concept inventory in physics (the *Force Concept Inventory,* Hestenes, Wells, and Swackhamer, 1992) took years of refinement and testing. Developing such concept inventories for CS, however, would enable research designs that compare different pedagogies for the same content topics or concepts. So while many researchers and national organizations are trying to model the CS standards after the Next Generation Science Standards (NGSS), some contend that there are limits to such a process because the epistemology of CS is uniquely different.

Research into the design of programming tools and experiences. Many interviewees pointed out the lack of a productive feedback loop from empirically supported findings to the design of programming tools and experiences:

> There is a lot of myth and a lot of happy stories but in terms of best practices, we have very little at K–8. There are certainly a lot of people using Code.org tutorials or Scratch but there's very little evidence about what happens and the quality of the learning in those kinds of settings. (Shapiro, 2017)

For example, it has been known for years that variable initialization in most block-based languages is difficult and the interface designs do not work well. Although this has been known for quite some time, language designers do not seem to use evidence to drive tool refinement, claiming that the development of CSEd software tools are rarely guided by systematic research into the kinds of concepts that are intrinsically challenging, and which of those challenges are about the particularities of the tools that we are using (Shapiro, 2017).

While this is consistent with our review of the field, it seems that this reflects a larger issue of a lack of productive connection between researchers, tool designers, and implementation developers (see, as an example of a cycle of design-based research, diSessa & Cobb, 2004; Ericson, Rogers, Parker, Morrison, & Guzdial, 2016). Weintrop and Franklin are examples of scholars who are conducting rigorous studies and finding out more about how to redesign tools and their use. Other researchers, however, feel powerless in the face of the big organizations that are driving CSEd today. It is difficult to know to what extent the major initiatives that have been funded would fundamentally change their programming languages or pedagogical approaches when faced with counter evidence from research. Many researchers expressed concern that their work would be just "noise," and that most of the design and high-level strategy decisions tend to be driven by other agendas. Establishing the routine of internal and external evaluations to validate and inform program changes and having venues and public spaces in which the results can be communicated and discussed would help ensure evidence is heard and counted.

Research into tools for formal learning environments. Despite the widespread use of several CS tools in formal learning environments, the interviewees articulated the following shortcomings:

- Most tools are not designed for classroom use and only a few have classroom management features or dashboards. Even fewer have tools for managing and assessing complex project-based work, which is a labor-intensive task in CSEd. Developing new tools or incorporating these functionalities into current software would greatly help teachers better manage classrooms.
- There are only a few tools that easily and efficiently facilitate the incorporation of CS into other disciplines. Apart from some well-established projects such as NetLogo (for science classrooms), teachers in arts, science, history, social studies, or even mathematics would have a challenging time finding classroom-ready tools for CSEd (with some exceptions, such as Bootstrap for mathematics or physics). The development of these tools would be a significant contribution to K–8 CSEd.
- Physical computing tools such as Arduino are popular in schools but are not designed for children. However, child-friendly platforms such as Lego Mindstorms and Hummingbird as well as low-cost options such as Microbit, Makey Makey, and Gogo Board are starting to make their way into classrooms. Resnick, Horn, and Shapiro believe that this is an important and fertile area for development and that there are many unexplored opportunities for programming objects in the physical world.

Research into other forms/paradigms of programming (machine learning, concurrent programming). The majority of software tools used today in K–8 employ the block-based programming paradigm (e.g., Scratch, Alice, Blockly-based languages), but interviewees mentioned several new and emerging approaches to programming that are still far from classrooms. These approaches include parallel programming, machine learning, flow-based programming, spatial computing, and "programming by example." Researchers also mentioned the importance of bringing new ways of interacting with the world into CSEd. These could include programming physical devices, web services, and new media forms. Traditional funding sources such as the National Science Foundation do not typically fund the development of software tools. Consequently, several interviewees stressed the need for ongoing funding streams for language/tool/curriculum development, so that the tools of the field evolve alongside CS and help diversify students' experiences.

Research into CS in the arts/creative computing. Interviewees pointed to a lack of tools for the "A" in STEAM learning (STEAM stands for science, technology, engineering, art, and mathematics), noting that there is an overwhelming concentration of resources in "STEM" tools and little funding or development for tools for the arts or creative expression[9] and this impacts what happens in classrooms. The LilyPad platform is an exception that provides indications of how important such developments could be (Buechley, Eisenberg, Catchen, & Crockett, 2008). Seed funding for tools and creative computing and arts was therefore posited as a productive direction for CSEd (Buechley, 2017; Margolis, 2017; Shapiro, 2017).

9.2 Developing a new strategy for CSEd research

Developing a new research paradigm for CSEd could help solve some of the challenges outlined in this report. To get there, we need more attention placed on making CSEd research a more stable and well-funded enterprise that will help to advance the field for years to come.

Defining a unique research paradigm for CSEd. The definition of a CS-specific education research paradigm seems to be a first and necessary step toward establishing CSEd as a stable research enterprise. This paradigm for CSEd research should replicate the rigor and methods of mathematics and science education, educational psychology, and learning sciences since many of the findings reported in CSEd research are not unique to CS as a domain and have long been studied in educational and cognitive research (Fincher, 2017; Sentance, 2017). Also, learning theories and pedagogies from the learning sciences could be especially useful as those take into account both cognitive and socio-cultural aspects of learning (Grover, 2017).

9 Some existing tools, such as Scratch or Alice, allow for the use of multimedia, graphics, and other tools for expression, but the general focus in CSEd is still concentrated in STEM disciplines.

There are, however, significant differences between CS and these disciplines. Mathematics and science education have a longer history and a more stable set of topics and tools. The laws of physics are not changing anytime soon and the representational forms in mathematics (such as algebra, or differential equations) typically take centuries to change. Conversely, being a "science of the artificial," CS content and tools can change radically in just a few years (Shapiro, 2017). Programming languages as we know them now could be radically transformed in 10 years, making much of the narrower, language-specific research obsolete (Horn, 2017). Mainstream programming will likely incorporate new paradigms (such as machine learning) making traditional coding less relevant. Consequently, simply transporting research and curricular frameworks from other disciplines is an insufficient strategy given the unique characteristics and epistemologies of CS. A crucial task for researchers and practitioners in the field will be to adapt existing paradigms and frameworks to create the pillars of a robust research and deployment program. The following sections provide some of the details involved in this adaptation process.

