Sowing the Seeds: A Landscape Study on Assessment in Secondary Computer Science Education

by
Aman Yadav,
David Burkhart,
Daniel Moix,
Eric Snow,
Padmaja Bandaru,
and Lissa Clayborn

Computer Science Teachers Association

Association for Computing Machinery
Advancing Computing as a Science & Profession

Sowing the Seeds of Assessment Literacy in Secondary Computer Science Education: A Landscape Study

by

Aman Yadav[1], David Burkhart[2], Daniel Moix[3], Eric Snow[4], Padmaja Bandaru[5], and Lissa Clayborn[6]

[1]Michigan State University [2]Sheridan High School
[3]Bryant High School [4]SRI International
[5]AMSA Charter School [6]CSTA

Computer Science Teachers Association (CSTA)
Publication Information

Computer Science Teachers Association
2 Penn Plaza, Suite 701
New York, NY, 10121-0701

CSTA ISBN: 978-1-4503-2828-9
ACM Order Number 999151

Acknowledgements

CSTA would like to thank all of the high school teachers who took part in the study. Without your assistance, this paper could not have happened. We would also like to thank Cameron Fadjo, Sarah Gretter, and Chris Stephenson for all of their feedback during the initial drafts of this paper.

A heartfelt thank you to Google for recognizing the importance of this topic and for funding the research.

Contents

Executive Summary

This study summarizes what is known about assessment of student learning in high school Computer Science (CS) in the United States (US), reports on the results of the landscape study, and concludes with recommendations for advancing the state of assessment in K–12 CS. With support from Google, the Computer Science Teachers Association (CSTA) Assessment Task Force conducted a study of secondary school educators to determine the state of computer science education assessment and how teachers assess student learning in their computer science classrooms. Based on interviews with computer science practitioners, we found that teachers use a variety of formative and summative assessment techniques, but also face a number of challenges finding valid and reliable assessments to use in their classrooms.

Quality assessment items are few and far between as teachers rely on a assortment of sources (test banks, colleagues, even their own undergraduate CS courses) to evaluate student learning in their classes. Furthermore, teachers in this study discussed how the unique nature of computer science, including how students approach algorithms to write their programs, makes assessment a challenging and time-consuming endeavor. The ubiquity of programs and code on the Internet also makes it difficult for teachers to accurately gauge what students know.

Given the challenges of assessment in computer science classrooms, we recommend that the computer science education community develops valid and reliable computer science assessments to evaluate student learning. Additionally, we recommend creating an online repository to allow computer science teachers access to high quality assessments. Given the availability of CSTA K–12 computer science standards since 2011, the time has arrived for the computer science education community to develop complementary assessments that match those standards. In summary, the CSTA Assessment Task Force recommends the following tasks for the computer science education community:

- Develop valid and reliable assessments aligned to the CSTA K–12 Computer Science Standards.
- Develop valid and reliable formative and summative assessments for programming languages beyond Java, such as Python, C#, etc.
- Develop an online repository of assessment items for K–12 computer science teachers.
- Develop a community of practice surrounding the use of assessment in computer science classrooms.
- Design and deliver professional development to increase K–12 computer science teachers' assessment literacy. In particular, train teachers to understand and implement classroom assessment.

Introduction

Assessment is often a key policy lever for improvement in most education reform movements. Teachers use assessment scores to help them evaluate their students' understanding of particular content and determine the extent to which their learning goals are being met. Researchers and policy makers look to assessment scores, in both appropriate and inappropriate ways, to help them determine the efficacy of new curriculum reform efforts, teacher practice, and even school quality.

Computer science education is in the middle of its own reform movement. Computer science-related careers are projected to grow significantly over the next several decades and, despite the increasing enrollments in many CS fields of study, the expanding enrollments are not projected to meet the growing need. At the same time, there is increasing recognition that the pipeline to CS careers, as well as future innovation in CS, can be strengthened by introducing students to computational thinking (CT) skills and concepts earlier in secondary education contexts, before the traditional Advanced Placement® Computer Science (AP® CS) offering near the end of high school.

This is beginning to happen with the current national and state focus on making computer science (CS) count as a high school math or science credit, or as core admissions credit for colleges and universities. Several of the largest school districts in the US are currently working with CS industry leaders such as Google and professional development providers such as *Code.org* to implement computer science programs in their core school day.

As these reform efforts begin to scale nationally, teachers, school and district leaders, researchers, policy makers and other stakeholders in CS education are looking for evidence of student learning, increased academic interest by students, and higher enrollment in CS at the university level, especially in underrepresented populations. Which students are participating in CS-focused reform learning? How much are they learning? Assessment scores are going to be seen as a primary source of this evidence for policymakers and administrators.

Currently, however, the nature of assessment literacy in secondary computer science education is generally unknown. Opinions differ on what is available to the community at a rate that is cost effective (or free), as well as what is easy to access, use, and implement. Research on assessment literacy indicates that teachers, particularly teachers new to a content area and/or teaching practice, often face many challenges when it comes to engaging in robust assessment practices in their instruction (DeLuca & Klinger, 2010; Popham, 2009). These challenges are relatively well-understood in other disciplines like math and science, as are pathways to helping teachers with these challenges. The same is not true, however, for computer science, and we cannot create pathways to address these challenges unless we know what they look like in the context of CS education.

PHOTO: ©WWW.ISTOCKPHOTO.COM/BERNARDASV

The assessment landscape in secondary CS is particularly complex because of the nature of the central construct—computational thinking (CT—being measured. While there are many different conceptions of the focal knowledge and skills underlying CT, including the utility of different programming languages for different purposes, what is common across most of these conceptions is that CT consists of a complex constellation of computationally-focused problem solving and design practices, and these inquiry-based practices are often challenging to validly measure in traditional assessments. Research on measuring inquiry-based practices in other disciplines (e.g., science) suggests that performance-based and open-ended tasks are often best for assessing these skills, but that these types of tasks are also more challenging to design and implement in the classroom environment.

Before we can sow the seeds for more robust assessment literacy practices in the current CS education reform landscape, we need to examine and better understand that landscape, especially in secondary CS education where many of the main reforms are focused.

In particular, we need to better understand how the key players, especially secondary CS teachers, utilize assessment in their work and some of the primary challenges they face when engaging in assessment practices in their classrooms. Moreover, teachers are often at the forefront of implementing performance-based measures of student knowledge and skills in their classrooms. We would do well to gain a better understanding of what secondary CS teachers are currently doing in this area and model those best practices toward the design and development of a standards-aligned set of performance-based tasks appropriate for measuring the hard-to-assess knowledge and skills underlying the CT construct (e.g., algorithmic thinking). In other words, we have to identify the seeds that, when sown, have the best chance of growing, bearing fruit, and nourishing assessment literacy in the CS education reform landscape.

The Computer Science Teachers Association (CSTA), with funding from Google, established an Assessment Task Force. The Task Force conducted a landscape study to determine challenges high school teachers face when assessing student understanding of computing concepts and explore assessment practices CS teachers currently use in their classroom can be leveraged to support the design and development of standards-aligned, performance-based assessments.