CSEd should be a stable, academically valued, and well-funded enterprise. As Guzdial (2017) pointed out, the number of CSEd graduate students in the U.S. is very small (previously estimated around 20). Very few computer science departments have tenured professors that do research exclusively on CSEd. Research on CSEd is not valued as much as pure CS fields for tenure and promotion, and there is only one chaired professor in the country dedicated to the topic. This lack of incentives relegates CSEd to a secondary activity for CS professors. CSEd-centric federal programs such as NSF's Broadening Participation in Computing Alliance Program (BPC-A) depend upon vocal leaders within the government and the NSF, and these programs are not permanent. For example, the NSF's Cyberlearning and Future Learning Technologies (Cyberlearning) program, which has funded many CSEd projects since 2011, was phased out in 2017; and the STEM+Computing program which funded research focused on integrating CT into STEM learning has been phased out after three years in 2018, with programs such as CSforAll:RPP being given precedence.

The interviewees pointed out that in many CS departments, CS professors downplay the importance of educational research for CS instruction (Sentance, 2017). And because CS is still an elective discipline in most countries including the U.S., schools of education place less value on CS than on mainstream disciplines such as language arts or mathematics. As a result, CS or Education Ph.D. students specializing in CSEd have a difficult time finding tenure-track positions in both types of schools. diSessa mentioned the example of physics education as a possible model:

> Twenty years ago, physics education was downplayed and of low status in physics departments. Now there is a recognizably important set of places where physics education has taken deep root in physics departments. This all takes concerted effort and a rather long timescale. This should be a project of persistent concern, effort, and funding. There is also a movement concerning "discipline-based research" in education, where various discipline specific faculties are trying to find ways to generalize and combine insights. (diSessa, in press)

Interviewees were adamant as to the need to incentivize universities to create those programs and fund them in sustainable ways. One concrete suggestion was the creation of five CSEd-endowed chairs at prestigious universities. Such chairs would cost between $1–2 million each, but since the endowed professors could be expected to fundraise on their own after the initial funding, the impact of such an initiative could last for 20 or 30 years at a very low cost.

Creating an achievable, innovative, and actionable research agenda for CSEd for the next decade. Mirroring the content coverage achieved by science or mathematics education (SMEd) research does not seem to be a productive path for CSEd given disciplinary differences. It will be impossible to research the learning of all important CS concepts for all age groups quickly enough to guide the several large-scale implementations that are now underway. Along the same lines, the content of CS itself changes more frequently than that of other disciplines. A more productive path would therefore be to bring together educators and researchers with diverse perspectives to create a paradigm that reflects the uniqueness of CSEd and supports a long-term research program. This paradigm would differ from science education. For example, while it is almost impossible to automate data collection for education research in a traditional science lab, it is relatively easy to instrument programming environments to log users' actions as they program (even though there are still challenges such as privacy and long-term data tracking). These tools could expand the types of studies possible in CSEd and reduce the need for, and prevalence of, controlled studies with just a few dozen students. Instead, such tools would enable less costly studies with more subjects and greater statistical power.

Some interviewees, however, expressed concern over the optimism for learning analytics. diSessa, Shapiro, and Resnick (2017) noted that previous efforts on the use of log files for understanding student learning in CS and science education generally were shown to have more limited usefulness than initially imagined. The interviewees were also concerned with the rise of automated assessments as a possible byproduct of these instrumented programming environments due to their low cost and novelty, and with the possibility that automated techniques would overshadow deeper types of students assessment (e.g., portfolios) that have been shown to be more informative and useful, especially at the K–8 level. They maintained that while automated techniques might be useful for some types of research, their use for direct student assessment should be viewed with extreme caution.

There are other types of data and usage strategies that would be important in the creation of a CSEd research paradigm. When students are working on a CS project, their thinking and debugging processes are often directly observable. This opens up possibilities of very detailed qualitative, microgenetic studies on the learning of CS. The fact that CS work is often done in a project-based fashion also creates new possibilities for more holistic measures such as portfolios or artifact analysis.

The design cycles in SMEd are quite long because the most typical research designs used in these disciplines require a large amount of time for implementation, data collection, design, and redesign. Also, the content in SMEd is relatively stable. In CSEd, it might be that research and redesign will take place in much shorter cycles, following a design-based research approach (Horn, 2017; Shapiro, 2017). SMEd are also very connected to traditional research paradigms from educational psychology and ethnographic methods. It will be essential to incorporate those methods into CSEd, but as studies increasingly incorporate automatically collected datasets and tools from machine learning, it will be crucial to come up with ways to combine these diverse data sources in meaningful ways. This amalgamation will require training for CSEd researchers to enable them to incorporate data mining, design, cognitive science, and human computer interaction.

Given all these characteristics of CSEd research, it might be that the ultimate goal of the field will not be to have answers for teaching each concept at every grade level, but rather a set of more general and adaptive principles and very agile tools and methods to test pedagogies, tools, and curricula in a more iterative way than other disciplines. Consequently, a better short-term agenda for CSEd might be not to do "definitive" studies on particular concepts such as conditionals or loops, but instead to create infrastructures, tools, and methodological paradigms that could be as adaptable as CS itself and could accommodate well-established mixed-methods educational research frameworks such as design-based research and action research. Implementing this kind of solution would require investing not only in empirical research, but also in:

- The instrumentation of current programming platforms, allowing data to be automatically collected (respecting institutional review board (IRB) regulations and privacy concerns) and the creation of software front ends as easy to use as SPSS or Excel that would allow educational researchers to analyze large CSEd logfiles and datasets.
- The creation of common data repositories to allow standardization, replication, and reuse of data. For example, such a repository could mimic Carnegie Mellon University's DataShop and be expanded to include qualitative data, interviews, protocols, and coding schemes. This would allow multiple researchers to use the same dataset to run multiple studies on CSEd, or for some research groups to specialize only in data analysis from secondary sources (an approach that has been very productive in economics and many other disciplines).
- Definition of diverse and inclusive research paradigms encompassing a variety of methods, from data mining to holistic measures such as portfolios, tackling topics that still invite further research such as conceptual learning, learning progressions, general CS skills, and studies on the growth of computational literacies.

- Understanding the applications, limitations, and potential combinations of multiple data types and analysis techniques, from data mining to ethnographies, giving all research traditions an appropriate degree of status and voice.
- Creation of training programs (boot camps, graduate-level courses) for current researchers or doctoral students to learn new qualitative and quantitative research methods.
- Special events and PD programs for popularizing research results to schools and practitioners, in which CSEd researchers would also get more familiar with well-established educational and research paradigms.