This paper begins with a review of the computer science education reform landscape and the computational thinking construct, including what is known about assessment in secondary CS education; reports on the results of the landscape study; and concludes with recommendations for advancing assessment literacy in secondary CS education.

COMPUTATIONAL THINKING CAN BE BROADLY VIEWED AS THE INTELLECTUAL AND REASONING SKILLS NEEDED TO MASTER AND APPLY ALGORITHMIC THINKING, PATTERN RECOGNITION, ABSTRACTION, DECOMPOSITION, AND OTHER COMPUTATIONAL TECHNIQUES TO PROBLEMS IN A WIDE RANGE OF FIELDS (E.G., SEE WING, 2008; NRC, 2010).

Identifying the Seeds: Computational Thinking, Secondary CS Ed Reform, Assessment Literacy

Computer science (CS)-related careers are projected to grow significantly over the next several decades (United States Bureau of Labor Statistics, 2012). Despite recent increases in CS enrollment among college and university students, projections indicate that the number of graduates will not be sufficient to address industry needs. At the same time, there is growing recognition that the pipeline to CS careers, as well as future innovation in CS, can be strengthened by introducing students to computational thinking (CT) skills and concepts earlier in secondary education contexts, before the traditional Advanced Placement® CS (AP CS) offerings near the end of high school.

Researchers and practitioners in computer science and education have worked since the publication of Wing's influential article (Wing, 2006) to define computational thinking (CT). Computational thinking can be broadly viewed as the intellectual and reasoning skills needed to master and apply algorithmic thinking, pattern recognition, abstraction, decomposition, and other computational techniques to problems in a wide range of fields (e.g., see Wing, 2008; NRC, 2010).

Researchers have suggested some important knowledge and skills underlying CT: abstraction, algorithmic thinking, modeling, scale, and working with patterns (Denning, 2009; NRC, 2010; Wing, 2008). Among these skills and bodies of knowledge, the ability to write a computer program—often viewed as the "test" of computer science skills—is not at the same conceptual level as these components of computational thinking. The National Research Council (NRC) report (NRC, 2010) emphasized that thinking computationally is not synonymous with computer science, computer or technology literacy; rather, it is more than programming, and it differs in nature from mathematical, scientific, and quantitative thinking.

This recent literature on defining CT is complemented by considering what computer scientists themselves have said about their practice, profession, and research accomplishments (NRC, 2004). The NRC (2004) report *Computer Science: Reflections on the Field, Reflections from the Field* lays out *essential* characteristics of computer science research: creating and manipulating abstractions, creating and studying algorithms, and manipulating symbolic representations. When we look at computer science education research, we find that these constructs have been widely studied (e.g., abstraction: Benneddsen & Casperson, 2008; Colburn & Shute, 2007; Hazzan, 2003; Kramer, 2007; Nicholson, Good, & Howland, 2009; Sakhnini & Hazzan, 2008; algorithms: Cortina, 2007; Guzdial, 2008). This provides further evidence of their importance.

At the K–12 level, definitions have been created for computational thinking, such as the following from *Computational Thinking in K–12 Education Teacher Resources* (CSTA/ISTE, 2011):

> a problem-solving process that includes (but is not limited to) the following characteristics: formulating problems in a way that enables us to use a computer and other tools to help solve them; logically organizing and analyzing data; representing data through abstractions such as models and simulations; automating solutions through algorithmic thinking (a series of ordered steps); identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources; and generalizing and transferring this problem solving process to a wide variety of problems.

Computational thinking found initial footholds in programs in afterschool engineering or robotics clubs, game development environments, and tools that encourage programming "play" with media and storytelling, such as Scratch and Alice. It is clear from the choices made in after-school, free-choice settings such as the Computer Clubhouses (Kafai, Peppler, & Chapman, 2009) that these tools can be attractive means to an end for students, such as in developing media computation skills.

More recently, computational thinking is also finding stronger, more formal footholds in high school with the new AP CS Principles (CSP) framework, the Exploring Computer Science (ECS) curriculum, as well as in the Next Generation Science Standards (NGSS). Indeed, many of the current national-, state-, and district-level efforts to make computer science (CS) count as a high school math or science credit, or as core admissions credit for colleges and universities, use CSP and/or ECS as cornerstones in their arguments. High-profile efforts include the recent adoption of ECS and development of a CSP framework-inspired course by *Code.org* (a nonprofit dedicated to expanding participation in computer science, sponsored by the high tech industry). *Code.org* has already recruited several large school districts for a four year pilot of ECS and AP CS and is offering free professional development, teacher stipends, and outreach/advocacy support (*Code.org*, 2013). *Code.org*'s overall goal is to recruit 100 districts to participate, which would mean that ECS could be implemented in up to 100 additional classrooms in the 2014–15 school year.

Given this increased focus and availability of computer science in the K–12 setting, it is imperative to also address the need for quality classroom assessments and how to support computer science teachers' use of formative and summative assessment. In particular, we need to understand teachers' assessment practices in computer science classrooms, such as what assessments teachers currently use, what are the limitations of these assessment techniques, and what is needed to improve the quality of computer science assessment. In order to better understand the computer science assessment landscape, we identify three representative computer science courses and provide an overview of each in the following section.

Assessment of Computing Concepts

Assessing student understanding in computer science is a challenging task further compounded by the fact that beginner computer science students find learning to program a difficult task (Greening, 1998). Learning to program involves using computational thinking skills, such as problem decomposition, developing algorithms, and abstraction, which makes assessment challenging. How do we measure these concepts using reliable and valid tools that can be used across different programming environments? How do we support teachers' efforts to adapt them to meet their own assessment needs? There have been some efforts in Israel to develop a nationwide exam to measure students' algorithmic thinking (Zur-Bargury, Pârv, & Lanzberg, 2013). In addition to assessing computing concepts, the exam was also evaluated using Bloom's taxonomy (Bloom and Krathwohl, 1956) so the questions were distributed across Remembering and Understanding, Applying and Analyzing, and Evaluating and Creating. In the United States, national efforts to assess student learning have mainly been led by College Board through the AP CSA course and more recently piloting of the AP CS Principles course. Below we focus on three computer science courses and how assessment is addressed across different CS courses including a programming intensive course (AP CSA) and two courses that focus more on computational thinking concepts (AP CSP and ECS). It is important to recognize that there are a number of other computer science courses offered in K–12 schools ranging from courses that focus primarily on computational thinking to courses that are programming intensive using a particular language (Python, C#, VB, etc.).

AP Computer Science (AP CSA)

The AP Computer Science A course is an introductory course in computer science. Because the development of computer programs to solve problems is a skill fundamental to the study of computer science, a large part of the course is built around the development of computer programs or parts of programs that correctly solve a given problem. The course also emphasizes the design issues that make programs understandable, adaptable, and, when appropriate, reusable. At the same time, the development of useful computer programs and classes is used as a context for introducing other important concepts in computer science, including the development and analysis of algorithms, the development and use of fundamental data structures, and the study of standard algorithms and typical applications. In addition, an understanding of the basic hardware and software components of computer systems and the responsible use of these systems are integral parts of the course. AP Computer Science A course represents college-level achievement for which most colleges and universities can be expected to grant advanced placement and/or credit.