10. Summary of Findings

The year 2017 marked the 50th anniversary of the LOGO programming language. In just five decades an entirely new domain of knowledge evolved from an idea in the minds of a few visionaries to national public policy. And while CSEd is a relatively new discipline with a less substantial research base, there is much reason for optimism. Ensuring that we continue this progress, however, requires the commitment, work, and flexibility of a large number of stakeholders. We are now facing the growing pains intrinsic to progressing from pilot projects to large-scale implementations and we must look and work beyond these growing pains to ensure that CSEd fulfills its educational promise in sustainable and equitable ways.

CS learning is challenging but it also offers teachers and learners the opportunity for transformation. It requires students to:

- understand what computers are and how they run programs (e.g., Ben-Ari, 1998; du Boulay, 1986; Guzdial, 2015);
- interpret, trace, and debug code (Lewis, 2012; Lister, 2016);
- steer away from several categories of misconceptions (Pea, 1986; Sorva, 2012);
- manage cognitive load (Lister, 2011; Margulieux et al., 2012);
- understand counterintuitive, obtuse notations and conventions in some programming languages (Stefik & Siebert, 2013);
- know content from other disciplines (e.g., reading, arithmetic, algebra, variables) and understand their overlaps and contradictions with CS (e.g., Franklin et al., 2017; Grover & Basu, 2017); and
- work in long-term projects and environments that are different from normal classrooms (Morrison et al., 2016).

Despite these challenges, CSEd offers many advantages and the potential to transform learning environments and school work. CS includes algorithms, design, data, making, creativity, and personal expression. An emerging approach to CSEd also facilitates productive collaboration in the classroom, connects to personally meaningful aspects of the lives of students, allows for new types of knowledge and assessments to be valued in schools, boosts the potential of project-based learning approaches, and opens possibilities of innovative ways to organize learning environments (e.g., Berland et al., 2013; Blikstein et al., 2014; Brennan, 2013; Buechley & Eisenberg, 2008; diSessa, 2000; Sherin, 2001; Turkle & Papert, 1990). Addressing and harnessing these advantages is important, particularly for K–8 learners, as our world becomes more technological and digital, and equitable participation requires CS fluency. This makes CSEd necessary in K–8 not just as an elective subject, but as a mandatory topic. There is no question anymore about the importance of CSEd, its place and need in public education, but there are differing opinions on why and how it should be done. Among the most prominent rationales for increasing access to CSEd is that it can serve as a foundational literacy upon which other knowledge/activities can be built, and as a powerful context for profound, authentic, and interdisciplinary learning in other subjects. CSEd can serve as an expressive, creative medium to allow young learners to express ideas in ways that are socially and culturally relevant, and also a valuable tool for civic and political participation.

Research has unearthed misconceptions to be addressed (Sorva, 2012), as well as effective pedagogies, classroom strategies, and language design principles to improve CSEd in K–8. For example, there's need for the design of robust and developmentally appropriate programming tools for multiple age groups and domains (e.g., Lister, 2016). The instrumentation of those tools (in combination with other data sources) could also provide additional insights into student learning. There is, still, a growing awareness of the need for CSEd research to become more rigorous[10] and to better connect with established knowledge bases in education and the learning sciences, as well as with emerging methodologies such as machine learning. These new mixed-methods research approaches and data sources can help CSEd implementation by creating tools and dashboards to help teachers with classroom activities such as managing and assessing complex project-based work and creating infrastructures for data-sharing among researchers.

Given the importance of CSEd, many of the interviewees believe that national rollouts of robust CSEd programs will require massive investment in the creation of state-level standards and curriculum, teacher preparation and certification, software/hardware infrastructure, and research. It is not clear if all stakeholders are aware of the depth of the effort, but many feel that partial rollouts have the potential to increase social disparities and educational inequalities, privileging more affluent or well-resourced schools and districts. Additionally, although large scale "CS exposure" programs are reaching millions of children, there is concern that they do not guarantee sustained engagement, in particular for underserved youth. Addressing these concerns requires better metrics, arms-length evaluation of programs, and more consensus on what constitutes success. In addition, exposure programs could benefit from follow-up activities, curricula, and sufficient resources to support deeper learning and stronger outcomes.

With an eye toward stronger outcomes, a reliance on high-quality standards, curricula, and assessments alone are not a guarantee of effective implementation. Education is always instantiated by teachers, so attention to pedagogy, teacher support, and the complex dynamics of adopting new curricula is crucial. Specifically, we found that teacher development is a key factor in the success of CSEd, both pre-service and in-service. And, the understanding of equity, inclusiveness, and unconscious biases about CS success are viewed as necessary to teacher development programs. If CSEd programs are not implemented with an eye towards equity, they risk deepening educational inequalities that already exist and defeating the purpose of CSEd as a force for youth empowerment and social justice.

Overall, when considering the progress made to date, the state of the art of the research, and the growing demand for large-scale rollouts, instead of the adoption of one single implementation model, researchers advocate for a repertoire of well-studied and well-rationalized models that are sufficiently flexible to be adapted to multiple local contexts. To deal with these demands, the number of researchers and research programs in CSEd will need to grow dramatically. In doing so, there's an expressed need to secure significant funding pathways to ensure the necessary research infrastructure is made available.

10 "Rigorous," in this context, refers to high-quality standards within all research paradigms: qualitative, quantitative, data mining, etcetera, and not only steering research towards elements that can be quantitatively measured.

11. Recommendations

Advancing CSEd in equitable ways requires a comprehensive approach that ensures all students are well prepared for the future. Building on the recent advances made in CSEd and the growing demand for more, the CSEd community should consider pursuing strategies that can benefit all students, especially those who are underserved. We highlight recommendations below that address the findings of this report.

11.1 Create clarity around the different visions of CSEd

- **Create clarity and alignment around the core rationales that varied stakeholders use to advance CSEd** (labor market, computational thinking, computational literacy, equity of participation), so that the solutions implemented build upon the similarities, compatibility, complementarity, and differences between them.
- **As CSEd grows, it should maintain some of its key transformational and innovative elements.** Such elements include the focus on project-based learning approaches, alignment with learner interests, culture, and ways of expression, exploration of new content areas, collaborative work, and openness to multiple ways of doing CS (epistemological pluralism).