The objectives of an AP Computer Science A course are comparable to those in the introductory sequence of courses for computer science majors offered in college and university computer science departments. The content of the college-level introductory programming course has evolved significantly over the years. Starting as a treatment merely of language features, it eventually incorporated first the notions of procedures and procedural abstraction, then the use of modules and data abstraction. At most institutions, the current introductory programming course takes an object-oriented approach to programming that is based on encapsulating procedures and data and creating programs with interacting objects. The current offerings of the AP Computer Science A Exam require the use of Java. Those sections of the exam that require the reading or writing of actual programs will use Java. (Excerpted from the course description for AP CS A from College Board, *http://bit.ly/1cLsGHz*).

The AP Exam for Computer Science A final assessment that the College Board offers is three hours long and seeks to determine how well students have mastered the concepts and techniques contained in the respective course outlines. Each exam consists of two sections: a multiple-choice section (40 questions in 75 minutes), which tests proficiency in a wide variety of topics, and a free response section (four questions in 105 minutes), which requires students to demonstrate the ability to solve problems involving more extended reasoning. The multiple-choice and the free-response sections of the AP Computer Science A Exam requires students to demonstrate their ability to design, write, analyze, and document programs and subprograms. Minor points of syntax are not tested on the exams. All student responses involving code must be written in Java. Students are expected to be familiar with and able to use the standard Java classes listed in the given AP Java subset.

The regular assessment components that teachers are suggested to use for AP Computer Science A appear in class work assignments (such as writing programs either collectively as a class or individually), allowing the teacher to illustrate specific objectives. These assignments are generally brief. Some of the concepts will be reinforced using some take home worksheets. The actual check for mastery is usually done by end-of-unit tests and/or programming projects where the general guidelines for the expected program output for a given input are provided to the students. The teachers use different strategies for assessing students' knowledge and acquisition of skills. Most first time AP Computer Science A teachers attend a week-long professional development training where they learn how to deliver the content and concepts. They are also provided with several strategies for different types of assessments aligned with the actual AP exam. There are various textbooks that are helpful in teaching the concepts for AP Computer Science A. These textbooks generally come with instructional resources.

Computer Science Principles (CSP)

The proposed Computer Science Principles (CSP) course focuses on seven big ideas in computer science that echo the components of computational thinking including abstraction, data and information manipulation, algorithms, and computational artifacts. Accompanying these big ideas are six "computational thinking practices," a term that, by the addition of the word "practices," reflects an orientation toward not just internal, individual "thinking" but "ways of being and doing" that students should exhibit when learning computer science and exhibiting computer science knowledge, skills, and attitudes. The six computational thinking practices include:

- *Connecting Computing:* This CT practice relates to the influence of computing and its implications on individuals and society.
- *Creating Computational Artifacts:* Given the creative nature of computing, this practice allows students to engage in computing by designing and developing computational artifacts.
- *Abstracting:* The third CT practice of the course focuses on students' understanding and applying abstraction to "develop models and simulations of natural and artificial phenomena, use them to make predictions about the world, and analyze their efficacy and validity" (College Board, 2014, p. 3).
- *Analyzing Problems and Artifacts:* CT also involves developing solutions, models, and artifacts for problems, as well as evaluating the appropriateness of the proposed solutions and artifacts.
- *Communicating:* The CS principles course proposes communication (both written and oral) as an important CT practice that allows students to describe the influence of technology and computation supported by data visualization and computational analysis.
- *Collaborating:* Finally, collaboration is a key CT practice, where peers learn to work together effectively to solve ill-structured problems that use computation.

PHOTO: ©WWW.ISTOCKPHOTO.COM/PEOPLEIMAGES

According to L. Diaz (personal communication, May 12 2015) The AP Computer Science Principles (CSP) assessment consists of two parts: a through-course assessment and the end-of-course AP Exam. Both of these will measure student achievement of the course learning objectives as specified in the AP CSP Curriculum Framework. The AP Exam will consist of single- and multiple-select multiple choice questions. The first AP CSP Exam will be administered in May 2017. On both the through-course assessment and the AP Computer Science Principles Exam, students will be asked to apply their understanding of the course learning objectives, including the essential knowledge statements and computational thinking practices.

The through-course assessment includes two performance tasks, one focusing on computing innovations and the other on programming. Each task includes specified instructions that students follow in order to complete the tasks. Students are required to work on these performance tasks during class time. Teachers will have flexibility in deciding when to administer each task. That said, the Advanced Placement® program highly encourages teachers to ensure that students have sufficient and successful learning experiences with the specified content and skills in each task before engaging in the formal administration of the task. This will help support students to successfully complete each of the performance tasks on their own.

The AP CSP performance tasks will not change from year to year. Rather, the tasks are designed to give students broad latitude in personally selecting the focus and topics for their engagement in these tasks. Students will upload digital artifacts and written responses via a Web-based digital application to the College Board for scoring purposes. Draft versions of pilot performance tasks can be found on the Computer Science Principles Pilot Teacher Community (*https://apcommunity.collegeboard.org/web/csprinciples/home/*).

Exploring Computer Science (ECS)

Exploring Computer Science (ECS) is a comprehensive school reform effort for computing education consisting of three elements: curriculum, teacher professional development, and policy efforts (Goode & Margolis, 2011). With the goal of broadening the participation of traditionally underrepresented students in pre-college computer science (9th and 10th grades), it was developed with a focus on urban schools, and created in response to barriers discovered for students taking AP CSA as a first computing class.

ECS is based on an *equitable* learning model: CS concepts, inquiry teaching and learning, and equity and classroom culture are all emphasized. ECS teaches that computer science is creative, that technology is a tool for solving problems, and that computer science has an impact on society. The curriculum is also designed around a core definition of inquiry by Nasir and Hand (2008):

> Inquiry instruction is a set of teaching and learning strategies through which students are expected and encouraged to help define the initial conditions of problems, utilize their prior knowledge, work collaboratively, make claims using their own words, and develop multiple representations of particular solutions" (Margolis et al., 2012).

Another strategy of the curriculum is to develop experiences that are culturally relevant and meaningful to the student population, based on the "funds of knowledge" work of Moll, Amanti, Neff, and Gonzales (1992) and the national Algebra project of Moses and Cobb (2001) that includes community connections to address beliefs about students' abilities to learn mathematics.

Initially focused on the Los Angeles Unified School District (LA Unified), ECS has begun to scale rapidly in response to demand from states, the CS industry, and CS education leaders. However, ECS began to scale despite limited evidence of how implementation impacts student learning. Ongoing small-scale studies of ECS are building an evidence-based argument for its efficacy, but the lack of common student learning measures limited the range of possible insights from these studies. A more systematic understanding of how ECS enactment affects student learning was critical to maximizing the impact of ECS on students' computational thinking skills and the CS pipeline.

Since 2011, SRI Education (SRI) has been partnering with the ECS developers and participating schools on a suite of related projects including Principled Assessment of Computational Thinking (PACT), whose broad goals are to improve CS teaching, learning, and adoption, primarily at the secondary level, by developing high-quality assessments for the ECS curriculum (see pact.sri.com for additional details). As of May 2015, SRI Education has designed, developed, and piloted assessments for ECS Units 1–4 and a cumulative assessment covering those units (see Rutstein, Snow, and Bienkowski, 2014; Bienkowski, Rutstein, and Snow, 2015).

It is important to emphasize that the kind of understanding that ECS students should have about important computational practices goes beyond recalling facts or giving inputs to a program and predicting its outputs. Rather, students should demonstrate "ways of being and doing" when learning and exhibiting computer science knowledge, skills, and attitudes. To this end, the assessment design was based on the model of inquiry-based learning in ECS and the learning objectives underlying each unit. The assessments presume that knowledgeable students should be able to apply, evaluate, and explain what they are learning, among other skills.

The ECS assessments are composed of scenario-based multi-part short answer tasks. While short answer questions do place more demands on students' ability to read and understand the scenario and interpret the instructions, the ability to communicate thinking through explanations is a critical skill for students to demonstrate in an inquiry-based curriculum such as ECS. To support students' understanding of the tasks, task information is often represented in multiple ways, using diagrams and pictures in addition to text.

The ECS assessments are currently designed for paper and pencil delivery. Students produce short answers in response to scenarios, commonly in order to explain their reasoning. SRI Education is currently designing and developing online versions of the assessments for ease of administration, scoring and reporting, and to test more authentic computational practices (i.e., using more dynamic interfaces). SRI Education will also be piloting multiple-choice (MC) assessment items starting with the online assessments in Fall 2015. The MC items will measure students' conceptual knowledge that is needed for the multi-part, scenario-based tasks.

The Computer Science Assessment Landscape Study

Given how varied the assessment is across the three courses discussed above, it becomes imperative to better understand how CS teachers are assessing their students' computing knowledge and skills. Hence, the Computer Science Teachers Association (CSTA), with funding from Google, undertook a study to:

1. provide greater understanding of the current state of 9–12 computer science assessment within the US,
2. determine challenges CS teachers face when assessing student understanding of computing concepts, and
3. ascertain the role of CS teachers in assessment design and development.

In order to accomplish these goals, CSTA conducted a landscape study of current US high school CS teachers' experiences with, and attitudes toward, assessment of

CT skills in their classrooms. The study consisted of a survey of all CSTA teachers and follow up phone interviews with a subset of these teachers. The CSTA assessment landscape committee undertook this research during Fall 2014 to examine computer science teachers' assessment practices. Specifically, the committee focused on the following two questions:

1. What assessment mechanisms do CS teachers use in their classrooms to examine student learning?
2. What are the challenges that CS teachers face when assessing student learning in computer science classrooms?

Methodology

During Fall 2014, CSTA members were invited to complete a background survey to collect demographic information about CS teachers' experiences in teaching CS courses. Specifically, the survey was used to collect data from teachers on the following variables:

- Computing courses taught
- Grade level taught
- Programming languages taught
- Years of teaching experience
- Years of CS teaching experience
- Type of school taught (Private vs. Public)
- Ethnicity
- Gender

The background survey was completed by 553 respondents. Teachers were then selected from the survey respondents for a follow up interview for an in-depth examination of their assessment practices. The teachers were selected based on years of CS teaching experience and training in teaching computer science as well as where they taught (public vs. private) to ensure a representative sample with diverse teaching backgrounds. Additionally, we used *data saturation* as the factor in determining an adequate sample size, which means that data do not shed any additional light on the phenomena of interest (Robinson, 2014). Specifically, we decided to conduct interviews as long as the interviews offered new insights into teachers' assessment practices and stopped gathering data once consistent themes emerged (Charmaz, 2006). Based on the data saturation concept, we interviewed 25 teachers for in-depth interviews to address the questions outlined above.

The interview sample included 25 computer science teachers with 13 female and 12 male teachers. The participants include 14 Caucasian, five African American, three Asian/Pacific Islander, two Hispanic, and one Native American teacher. Twenty two of the teachers taught at public schools and three teachers taught at private schools. In terms of teaching experience, seven teachers had taught for three years or fewer, eight teachers had taught between three and nine years, and 10 teachers had taught for over nine years. However, the majority of teachers (N=15) had fewer than three years of computing teaching experience with five teachers each having experience between three and nine years, and five teachers with over nine years of experience. Finally, the majority of the teachers (N=23) taught CS as an elective course while only two teachers taught CS as a required course.

Data Collection and Analysis

Teachers were interviewed using a semi-structured interview protocol to gauge their perspectives on assessing student learning. The interviews were conducted via phone, Google Hangouts, or Skype and were recorded for data analysis. The interview questions asked participants to share their thoughts on how they evaluated student learning and how they developed course assessments. The survey also asked participants to share their thoughts on the challenges of assessing student learning in the classroom. See **Appendix A** for the interview questions.

The committee members met to discuss the interview protocol and procedures for conducting the interviews. Each committee member involved in the interview process read "Learning About Others Through Interviewing" (Litchman, 2006) for training on qualitative interviewing and gathering information from the participants. The interviewers took notes during the interviews and later transcribed the interviews. The interview notes and transcripts allowed us to develop analytic codes and categories. This also allowed us to make decisions about data saturation and appropriate sample size (Robinson, 2014). The interviewers met regularly to discuss the interviews and share any initial thoughts about patterns and themes they saw emerge from the data. In order to establish interrater reliability, another rater coded two-thirds of the themes, which led to 88% agreement. The interviewers developed a set of initial themes that were shared using a Google Doc, which was continuously revisited to inform data collection and analysis. Once the interviewers saw similar themes emerge from the teacher interviews, we met to collapse the themes into overarching categories.

Results

The goal of the study was to develop an understanding of computer science teachers' assessment practices and challenges. The qualitative analysis of the interviews generated a number of themes, which were collapsed into three overarching categories: types of assessments, challenges of assessment, and resources needed for assessment. Below, we present each of the overarching categories.

Types of Assessment

One of the main overarching categories that emerged from the analysis of the data suggested that computer science teachers use a number of approaches to assess student learning in their classroom, structured around two main themes: (1) formative assessment; and (2) summative assessment. Within summative assessment, teachers reported using a range of techniques to measure student understanding that could be categorized into three sub-themes: (a) multiple-choice assessment items; (b) open-ended assessment items; and (c) rubrics to grade assignments.

Formative Assessment

One of the themes that emerged from the interviews was teachers' use of formative assessments in the classroom to monitor student understanding of computing concepts. A majority of the teachers (N=18) discussed using a number of formative assessment approaches, such as writing short programs to gauge their understanding of concepts. For example, one teacher stated,

> The assessment [formal] that happens during the class time is that we write programs either as a unit collectively or I model them before. And then there are work periods during the class where I assign problems [for students] to write some programs. I will go around and get each student to explain their code to me...So the formative assessments are in terms of reading code.

Teachers also discussed using small quizzes at the beginning of each class as a formative assessment tool. One teacher stated, "Every day when the students come in, the first five minutes of class we'll have a review question but it is review like a deeper learning comprehension level. They are allowed to help each other and work together on it."

Formatively assessing students by looking over students' shoulders is another technique used by a number of teachers (N=7) as students work on completing their projects. This is highlighted by one teacher, who said,

> My assessment of them is ongoing by just looking over their shoulders. Half (of the class time) is spent doing labs, so I am walking around looking over their shoulders. My assessment is really looking at them and seeing what they can do. I'm watching to see when they need help.