11.2 Make participation equitable

- **National rollouts of CSEd must prioritize and evaluate their impact on improving the equitable participation of all students regardless of backgrounds, motivations, preparations, and abilities.** The demand for computing skills is growing rapidly not only for economic reasons but in all aspects of children's lives. Preparing all students for the future requires institutions and mechanisms that shape and support CSEd to develop plans and to assess how effective they are in providing learning opportunities for all students.
- **CSEd should be mandatory content in public schools** in order to overcome biases and structural inequalities that prevent equitable participation. As long as CSEd continues to be viewed as an elective or specialty subject, concerns will persist about the unequal presence of CS in public schools, the quality of instruction, and educators' and counselors' unconscious bias regarding who is "suited" to take CS classes.

11.3 Ensure teachers are prepared and supported

- **Develop integrated systems of teacher certification, training programs, and professional incentives, with special attention to the pre-service pipeline.** The interactions between teachers and students in classrooms are a determining factor in whether students learn CS successfully. Teachers are the linchpin in any effort to implement and change CSEd and so the preparation, effective development, and retention of CSEd teachers need to be prioritized.
- **Provide high-quality teacher preparation and induction models focused on inclusive CS pedagogical content knowledge.** In addition to exposing teachers to CS content, teacher preparation programs must also provide teachers with time to learn and practice inclusive CS pedagogies. These pedagogies need to be interwoven into the entire PD program.

11.4 Create continuity and coherence around learning progressions

- **Describe recommended sequences for CS knowledge and skills that can build on one another as students learn new topics over time.** With clear connections between what comes before and after a particular point in the learning progression, teachers can scaffold any missing knowledge or skills and determine the next steps to move the student forward.
- **Develop robust and developmentally-appropriate programming tools for multiple age groups, especially for K–8, and domains that also provide additional insights into student learning.** We should develop new programming tools and dashboards that can also help teachers with classroom activities such as managing and assessing complex project-based work, as well as infrastructures for research data sharing.

11.5 Commit to ongoing and thorough research

- **CSEd research funders, researchers, practitioners and policymakers should develop a strategic plan for CSEd research.** The plan should provide a long-term achievable, innovative, and actionable research agenda to address critical challenges identified in this report. To sustain this strategy, there must be a shared commitment among stakeholders to make CSEd research an integrated, stable, academically valued and well-funded enterprise for years to come.

12. Conclusion

In sum, the time is ripe for thoughtfully targeted and comprehensive action to advance the CSEd community. A large and diverse body of perspectives indicates that we must address the social, economic, and cultural barriers surrounding computing. If access and inclusiveness are addressed effectively, we can meet current and future workforce and citizenship demands. And we can do so in ways that equitably drive technological and social progress and give youth new avenues for personal expression and empowerment. This effort requires the cooperation and coordination of interdisciplinary, inter-sector teams that thoughtfully design, implement, evaluate, and learn from CSEd initiatives. Only in this way can we achieve the hoped-for scale and sustainability, and realize the ultimate vision of generations of researchers, practitioners, and policy makers that have been trying, for the last 50 years, to bring CS to all students.

Appendix A: Methods

We utilized three main data sources for this report: interviews, literature reviews, and analysis of papers recommended by the interviewees. For the interviews, we selected leaders in the field from various universities, institutions, and organizations, trying to balance intellectual traditions, academic backgrounds, and expertise. The final group of interviewees consisted of 14 practitioners, researchers, and scholars shown below.

Matthew Berland	University of Wisconsin-Madison
Leah Buechley	Rural Digital
Michael Clancy	University of California, Berkeley
Andrea "Andy" diSessa	University of California, Berkeley
Sally Fincher	University of Kent
Shuchi Grover	Formerly SRI International
Mark Guzdial	Georgia Institute of Technology
Mike Horn	Northwestern University
Jane Margolis	University of California, Los Angeles
Mitchel Resnick	Massachusetts Institute of Technology
Sue Sentance	King's College, London
Ben Shapiro	University of Colorado, Boulder
David Weintrop	University of Maryland
Pat Yongpradit	Code.org

All invited interviewees accepted to be interviewed, except one professor who nominated another scholar in his own department (Michael Clancy, University of California, Berkeley), and Andrea diSessa, who preferred to send an in-preparation paper instead (the paper is used in this report in lieu of an interview, and listed in the Works Cited).

We used a semi-structured protocol for the interviews which included questions about the relevance and importance of teaching CS, the main research findings in the field, and research, policy, and implementation agendas for the next year **(see Appendix B for interview protocols).** The interviews were conducted remotely via videoconference, audio recorded, transcribed in their entirety, and analyzed by the author of this report. All participants were given the option of anonymity and none opted for it.

Principal themes were extracted from the initial coding: (a) teacher preparation; (b) policy and scale up; (c) curriculum development; (d) cultural, diversity, and equity issues; (e) pedagogy; and (f) history of CSEd. These categories informed a further refining of the coding, so the data was recoded for more fine-grained topics, resulting in approximately 1,000 excerpts grouped into 130 codes. Those codes were then re-categorized in terms of the six initial themes and informed the structure of the document.

The literature was selected using a combination of recommendations from the interviewees, well-established policy documents such as the CSTA K–12 Computer Science Standards (Seehorn et al., 2011) and the K–12 Computer Science Framework (K–12 Computer Science Framework Steering Committee, 2016), foundational works in the field, and existing literature reviews. We used the literature to add a layer of peer-reviewed research to the topics extracted from the interviews, and triangulated research findings across interviews and the literature.

We chose this hybrid format (interviews and reviews) to simultaneously capture well-established facts and findings but also novel information that has not yet made it to the publication venues in the field. Also, some of the important challenges and issues in CSEd often do not show up in peer-reviewed publications because many active members of the community are tool developers instead of researchers—so their work would not be necessarily captured in a traditional literature review. This combined use of interviews and literature gave us a more comprehensive view of the state of the very young and dynamic field of CSEd.

Appendix B: Interview Protocols

Long interview protocol

First part: History

1. CSEd has seen a resurgence in the last 5 or 10 years after a decade of relative silence. Is that an accurate portrait, and what is your version of the history of CSEd? What was your participation in this history?
2. There are several reasons for teaching CS in K–8. Some people say that it is a marketable job skill, some say it is a general thinking skill (computational thinking), and some others say that it is a broader literacy (computational literacy) just like reading and writing, which you could use to learn all subjects. What should be the reasons for us to teach CS in K–8?