A number of teachers (N=6) also discussed using student response systems, such as Poll Everywhere (*http://www.polleverywhere.com*), clicker questions, and the Socrative (*http://www.socrative.com*) application to check in-the-moment student understanding. One teacher stated,

> Using the Socrative application online, I can see the answers as they come in. And then I have to judge as a teacher whether or not the students understand the concept and if they don't I go into more detail and review it again with them if I feel like they are not understanding it. If I see correct answers on my screen, then I move on with the next topic and the lecture.

Poll Everywhere is another tool teachers use to check student understanding during in-classroom time. As one teacher noted:

> Throughout class I use Poll Everywhere. I use that a lot in class. At the end of class I also use an exit ticket. Those exit tickets can be done in a variety of ways. Shared Google Docs, email, or they can do a fabulous drawing and illustrate what they are trying to tell me. I try to give them a variety of ways and have even used text messaging or have them send me a voice memo.

Participants also noted that their use of Learning Management Systems (LMS) such as Classcraft (*http://www.classcraft.com*) has changed the way they monitor student progress in the course, helping them make sure that students are meeting learning goals:

> I had some (students) not doing their homework, not doing their homework, not doing their homework, and I struggled. I knew that two of the boys were gamers. I got to tell you what, when you use that thing and they know they get gold coins for being here before the bell rings, they get set and ready to go. They know that they get experience points or help points for answering certain types of questions in class and they have to defend their answers. This is a game. This is a war. You have to defend your answer in order for it to be right. That has changed the whole dynamic of the room. They will fight to get those points and they get extra points if homework is done by midnight. This gives me time the next morning to look over it and see what holes I need to fix for the next day.

Additionally, this teacher discussed using EduCanon (*http://www.educanon.com*) as a formative tool in her flipped classroom, stating:

> For the AP Computer Science class only, I flipped the classroom and they have videos they watch. I use a source called EduCanon so that it

PHOTO: ©WWW.ISTOCKPHOTO.COM/SKYNESHER

> stops them and sometimes I ask them to write out syntax and the sometimes I ask them to explain a piece of code and sometimes I ask them to regurgitate what they just heard. Then I get up at four every morning to aggregate those data to look at what they missed and what holes I need to fill.

A beginning CS teacher also mentioned an additional learning management system:

> I use a learning management system called Haiku (*https://www.haikulearning.com*). It is mostly self-paced in that I try to follow a mastery type of model. They submit everything in the learning management system and I can check it as I go along. I can give a grade of A, B or resubmit so that they have the chance to resubmit until they have mastered the concept.

In summary, teachers discussed a number of formative assessments as a part of their everyday instruction and how they used the data in their own teaching. Mainly teachers discussed using formative assessment to collect information on whether students were meeting the learning goals of the lesson, which teachers use to determine future instruction. On the other hand, teachers used summative assessment to collect aggregate data on student learning and whether students learned the material at the end of a unit or course.

Summative Assessment

Within types of assessment, teachers discussed how they use summative assessment in their classes to evaluate student learning at the end of a unit. The goal of a summative assessment is to provide students with information concerning what they learned and how well they mastered course concepts. Summative assessments are also used to assign grades in the course. In other words, formative assessments focus on the process of learning while summative assessments are primarily used to assess the final product (i.e., projects, tests, etc.). Teachers indicated that they used three distinct types of summative assessments that enabled them to determine how well students understood computing concepts at the end of a unit. The three types of summative assessments reported are: rubrics, multiple-choice, and fill-in-the blanks.

Multiple-choice

A majority of the teachers (N=19) reported using multiple choice questions to evaluate student learning at the end of each unit as well as toward the end of the course. However, a number of teachers noted that using multiple choice questions to assess student understanding can be problematic because not every student is good at taking multiple choice tests and because multiple choice questions sometimes involved guesswork on the part of students. The following response from a teacher on summative assessment, in general, highlights the tensions that using multiple choice questions creates for practitioners.

> For summative assessment there is a chapter test for every chapter. Again with those I like to try to make them as close to the AP exam as possible. So I try to have them approximately rated in points (weighting) multiple choice questions and open response questions. On the lower end of the spectrum there are students that basically aren't good at taking tests and don't write enough code so they don't do very well on either [multiple choice and open response questions]. Then there is another group which isn't so good at taking tests [multiple choice], but they do very well on open responses because they know how to write code. So what I am trying to do is to get the multiple choice people to be better at writing code and get people who are better at writing code to be better at multiple choice.

Another teacher also reported that multiple choice questions are inauthentic and students can guess the responses. She stated,

> Students do not do well on multiple choice because of guessing… they [also] do not necessarily do well on them because students don't see the multiple choice as authentic saying, 'in the real world, we can just Google this.'

The teachers indicated that they typically developed their multiple choice questions using test banks from textbook publishers or questions from previous AP exams, as highlighted by the following comments:

> I rely strictly on my textbook publisher for the multiple choice questions. The book has materials that came with it, so I am using their multiple choice for now.

> [I] use the question banks provided by the textbook and old AP exams.

Teachers' views on the use of multiple choice questions highlighted the challenges they face in accessing quality assessment items that accurately assess student learning. In addition to multiple choice questions, the teachers reported using short open-ended questions to measure their students' cumulative understanding at the end of a unit.

Open-ended/Fill-in-the-blank/Free response

Another summative assessment technique the teacher participants (N=11) mentioned involves using open-ended and free response questions, such as writing code. They noted that students who understand the concepts are good at open-ended responses. For example, one teacher stated:

> Students that understand things conceptually and are good at test taking do great on multiple choice questions. They also understand how to write code because they do it a lot and they conceptually understand so they write good open-ended responses.

Another teacher opined that open-ended questions are the most effective technique assessing student understanding:

> We do quizzes and the quizzes focus primarily on vocabulary. I want students to have a command of the vocabulary of programming in general and try to focus on vocab language from computer programming that would apply to any language. I want to make sure that students know those terms. Also students need to be familiar with syntax. So quizzes have to do with the names of the functions and keywords in the Python language. I prefer doing short answer rather than doing multiple choice. The multiple choice questions are too easy for students to guess the right answer. By forcing the students to fill in the blank, it really separated the students who know the material quite well from students that have a weaker understanding.

The same teacher observed that even students who are comfortable with open-ended questions have difficulty writing actual code if they do not have access to an Integrated Development Environment. The teacher stated,

> Some students understand conceptually and know how to take tests and do very well on multiple choice. But when it comes to writing code without their IDE, because it is a pencil and paper test, they don't know how to do it. They basically fumble on the open responses.

Open-ended questions allow teachers to assess students' problem solving skills and their creativity as one teacher noted, "My rubrics list what they need to do on their projects and then I use creativity as the difference between a B and a B+." The use of open-ended questions as a part of the summative assessment allowed teachers to determine whether students had a conceptual understanding of the topics. The open-ended questions also seemed to address the limitations of multiple-choice questions, which could be answered with guesswork.

Rubrics

A number of teachers (N=16) discussed using rubrics to assess students' programming projects at the end of a unit. They suggested that these rubrics served two purposes. First, teachers stated that the rubrics served as an summative assessment evaluation for the end-of-unit programming projects/assignments. Second, they used the rubric as an objective tool for setting expectations for students' projects and what students needed to accomplish. In this sense, the teachers indicated that they used the rubric at the beginning to convey to students what was expected on their projects and then used it later to grade the students' work. One teacher reflected on the use of rubrics, stating:

> The programming assignments in the lab are where I use rubrics. I give the labs and give them plenty of time to do it. We have an LMS system where I can let them submit their work. I give them class time, but their due dates are sometime over the weekend. I pull in the programs, recompile them, look at their code and look at their solution. It is somewhat subjective on my part. If you have use a bunch of ifs (if-then-else statements) when there was a slicker, more elegant solution, inefficient looping and things like that. The rubric has some basic areas: Completion (does it do what it's supposed to do?), Run time (Does it actually run and are the user prompts appropriate?), Coding standards (I like them to use comments and basics of documenting their work.)

ASSESSING STUDENT LEARNING IS PARTICULARLY CHALLENGING WHEN THERE ARE A MULTITUDE OF POSSIBLE METHODS AND PROGRAMMING SOLUTIONS FOR A SINGLE PROBLEM, ALL OF WHICH ARE EQUALLY CORRECT.

Another teacher indicated a similar preference for using rubrics to set expectations for students before they begin their programming projects, stating: "I give them a guideline on the rubric." The use of scale within the rubric to grade student work was a key aspect for summative assessment, as reflected in the following teacher comment:

> I have a rubric [to evaluate student work], it has to first of all generate correct results. Then the second thing is you can now start to think about other things, like do you understand the algorithm, can you generalize this algorithm....the actual algorithmic efficiency. We focus on syntax, semantics, and style. So on a four point rubric—one is that they hardly did anything, two is that the program won't compile although they did some work, three is that the program will compile, but generate incorrect results, and four is that the program compiles and is semantically correct. Then we talk about style after that—did they do the Javadoc, variable naming, some of the more esoteric things.

When discussing how rubrics were developed, teachers stated that they developed the rubrics on their own or adapted them from other teachers. For example, one teacher noted:

> I try to do programming projects and use rubrics to say 'this is what I am looking for.' And then they [students] know this is how it will be graded. I copied it [the rubric] from a college-level computer science rubric and then adapted it to what I am doing.

Hence, teachers used rubrics not only to grade those projects, but also to allow students to understand what was required to successfully complete the project.

CS Assessment Challenges

During the interviews, teachers also discussed challenges of assessing student learning that they perceive are unique to computer science (or at least far less common in other academic disciplines). Specifically, participants pointed out that there is no one correct strategy or one correct solution to a given programming challenge. They also noted that the availability of programs online makes it difficult to identify and assess original student work. Below we discuss each of these challenges.

Unique nature of computer science

Teachers (N=10) reported that computer programs were the most challenging artifact to assess for several reasons, including the uniqueness of programs and the time and resources needed to grade the volume of programs. For example, teachers reported that assessing student learning is particularly challenging when there are a multitude of possible methods and programming solutions for a single problem, all of which are equally correct. One teacher stated,

> The main thing with computer programming, if you were in the field there could be five different ways to code the answer to a problem and it may be right, but standard-wise it may not be right, but they still got the right answer. As a teacher it is easy for me to see that, but I could see that a general education teacher might not see that. I maybe have 30 students, then I have 30 types of code for one program.

Furthermore, the study participants reported that programming students produce a large volume of code and the time required to grade students programs can be overwhelming. As one teacher noted: "Having the time to assess the concepts, the syntax, and the projects and depth of understanding, there is so much there to assess and for me to find the time to test it all is very difficult." Another teacher shared a similar view of grading stating, "It takes me forever to get projects graded. I feel bad. I like to have them all done at one time so that I am fair and objective in the grading."

Other teachers reported that they struggled with divergent ideas within a single assignment, finding it hard to grade solutions written in unexpected ways. Teachers in mastery learning environments said they struggle with students falling behind. One teacher is "finding [differentiation] difficult … and making sure they are all ready to go onto the next task."

Ubiquity of the programs

Some teachers also stated that the availability of programs online makes it difficult to assess student understanding of computer science concepts as demonstrated in programming assignments and projects. One teacher stated,

> A major [assessment] problem is the non-uniqueness of the computer programs and the ubiquity of them on the Internet. One of my former students sent me something that a professor at his university gave to his students and said "if you get it off the Internet it will be noticeable, the TA will know, I will know, and you will make this sad kitten even sadder" and had a picture of it. I tell them that you learn coding—by looking at other people's and modifying it, manipulating it, and looking to see how they approached the algorithm.

Additionally, teachers who re-used assignments from year-to-year and from shared sources found that students often were able to search the Internet for solutions, invalidating the assignment as a measure of student mastery.

The challenges CS teachers face in assessment have created a need for additional resources not yet commonly available and in some case, not yet developed. The next section discusses the resources teachers indicated that they believe are needed to improve assessment in computer science.

Resources Needed

Analysis of the interview data suggested a need to develop assessment resources for teachers. Specifically, teachers (N=11) discussed the need to develop variety of resources that would improve their ability to accurately assess student learning in CS. Overall, the interview data suggested a pressing need for teacher access to develop online assessment repositories, auto graders, computer-based simulations, and professional development to learn about assessment.

Teachers also expressed the need for an online central repository of valid assessment items that could serve as a one-stop source of CS questions and assessments. One teacher stated,

> Just something that has anything that is like Khan Academy, that has more questions related to computer science and computer programming. Something that has prebuilt questions. You can search the topic and they have a database full of programming questions that are related to looping, arrays, abstraction, so that we can search. Right now, there is so much CS teachers have to do initially, the resources are there, but the questions, the assessments, all that stuff has to be made. For computer science teachers having some of these questions or some kind of platform that is available to us would be absolutely wonderful because that is one thing I have not come across.

Another teacher echoed the desire for an online assessment repository, noting that such a resource would allow him to to easily assemble test items for specific areas, stating, "I wish I would have more access to the available online resources for putting together exams for specific subject areas."

Teachers also suggested that autograding tools, such as CodingBat (*http://codingbat.com*) would help them grade and check for plagiarism at the same time. According to one study participant:

> All CS teachers are desperate for autograders. There should be something where we can..I have tried a bunch of different ones, I tried iJava, which is fine but you have to match their curriculum.... I developed a lot of my problems on CodingBat, but the problem is that the test cases are limited...it's really very very small algorithms...it's as if you are testing algorithmic statements rather than methods.... That [autograder] is desperately needed by us because grading this stuff takes forever.