Second part: State of the art

3. What is the state of K–8 CSEd? What have we achieved in this area in terms of scale, depth, mindshare, and research? Do you know of a particularly powerful experience in CS education at the K–8 level?
4. What is the typical experience of a K–8 student today regarding CS? What type of contact do they have with programming? How is that experience on the high-end and low-end of the educational spectrum? In other words, how does it look like for a student in a high-achieving institution, versus an average or low-achieving public school?
5. In terms of research in CSEd, in your opinion, what are the most well-established facts and findings about how K–8 students learn to program? Are there some undisputable findings about that, or at least the closest candidates? What are some counterintuitive findings in your own work and in work elsewhere about this?
6. Getting more specific: Which concepts are most problematic for students and at what age do these difficulties begin? Are you aware of specific pedagogical approaches or supports that successfully mitigate these difficulties?
7. In schools, what are the most effective ways to teach CS concepts to a broadly diverse student audience?
8. Compared to mathematics education and/or science education, where are we with CSEd in terms of understanding the nature of CS learning?
9. What are the seminal studies from the past 10 years that examine how and when students best learn computer science? What is on your list of must-read papers (includes reviews) for someone who wants to get informed about this topic?

Third part: Future

10. What do you see as the top three most important CSEd research topics in five years? In 15 years?
11. What are the three most important challenges for the research community in CSEd in the next five years? What are the most important studies we need to do, the tools we need to develop, and/or the most important learning questions we need to answer?
12. What are the most important policy challenges for the next five years in CSEd?
13. How would you like to see CSEd in 15 years?

Short interview protocol

1. What should be the main reasons to teach CS in K–8, and why?
2. In your opinion, what are the most well-established facts and findings about how K–8 students learn to program, or at least closest candidates? This can be in terms of the social engineering of classrooms or group/pair work while learning to code, design principles for programming environments, concepts that are easier or harder, etcetera.
3. Getting more specific (if you have this information): From your own work or from elsewhere, which concepts are most problematic for students and at what age do these difficulties begin? Are you aware of specific pedagogical approaches, tools, or supports that successfully mitigate these difficulties?
4. What are the three most important challenges for the research community in CSEd for the next five years? These challenges could be in terms of important studies we need to do, the tools we need to develop, policy initiatives, and/or the most important learning questions we need to answer?
5. What is on your list of must-read papers (includes reviews) for someone who wants to get informed about this topic?

Appendix C: CS Education Timeline

1960s	1961	Alan Perlis delivers lecture at the "Computers and the World of the Future" Symposium at Massachusetts Institute of Technology (MIT), stating that "everyone should learn to program as part of a liberal education."
	1964	John Kemeny and Thomas Kurtz (Dartmouth College) create the BASIC programming language.
	1967	LOGO computer language created by Seymour Papert, Cynthia Solomon, and Wally Feurzeig.
1970s	1977	Adele Goldberg and Alan Kay publish "Personal Dynamic Media," inspiring the development concept of the Dynabook and the development of SmallTalk.
1980s	1980	"Mindstorms" published by Seymour Papert. Robert Taylor publishes "The Computer in the School: Tutor, Tool, Tutee."
	1981	Turtle Geometry published by Hal Abelson and Andrea diSessa. Bank Street LOGO project starts led by Roy Pea and Midian Kurland. Soon after, Marcia Linn and Michael Clancy (University of California, Berkeley) join the first National Institute of Education study on programming. Richard Pattis creates the Karel the Robot programming language.
	1984	Introduction of the AP Computer Science A Exam with almost 7,000 students.
	1985	MIT-led LOGO experiment at the Hennigan Elementary School in Boston commences. MIT Media Lab created, Papert's efforts expand through his Epistemology & Learning group, "constructionism" term coined.
	1988	The LEGO Company launches LEGO/LOGO.
1990s	Early '90s	Reversal in federal funding, research in CSEd slows down in the U.S.
	1991	Launch of StarLogo, the first massively parallel programming language for non-experts.
	1993	Launch of new programming environments: NetLogo (Uri Wilensky), AgentSheets (Alex Reppening), LogoBlocks (MIT Media Lab), Alice (Randy Pausch), E-toys (Alan Kay).
	1995	MIT Media Lab releases the Cricket, the first full platform for robotics expressly designed for children.
	1996	"Computational thinking" is first introduced by Seymour Papert.
	1998	Launch of LEGO Mindstorms robotics kit.
2000s	2001	First Multi Institutional, Multi National of Assessment of Programming Skills of First-Year CS Students (MIMN) study in CSEd, led by Mike McCracken.
	2001	"Changing Minds: Computers, Learning, and Literacy" by Andrea diSessa introduces the idea of "computational literacy."'
	2002	"Unlocking the Clubhouse: Women in Computing" by Allan Fisher and Jane Margolis addresses gender and computing. MIT Media Lab releases the GoGo Board, the first open-source platform for robotics, designed expressly for developing nations.
	2003	ACM Task force for K–12 Computer Science Education formed and publishes "A Model Curriculum for K–12 Computer Science Education."

	2004	Computer Science Teachers Association (CSTA) created, takes on creating the CSTA K–12 Computer Science Standards.
	2004	Second Multi Institutional, Multi National (MIMN) of Reading and Tracing Skills in Novice Programmers, led by Raymond Lister.
	2005	Bootstrap and the scaffolding research projects start. International Computing Education Conference (ICER) conference starts. National Science Foundation (NSF) forms the Broadening Participation in Computing research program. Release of the Arduino platform, which rapidly becomes the standard for physical computing.
	2006	The term “computational thinking” appears in an influential paper by Jeanette Wing, as she begins her tenure at the NSF. First Maker Faire in the San Francisco Bay Area.
	2007	Scratch programming environment is launched by the Lifelong Kindergarten group at MIT.
	2008	CS10K effort is launched and funded by NSF. Jane Margolis publishes “Stuck in the Shallow End: Education, Race, and Computing.” Margolis and her team, including Joanna Goode and Gail Chapman, launch Exploring Computer Science (ECS) in Los Angeles, addressing issues of race and underrepresentation in CS.
	2009	Launch of CSEd Week by ACM and CSTA.
2010s	2010	Computing in the Core initiated by ACM/CSTA/Microsoft/Google/NCWIT, one of the first major advocacy efforts for CSEd.
	2013	Code.org launches Hour of Code. CSEd organizations explode: Codecademy, CodeHS, Tynker, Treehouse, Khan Academy, Iridescent Learning, GirlsWhoCode, Black Girls Code.
	2013	CS4All program launches in Chicago public schools.
	2014	President Barack Obama becomes the first U.S. president to write a line of code and announces federal support for CS4All.
	2015	Hour of Code reaches 100 million “hours served”; Arkansas becomes first state to require all public and charter high schools to offer CS; CS in San Francisco Unified School District; CS in New York City schools.
	2016	Countrywide initiatives for teaching CS to all children in the U.K. (Computing in Schools Project), in the U.S., as well as Denmark, Finland, and other countries.
	2017	White House announcement to expand access to STEM and CSEd. NSF and College Board partner to design a new and innovative Advanced Placement CS course, “Computer Science Principles.”