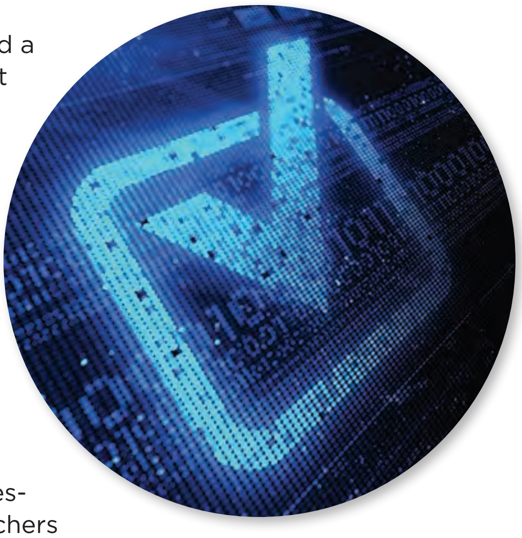

PHOTO: ©WWW.ISTOCKPHOTO.COM/TEEKID

Another participant expressed the need for "Building projects, ones that start at one level and then build on complexity." Similarly, another teacher proposed "I'd like something that gives the students an opportunity to try out their solution online and gives them variations on the problem until they get it right so they're not just guessing until they find a correct answer."

Finally, teachers using languages other than Java were interested both in having assessment resources provided to them as well as professional development opportunities to create their own. Specifically, teachers indicated that they would like assistance with creating assessment tools to meet their classroom needs. For example, one teacher noted, "I am a beginning programmer myself so I would like to see more direction [on creating assessment] instead of just throwing together what I think is good." The teachers who teach use Python and Scratch as languages and environments particularly would benefit from these additional resources.

Conclusion

The results from this study highlight assessment practices that high school teachers use to examine student understanding of computing concepts as well as challenges they face when assessing student learning in their classrooms. Specifically, teachers discussed various formative (such as short code review lab assignments, clicker systems, etc.) and summative assessment practices (such as multiple choice questions, rubrics to evaluate programming projects, etc.) they use to measure student learning. Teachers also reported a number of challenges that they perceive are unique to assessment in computer science classrooms, including the number of resources and time required to develop problems/assignments and grade student work. Additionally, teachers discussed the need to have access to online resources, specifically a curated assessment repository where they could find assessment items for specific computer science concepts.

The study participants also indicated that they are currently using a piecemeal approach to developing assessments, including textbook based test banks and adapted rubrics; however, these sources are commonly not aligned to national standards such as the *CSTA K–12 Computer Science Standards* (*http://csta.acm.org/Curriculum/sub/K12Standards.html*). The results from this study serve as a call to action for the computer science education community to develop valid and reliable assessments that are aligned to the *CSTA K–12 Computer Science Standards*. Such a common assessment library would allow teachers to search for items that are aligned with specific CSTA standards and computing concepts.

The need for assessment resources is further highlighted by the push to use assessments to evaluate teacher effectiveness. For example, in Ohio, teachers who are not teaching Value-Added courses must create Student Learning Objectives (SLOs) (*http://education.ohio.gov/Topics/Teaching/Educator-Evaluation-System/Ohio-s-Teacher-Evaluation-System/Student-Growth-Measures/Student-Learning-Objective-Examples*). Computer Science is currently not a Value-Added course so CS teachers are required to create SLOs using recognized standards. From these standards, a pretest and posttest is created by the individual teacher. The pretest is administered to students and the teacher then writes the SLO detailing how the teacher plans to meet the expected district chosen growth target on the posttest. When creating the pre and posttest, the teacher needs to link the assessment items to recognized (national or state) standards.

Teachers not teaching the AP CS course typically link their curricula to the *CSTA K–12 Computer Science Standards*. However, there is no corresponding CSTA standards-based assessment for teachers to examine student learning. The lack of assessment items leads teachers to develop their own assessments, which do not always accurately assess student learning. The problem is further exacerbated because teachers know very little about assessment and "are inadequately trained and ill-prepared to develop, administer, and interpret the results of various types of assessments" (Koh, 2011, p. 257). Hence, there is an imperative need to develop computer science teachers' assessment literacy through professional development as well as have a curated collection of valid and reliable assessment items that address specific CSTA standards.

Given the lack of assessment literacy among teachers due to inadequate training (DeLuce & Klinger, 2010; Popham, 2009), there is a need to provide them with additional support and resources for developing valid and reliable items. The CSTA Assessment Task Force recommends that quality professional development needs to be provided for computer science teachers to improve their assessment literacy, specifically in designing and implementing assessments at the classroom level (Koh, 2011). Specifically, the CS community should *design and deliver professional development opportunities that prepare computer science teachers to be competent to not only develop high-quality assessments and rubrics*, but also make sound judgements about student performance (Koh, 2011). Researchers have suggested that one-time professional development workshops have limited value and successful professional development involves localized, ongoing, and sustained efforts (Cohen and Hill, 1998; Koh, 2011). CSTA, with its wide network of chapters at the national and international level, could develop a professional development model (and resources) that could be adapted for local needs.

The professional development to foster assessment literacy for computer science teachers should include both summative and formative assessments. The focus on summative assessment would allow teachers to use "assessment-based evidence when arriving at decisions about already-completed instructional events" (Popham, 2009, p. 5). This evidence could be end-of-unit tests or end-of-course tests to examine the effectiveness in increasing students' understanding of computer science concepts. On the other hand, the focus of formative assessment professional development should prepare teachers to develop assessments and collect data that allow them to "adjust their ongoing instructional activities, or by students to adjust the ways they are trying to learn something" (Popham, 2009, p. 5).

Beyond the professional development opportunities, we recommend that the *CS education community lead the development of a curated assessment library for teachers*. One such example of assessment repository is the American Association for the Advancement of Science (AAAS) (*http://assessment.aaas.org/pages/home*), which allows teachers to create accounts and search for assessment items according to topics and subtopics. The site also provides information as to the percentage of students in middle and high schools having answered the questions correctly and further breaks the statistics down into correct answers by male, female, English speaking, and non-English speaking students. A similar effort to develop an assessment repository for computer science teachers is needed, which would allow them to search for assessment items by a CSTA standard and substandard. A search of a site such as this would provide results with project assessment items, and rubrics. Teachers could also contribute assessment items that could be vetted by a research/evaluation committee. The assessment professional development opportunities discussed previously could include components to train teachers on the use of the repository as well as how to contribute their own assessment artifacts.

In summary, we recommend that the computer science education community undertake substantial efforts to create or collect high-quality assessments aligned to the *CSTA K–12 Computer Science Standards* with the specific goal of ensuring participation and success of all students, and also to provide guidance and support to teachers to adequately reflect on the instruction and spend less effort trying to find resources or different kinds of assessments.

Definitions

Computer Science is the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society."—*ACM/CSTA Model Curriculum for K-12 Computer Science*

Computational Thinking provides a framework and vocabulary for computational thinking that will resonate with all K–12 educators. ISTE and CSTA gathered feedback by survey from nearly 700 computer science teachers, researchers, and practitioners who indicated overwhelming support for the operational definition.

A Performance Task is "a goal-directed assessment exercise. It consists of an activity or assignment that is completed by the student and then judged by the teacher or other evaluator on the basis of specific performance criteria" (Source: North Central Regional Educational Laboratory, n.d.).

References

Bennedssen, J., & Caspersen, M. (2008). Abstraction ability as an indicator of success for learning computing science? Paper presented at the ICER '08 Proceeding of the Fourth International Workshop on Computing Education Research.

Biekowski, M., Rustein, D., & Snow, E. (2015). *Computer Science Concepts in the Next Generation Science Standards*. Paper presented at the 2015 annual meeting of the American Educational Research Association (AERA), Chicago, IL.