Works Cited

Adelson, B., & Soloway, E. (1985). The role of domain experience in software design. *IEEE Transactions on Software Engineering* (11), 1351–1360.

Anderson, J. R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 8*(2), 87–129.

Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE Bulletin, 30*(1), 257–261.

Berland, M. (2017, April). Phone interview with Paulo Blikstein.

Berland, M., Martin, T., & Benton, T. (2013). Using Learning Analytics to Understand the Learning Pathways of Novice Programmers. *Journal of the Learning Sciences, 22*(4), 564-599.

Bevan, J., Werner, L., & McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class. In *Proceedings of the 15th Conference on Software Engineering Education and Training - CSEE&T 2002* (pp. 100–107). Covington, KY.

Bhuiyan, S., Greer, J., & McCalla, G. (1990). Mental models of recursion and their use in the SCENT programming advisor. In *Proceedings of the International Conference on Knowledge Based Computer Systems - KBCS 89* (pp. 133–144). Bombay, India.

Blikstein, P. (2011). Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge - LAK 2011* (pp. 110–116). Banff, Canada.

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in novices' learning of computer programming. *Journal of the Learning Sciences, 23*(4), 561–599.

Bonar, J., & Soloway, E. (1983). Uncovering principles of novice programming. In *Proceedings of the 10th ACM Symposium on Principles of Programming Languages - SIGACT-SIGPLAN* (pp. 10–13). Austin, Texas.

Booth, S. (1992). Learning to program: *A phenomenographic perspective*. Gothenburg: Acta Universitatis Gothoburgensis.

Bowman, B. T. (1985). Computers and young children. In *Proceedings of the National Association for the Education of Young Children*. New Orleans, LA.

Brennan, K. (2013). Learning computing through creating and connecting. *Computer, 46*(9), 52-59.

Buechley, L. (2017, April). Phone interview with Paulo Blikstein.

Buechley, L., & Eisenberg, M. (2008). The LilyPad Arduino: Toward wearable engineering for everyone. *IEEE Pervasive Computing, 7*(2), 12–15.

Buechley, L., Eisenberg, M., Catchen, J., & Crockett, A. (2008). The LilyPad Arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems - CHI 2008* (pp. 423–432). Florence, Italy.

Butler, D., & Close, S. (1989). Assessing the benefits of a LOGO problem solving course. *Irish Educational Studies, 8*(2), 168–190.

Carmichael, H. W. (1985). *Computers, Children and Classrooms: A Multisite Evaluation of the Creative Use of Microcomputers by Elementary School Children.* Toronto, Canada: Ontario Ministry of Education.

Catrambone, R. (1998). The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General, 127*(4), 355-376.

Chi, M., Feltovich, P., Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science, 5*(2), 121–152.

Clancy, M. (2017, June). Phone interview with Paulo Blikstein.

Clements, D. H. (1990). Metacomponential development in a LOGO programming environment. *Journal of Educational Psychology, 82*(1), 141.

Clements, D. H., & Meredith, J. S. (1993). Research on LOGO: Effects and efficacy. *Journal of Computing in Childhood Education, 4*(4), 263-290.

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE 2012* (pp. 141—146). Raleigh, NC.

De Corte, E., & Verschaffel, L. (1989). Logo: A vehicle for thinking. In B. Greer & G. Mulhern (Eds.), *New directions in mathematics education* (pp. 63–81). London/New York: Routledge.

diSessa, A. (1985). A principled design for an integrated computational environment. *Human-Computer Interaction, 1*(1), 1–47.

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

diSessa, A. (2018, in press). Computational literacy and “the big picture” concerning computers in mathematics education. *Mathematical thinking and learning, 20.*

diSessa, A., & Cobb, P. (2004). Ontological Innovation and the role of theory in design experiments. *Journal of the Learning Sciences, 13*(1), 77–103.

du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research, 2*(1), 57–73.

Ericson, B. J., Rogers, K., Parker, M., Morrison, B., & Guzdial, M. (2016). Identifying design principles for CS teacher ebooks through design-based research. In *Proceedings of the 12th Annual International Conference on International Computing Education Research - ICER 2016* (pp. 191–200). Melbourne, Australia.

Fields, D. & Giang, M. & Kafai, Y. (2013). Understanding collaborative practices in the Scratch online community: Patterns of participation among youth designers. Proceedings of the Computer-Supported Collaborative Learning Conference (CSCL), (pp. 200–207).

Fincher, S. (2015). What are we doing when we teach computing in schools? *Communications of the ACM, 58*(5), 24-26.

Fincher, S. (2017, June). Phone interview with Paulo Blikstein.

Fisler, K. (2014). The recurring rainfall problem. In *Proceedings of the 10th Annual International Conference on International Computing Education Research - ICER 2014* (pp. 35–42). Glasgow, Scotland.

Fleury, A. (1991). Parameter passing: The rules the students construct. ACM *SIGCSE Bulletin, 23*(1), 283–286.

Franklin, D., Skifstad, G., Rolock, R., Mehrotra, I., Ding, V., Hansen, A., …Harlow, D. (2017). Using upper-elementary student performance to understand conceptual sequencing in a blocks-based curriculum. In *Proceedings of the 47th ACM Technical Symposium on Computer Science Education - SIGCSE 2017* (pp. 231–236). Seattle, WA.

Glaser, B. G. (1992). *Basics of grounded theory analysis: Emergence vs. forcing.* Mill Valley, CA: Sociology Press.

Google LLC., & Gallup Inc. (2016). Diversity gaps in computer science: Exploring the underrepresentation of girls, Blacks and Hispanics. Retrieved October 1, 2017 from http://goo.gl/PG34aH.

Graham, P. (2004). *Hackers & painters: Big ideas from the computer age.* Sebastopol, California: O’Reilly Media.

Grover, S. (2017, April). Phone interview with Paulo Blikstein.

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 47th ACM Technical Symposium on Computer Science Education - SIGCSE 2017* (pp. 267–272). Seattle, WA.

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

Grover, S., Pea, R., & Cooper, S. (2014). Expansive framing and preparation for future learning in middle-school computer science. In *Proceedings of the International Conference of the Learning Sciences - ICLS 2014* (pp. 992–996). Boulder, CO.

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

Grover, S., Pea, R., & Cooper, S. (2016). Factors influencing computer science learning in middle school. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE 2016* (pp. 552–557). Kansas City, MO.

Guzdial, M. (1993). E*mile: Software-realized scaffolding for science learners programming in mixed media.* (Doctoral dissertation), University of Michigan.

Guzdial, M. (2013). Exploring hypotheses about media computation. In *Proceedings of the 9th Annual International Conference on International Computing Education Research - ICER 2013* (pp. 19–26). San Diego, CA.

Guzdial, M. (2014). The most gender-balanced computing program in the USA: Computational Media at Georgia Tech. Retrieved June 1, 2017 from https://computinged.wordpress.com/2014/09/02/the-most-gender-balanced-computing-program-in-the-usa/

Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics, 8*(6), 1–165.

Guzdial, M. (2017, April). Phone interview with Paulo Blikstein.

Hermans, F., & Aivaloglou, E. (2017). *To Scratch or not to Scratch?: A controlled experiment comparing plugged first and unplugged first programming lessons*. Retrieved October 1st 2017 from http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2017-015.pdf

Hestenes, D., Wells, M., & Swackhamer, G. (1992). Force concept inventory. *The Physics Teacher, 30*(3), 141–158.

Hillel, J., & Kieran, C. (1987). Schemas used by 12-year-olds in solving selected turtle geometry tasks. *Recherches en Didactique des Mathématiques, 8*(1.2), 61–102.

Horn, M. (2017, April). Phone interview with Paulo Blikstein.

Hoyles, C., & Noss, R. (1989). The computer as a catalyst in children's proportion strategies. *Journal of Mathematical Behavior, 8,* 53–75.

Hoyles, C., & Noss, R. (1992). A pedagogy for mathematical microworlds. *Educational Studies in Mathematics, 23*(1), 31–57.

Hoyles, C., Sutherlands, R., & Noss, R. (1991). Evaluating computer-based microworld: What do pupils learn and why? In *Proceedings of the 15th Conference of the International Group for the Psychology of Mathematics Education* (pp. 197–204). Assisi, Italy.

K–12 Computer Science Framework Steering Committee. (2016). *K–12 Computer Science Framework* (978-1-4503-5278-9). Retrieved June 1, 2017 from New York, NY: http://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf

Kahney, H. (1983). What do novice programmers know about recursion. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI 1983* (pp. 235–239). Boston, MA.

Lehrer, R., & Smith, P. (1986). LOGO learning: Are two heads better than one. Paper presented at the *Annual meeting of the American Educational Research Association - AERA 1986*. San Francisco, CA.

Lewis, C. M. (2012). The importance of students' attention to program state: A case study of debugging behavior. In *Proceedings of the 8th Annual International Conference on International Computing Education Research - ICER 2012* (pp. 127–134). Auckland, New Zealand.

Lister, R. (2011). Computing Education Research: Programming, syntax and cognitive load. *ACM Inroads, 2*(2), 21–22.

Lister, R. (2016). Toward a Developmental Epistemology of Computer Programming. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education - WiPSCE 2016* (pp. 5–16). Münster, Germany.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., …Seppälä, O. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin, 36*(4), 119–150.

Liu, M. (1997). The effects of HyperCard programming on teacher education students' problem-solving ability and computer anxiety. *Journal of Research on Computing in Education, 29*(3), 248–262.

Maltese, A., & Tai, R. (2011). Pipeline persistence: Examining the association of educational experiences with earned degrees in STEM among US students. *Science Education, 95*(5), 877–907.

Margolis, J. (2017, April). Phone interview with Paulo Blikstein.

Margulieux, L. E., Guzdial, M., & Catrambone, R. (2012). Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the 8th Annual International Conference on International Computing Education Research - ICER 2012* (pp. 71–78). Auckland, New Zealand.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, …Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin, 33*(4), 125–180.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin, 34*(1), 38–42.

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

Morrison, B. B., Margulieux, L. E., Ericson, B., & Guzdial, M. (2016). Subgoals help students solve Parsons problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE 2016* (pp. 42–47). Memphis, TN.

Nastasi, B. K., Clements, D. H., & Battista, M. T. (1990). Social-cognitive interactions, motivation, and cognitive growth in LOGO programming and CAI problem-solving environments. *Journal of Educational Psychology, 82*(1), 150–158.

National Research Council. (2006). *America's lab report: Investigations in high school science.* Washington, DC: National Academies Press.

National Research Council. (2012). *A Framework for K–12 Science Education: Practices, Crosscutting Concepts, and Core Ideas.* Washington, DC: The National Academies Press.

NGSS Lead States. (2013). *Next Generation Science Standards: For States, By States.* Washington, DC: The National Academies Press.

Noonan, R. (2017). *STEM Jobs: 2017 Update (ESA Issue Brief # 02-17).* Retrieved October 1st 2017 from http://www.esa.gov/reports/stem-jobs-2017-update

O'Neill, C. (2016). *Weapons of Math Destruction.* New York, NY: Crown Publishing Group.

Palumbo, D. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research, 60*(1), 65–89.

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

Papert, S. (1995). Why school reform is impossible. *Journal of the Learning Sciences, 6*(4), 417–427.

Pea, R. (1986). Language-Independent Conceptual 'Bugs' in Novice Programming. *Journal of Educational Computing Research, 2*(1), 25–36.

Pea, R., Kurland, D. M., & Hawkins, J. (1987). Logo and the development of thinking skills. In R. Pea, Sheingold, K. (Ed.), *Mirrors of mind* (pp. 193–317). Norwood, NJ: Ablex.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Paterson, J. (2007). A survey of literature on the teaching of introductory programming. A*CM SIGCSE Bulletin, 39*(4), 204–223.

Perkins, D. N., Schwartz, S., & Simmons, R. (1988). Instructional strategies for the problems of novice programmers. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 153–178). Hillsdale, NJ: Lawrence Erlbaum.

Resnick, M. (2017, April). Phone interview with Paulo Blikstein.