Bureau of Labor Statistics. (2012). Occupational outlook handbook: Computer and information technology occupations, March 29, 2012. Available from *http://www.bls.gov/ooh/Computer-and-Information-Technology/*

Code.org. (2013). *Code.org pilot district partnership model*. Retrieved from *http://code.org/files/DistrictPartnershipPlan.pdf*

Colburn, T., & Shute, G. (2007). Abstraction in computer science. *Minds and Machines, 17*(2), 169–184.

Cortina, T. J. (2007, March). *An introduction to computer science for non-majors using principles of computation*. Paper presented at the 40th ACM Technical Symposium on Computer Science Education: SIGCSE '07, Covington, KY.

CSTA/ISTE (2011). Computer Science Teachers Association and the International Society for Technology in Education: *Computational Thinking Teacher Resources, Second Edition*. Available from *http://csta.acm.org/Curriculum/sub/CompThinking.html* or *http://www.iste.org/learn/computational-thinking.aspx*

DeLuca, C., & Klinger, D. A. (2010) Assessment literacy development: identifying gaps in teacher candidates' learning. *Assessment in Education: Principles, Policy & Practice, 17*(4), 419-438

Denning, P. J. (2009). The profession of IT beyond computational thinking. *Communications of the ACM, 52*(8), 28–30.

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: the Exploring Computer Science program. *ACM Inroads, 3*(2). doi:10.1145/2189835.2189851

Goode, J., & Margolis, J. (2011). Exploring computer science: A case study of school reform. *ACM Transactions on Computing Education, 11*(2), 1–16. doi:10.1145/1993069.1993076

Greening, T. (1998). Computer science: Through the eyes of potential students. Proceedings of the 3rd Australasian conference on Computer science education.

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM, 51*(8), 25–7.

Hazzan, O. (2003). How students attempt to reduce abstraction in the learning of mathematics and in the learning of computer science. *Computer Science Education, 13*(2), 95–122.

Kafai, Y. B., Peppler, K. A., & Chapman, R. N. (Eds.). (2009). *The Computer Clubhouse: Constructionism and creativity in youth communities*. New York, NY: Teachers College Press.

Koh, K. H. (2011) Improving teachers' assessment literacy through professional development, *Teaching Education, 22* (3), 255–76.

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

Margolis, J., Ryoo, J. J., Sandoval, C. D., Lee, C., Goode, J., & Chapman, G. (2012). Beyond access: broadening participation in high school computer science. *ACM Inroads, 3*(4), 72–8.

Moll, L., Amanti, C., Neff, D., & Gonzalez, N. (1992). Funds of knowledge for teaching: Using a qualitative approach to connect homes and classrooms. *Theory Into Practice, 31*(2), 132–41.

Moses, R. & Cobb, C. (2001). *Radical equations: Math literacy and civil rights*. Boston, MA: Beacon Press.

Nasir, N.S. & Hand, V. (2008). From the court to the classroom: Opportunities for engagement, learning, and identity in basketball and classroom mathematics. *Journal of the Learning Sciences 17*(2),143–79.

National Research Council. (2004). *Computer science: Reflections on the field*. Washington, DC: The National Academies Press.

National Research Council Committee on the Workshops on Computational Thinking. (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: National Academy Press. Available at *http://www.nap.edu/openbook.php?record_id=12840&page=R1*

Nicholson, K., Good, J., & Howland, K. (2009). *Concrete thoughts on abstraction*. In Proceedings of Psychology of Programming Interest Group (PPIG 2009), pp. 38–47, Limerick, Ireland.

North Central Regional Educational Laboratory. (n.d.). Retrieved May 13, 2015 from NCREL *http://www.ncrel.org/sdrs/areas/issues/methods/assment/as8lk44.htm*

Popham, W. (2009). Assessment literacy for teachers: Faddish or fundamental? *Theory Into Practice, 48*, 4-11.

Rustein, D., Snow, E., Biekowski, M. (2014). *Computational thinking practices: Analyzing and modeling a critical domain in computer science education*. Paper presented at the 2014 annual meeting of the American Educational Research Association (AERA), Philadelphia, PA.

Sakhnini, V., & Hazzan, O. (2008). Reducing abstraction in high school computer science education: The case of definition, implementation, and use of abstract data types. *Journal on Educational Resources in Computing (JERIC), 8*(2), 1–13.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–5.

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366*(1881), 3717–25.

Zur-Bargury, I., Pârv, B., Lanzberg, D. (2018). A nationwide exam as a tool for improving a new curriculum. ITiCSE '13 Proceedings of the 18th ACM conference on Innovation and technology in computer science education

Appendix A: Interview Protocol

1. Do you align your CS courses with any standards? If so, what are they?
   - For example, CSTA, State standards, common core, etc.

2. How do you evaluate student learning in your course(s)?
   - What things do you do on a daily basis to measure student understanding?
   - When you've finished a unit or a project, what ways do you measure student competency?

3. Can you talk about how you develop your course assessments?
   - Did you come up with the questions?
     - If yes, how did you come up with the questions?
     - If no, where did you get your assessment?

4. What professional resources do you use to support your assessment needs?
   - Are there resources you use to develop assessments? If so, what are they?
   - Are there other resources you use to support your assessment practices more broadly? If yes, what are they?
   - Do you feel you have an adequate amount of available resources for CS assessment?

5. What assessment related resources would you like to see developed?

6. Do you use a textbook in your course?
   - Was the textbook your choice, or assigned by your school admin.?
   - To what extent do you use the assessment materials included with your textbook? To what extent do you adapt/modify these resources?
   - Do you supplement your text with other assessment resources?
   - If you don't use a textbook, what evaluation resources do you use?

7. To what extent do your students score themselves, have their peers score them, or have you score them?
   - How do students know how they are to be scored?

8. Are there any obstacles to assessing student learning in computer science? If so, can you please elaborate.
   - If you have experience teaching another subject area, what are the challenges unique to assessing student learning in CS?

9. Can you elaborate more on your assessments practices?
   - Project-based?
   - Multiple-choice?
   - Presentation-mode?
   - Code-review?
   - To what extent do you use multiple-choice versus other methods to measure student learning?
   - In what ways are your students most comfortable/effective in expressing what they know?
     - What ways seem to make students most uncomfortable in expressing themselves?

10. Do you have students with IEPs and/or 504 plans enrolled in your CS course(s)?
    - If yes, how do you address the assessment needs of these students?

11. In what ways do you measure creativity in your classroom?
    - How do you determine the extent to which a student's work is creative?

12. In what ways do you measure collaboration in your classroom?
    - How do you determine extent to which a student's work is collaborative?

13. In what ways do you measure inquiry in your classroom?
    - How do you determine the extent to which a student is engaging in inquiry?

14. Describe the different ways you use student evaluation data to inform your work and your students'.
    - Actions you take in the present
    - Actions you intend to take in the future
    - Actions your students take as a result of evaluation data

Notes

csta | Computer Science Teachers Association

Computer Science Teachers Association
2 Penn Plaza, Suite 701
New York, NY 10121-0701

T: (212) 626-0507
F: (541) 636-4655
E: *l.clayborn@csta-hq.org*

www.csta.acm.org