Rist, R. S. (2004). Learning to program: Schema creation, application, and evaluation. In S. Fincher & M. Petre (Eds.), *Computer Science Education Research* (pp. 175–195). London, UK: Taylor & Francis.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172.

Ruvalcaba, O., Werner, L., & Denner, J. (2016). Observations of pair programming: Variations in collaboration across demographic groups. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE 2016* (pp. 90–95). Memphis, TN.

Sajaniemi, J., & Kuittinen, M. (2008). From procedures to objects: A research agenda for the psychology of object-oriented programming education. *Human Technology, 4*(1), 75–91. Retrieved June 1st 2017 from http://www.humantechnology.jyu.fi

Samurçay, R. (1989). The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. In E. Soloway & J. Spohrer (Eds.), *Studying the novice programmer* (pp. 161–178). Hillsdale, NJ: Lawrence Erlbaum.

Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., …Verno, A. (2011, April 1st 2017). CSTA K–12 Computer Science Standards: Revised 2017. Retrieved from https://www.csteachers.org/page/standards

Sentance, S. (2017). (2017, June). Phone interview with Paulo Blikstein.

Shapiro, B. (2017, April). Phone interview with Paulo Blikstein.

Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning, 6*(1), 1–61.

Sleeman, D., Putnam, R. T., Baxter, J., & Kuspa, L. (1986). Pascal and high school students: A study of errors. *Journal of Educational Computing Research, 2*(1), 5–23.

Smith, P. A., & Webb, G. I. (1995). Reinforcing a generic computer model for novice programmers. In *Proceedings of the 7th Australian Society for Computer in Learning in Tertiary Education - ASCILITE 1995.* Melbourne, Australia.

Soloway, E., Adelson, B., & Ehrlich, K. (1988). *Knowledge and processes in the comprehension of computer programs.* In M. T. H. Chi, R. Glaser, & M. Farr (Eds.), The nature of expertise (pp. 129–152) Hillsdale, NJ: Lawrence Erlbaum. Associates, Inc.

Sorva, J. (2012). *Visual program simulation in introductory programming education*. (Doctoral dissertation), Aalto University, Finland. Retrieved June 1, 2017 from http://lib.tkk.fi/Diss/2012/isbn9789526046266/isbn9789526046266.pdf.

Spohrer, J., & Soloway, E. (1986). Novice mistakes. Are the folk wisdoms correct? *Communications of the ACM,* 29(7), 624–632.

Stager, G. (2017). A Modest Proposal. Retrieved October 30, 2017 from http://stager.tv/blog/?p=4153.

Stefik, A., & Siebert, S. (2013). An Empirical Investigation into Programming Language Syntax. *Transactions on Computing Education, 13*(4), 1–40.

Suthers, D. D. (2006). Technology affordances for intersubjective meaning making: A research agenda for CSCL. *International Journal of Computer-Supported Collaborative Learning, 1*(3), 315-337.

Taylor, C., Zingaro, D., Porter, L., Webb, K. C., Lee, C. B., & Clancy, M. (2014). Computer science concept inventories: Past and future. *Computer Science Education, 24*(4), 253-276.

Tew, A., & Dorn, B. (2013). The case for validated tools in computer science education research. *Computer, 46*(9), 60–66.

Turkle, S., & Papert, S. (1990). Epistemological pluralism: Styles and voices within the computer culture. *Signs,* 128–157.

U.S. Department of Labor. (2007). *The STEM workforce challenge: The role of the public workforce system in a national solution for a competitive science, technology, engineering, and mathematics (STEM) workforce.*

Visser, W. (1987). Strategies in programming programmable controllers: A field study on a professional programmer. In G. Olson, S. Sheppard, & E. Soloway (Eds.), *Empirical Studies of Programmers: Second Workshop* (pp. 217–230). Norwood, NJ: Ablex.

Vogel, S., Santo, R., & Ching, D. (2017). Visions of computer science education: Unpacking arguments for and projected impacts of CS4All initiatives. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education - SIGCSE 2017* (pp. 609–614). Seattle, WA.

Weintrop, D. (2017, May). Phone interview with Paulo Blikstein.

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

Weintrop, D., & Wilensky, U. (2015a). To block or not to block, that is the question: Students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children - IDC 2015* (pp. 199–208). Medford, MA.

Weintrop, D., & Wilensky, U. (2015b). Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In *Proceedings of the 11th Annual International Conference on International Computing Education Research - ICER 2015* (pp. 101–110). Omaha, NE.

Weintrop, D., & Wilensky, U. (2017). Between a Block and a Typeface: Designing and Evaluating Hybrid Programming Environments. In *Proceedings of the 16th International Conference on Interaction Design and Children - IDC 2017* (pp. 183–192). Stanford, CA.

Wilensky, U. (1999, updated 2006, 2017). NetLogo [Computer software] (Version 6). Evanston, IL: Center for Connected Learning and Computer-Based Modeling. Retrieved from http://ccl.northwestern.edu/netlogo

Wilensky, U., & Papert, S. (2010). Restructurations: Reformulating knowledge disciplines through new representational forms. In *Proceedings of Constructionism 2010.* Paris, France.

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep or a firefly: Learning biology through constructing and testing computational theories. *Cognition & Instruction, 24*(2), 171–209.

Wilkerson-Jerde, M., Wagh, A., & Wilensky, U. (2015). Balancing curricular and pedagogical needs in computational construction kits: Lessons from the DeltaTick Project. *Science Education, 99*(3), 465–499.

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

Yongpradit, P. (2017, April). Phone interview with Paulo Blikstein.

About

About Google

Google's core mission is to organize the world's information and make it universally accessible and useful. Google creates products to increase access to opportunity, break down barriers and empower people through technology. To help reach these goals, Google works to inspire young people around the world not just to use technology but to create it. There is a need for more students to pursue an education in computer science, particularly girls and minorities, who have historically been underrepresented in the field. For more information about Google's computer science education efforts, visit g.co/csedu.

About TLTL

Paulo Blikstein is an Assistant Professor of Education and (by courtesy) Computer Science at Stanford University, where he directs the The Transformative Learning Technologies Lab (TLTL). The TLTL, part of the Stanford University Graduate School of Education, is an interdisciplinary research group with graduate students and scholars from education, computer science, engineering, and psychology, focusing on how new technologies can deeply transform the learning of science, engineering, and mathematics. For more information about the TLTL, visit tltlab.org.

Paulo's contribution to this publication was as a paid consultant, and was not part of his Stanford University duties or responsibilities.