Research Imperatives for CS Education in K-12
Shuchi Grover, Stanford University

In the flurry of recent research on CT and computing education in K-12, much attention has been focused on tools and environments believed to foster computational competencies. Additionally many free online venues that teach programming languages, as well as online tutorials and curricula to teach CS have mushroomed over the last couple of years. However, there has been little to no empirical inquiry to compare the effectiveness of the various tools and curricula that are popularly believed to help develop CT skills in the context of different grade levels in K-12 educational settings and diverse student populations. Additionally, as more resources and curricula move online, they must be tested in various online, face-to-face, and blended learning settings in order to establish best practices and guidelines for their use in classrooms. Lastly, issues of assessment deserve more attention than ever. How can research assist in developing and testing assessment measures that are programming language-agnostic and an accurate measure of student understanding of computing constructs?

Investigations aimed at addressing the following related research questions will facilitate age-appropriate use of freely available tools and curricula accompanied by a cognizance of how best to leverage them in K-12 classrooms across the country for structured CS education. Additionally, providing CS teachers with assessment measures appropriate to specific tools as well as those that test for deep learning of computational thinking that goes beyond learning of specific programming environments and syntax will aid in building required computational competencies and prime students for success in future experiences in computing and other fields that require computational work.

Research Imparative 1: Systematic evaluation of different tools and curricula for CS Education at various levels of K-12 education, and a framework for evaluating and comparing these various resources.

Scratch, Alice, Blockly, MIT App Inventor, Game Maker, Kodu, Greenfoot, and Stagecast are well-known among many easy-to-use graphical and visual programming tools that are favored by researchers and educators attempting to introduce younger children and novices to computing. Text-based environments such as LOGO are old favorites among some, and Python is also becoming increasingly popular. In addition, robotics kits, such as LEGO Mindstorms: tangible media such as Arduino and Gogo Boards; and Web-based simulation authoring tools such as Agentsheets and Agentcubes are also believed to encourage game-design and robotics projects that help with exploration of computational thinking.

In order for research to methodically impact practice and influence introduction of computing education in K-12 classrooms, it is essential that there be robust understanding of what each of the tools suggested for inclusion in computing curricula brings to the table, how the various tools stack up in allowing both boys and girls from all sections of society to engage with the identified component elements of CT, and how shortfalls, if any, in a tool may be compensated for by appropriately combining with other computational experiences.
More specifically, do the various computational tools and environments in use today excel (or fall short) along various dimensions compared to others when used as a means to introduce CT skills in children? If so, what are these dimensions and distinctions? Computational learning experiences are distinct by virtue of the artifacts children create, the depth of complexity computational projects demand, and learner engagement, which are in turn driven by the inherent affordances of the computational tool in use (Grover, 2013, AERA paper). What specific elements of CT, for example, are children engaging in while creating an e-Textile project with the Lilypad Arduino versus designing a game in Scratch versus building a robotics system using Gogo Boards versus telling a story using Alice or developing a mobile app using MIT App Inventor? Are they any lacunae in dimensions of CT that these popular tools expose kids to? If so, what are these, and how should any shortfall be met through additional curricula using other tools? What are the logistical barriers to using tangible tools that require physical components?

Recently released online venues like Tynker, CodeHS, Khan Academy and CodeAcademy tout sequences of curriculum that can be used by K-12 learners to learn computational thinking and programming. Tynker, a Scratch-inspired computing platform aims to teach computational learning and programming skills to children of all ages. Unlike Scratch, it offers a set of structured lessons, that allow students to earn badges by progressing through levels, and there are also quizzes in addition to tutorials. The classroom version has analytics to support the teacher in assessing student progression and learning. CodeHS video-based curriculum claims to be a “class in a box” that starts with programming in Karel, then it introduces the basics of JavaScript, and then teaches simple graphics and animations. Khan Academy’s recently launched CS curriculum has undergone a “tutorialization” that includes a mix of videos and coding "talk-throughs". CodeAcademy provides learners an interactive interface so they can “learn to code” Javascript, HTML/CSS, PHP, Python, Ruby, and APIs, all for free.

How do these structured curricula stack up, and how well do they accomplish curricular goals and standards as outlined by ACM CSTA for various grade levels? What are classroom experiences of teachers using these tools? Unfortunately, there is no research yet to answer any of these questions. Additionally, and other recent online initiatives like the DARPA-funded CS2N and TopCoder also have little research that assesses their effectiveness for middle and high school.

**Research Imperative 2:** Best practices and guidelines for leveraging structure online curricula in all-online or blended classroom settings.

As more resources and structured curricula are made widely (and freely) available online, research must provide guidance on how teachers should incorporate them in their classroom settings? How should face-to-face interaction between teachers & students, and among students be balanced in blended learning settings so that the social affordances of classroom settings can be leveraged for computational learning? What is the teacher’s role in such settings?

As an example, the author created and tested an online curriculum created on the Stanford OpenEdX platform to introduce middle school students to foundational ideas of
computational thinking. It comprises short Khan Academy-style video lectures that teach computing concepts through worked examples in Scratch. The videos were interspersed with quizzes, thought questions, short activities and programming projects in Scratch. The students did everything during classroom time including watching the videos as a quarter of the classroom consisted of students from disadvantaged neighborhoods with little to no access to computers at home. Even though the video-watching activity was done individually, several Scratch projects and thought questions involved students working in pairs as that was found to be more motivating and helpful for children in the 12-14 age group. The research using this configuration was largely successful, however it would be interesting to see how it would work with children doing more or less on their own.

Additional related lines of inquiry could also be pursued, such as- Can or should multiple online tools be used in primary and secondary classrooms? What could be a good sequence for use of these free tools and resources? Which are the best for early introduction? What should follow next and so on until students are considered “ready” for college computing coursework. Or can some tools and resources be used in conjunction with each other to complement the strengths and weaknesses and give learners a deeper understanding of computational learning through seeing the same constructs in different environments. These are interesting questions as they also leverage educational research that working with analogous representations of concepts and seeing contrasting cases is known to be beneficial for conceptual learning.

Lastly, it could be quite possible that some of these resources work better for certain grades and in conjunction with other tools rather than alone (regardless of what the tool/curriculum creators claim). For example, given the ease of block-based programming environments, one could introduce programming with Scratch or Tynker or Alice and then graduate to Khan Academy. What would a suitable entry point be in Khan Academy’s curriculum sequence in such a case?

**Research Imperative 3: Assessment of Computational Learning**

As mentioned in Grover & Pea, 2013, “Without attention to assessment, Computational Thinking (CT) can have little hope of making its way successfully into any K–12 curriculum. Furthermore, to judge the effectiveness of any curriculum incorporating CT, measures that would enable educators to assess what the child has learned need to be validated…What, for example, can we expect children to know or do better once they’ve been participating in a curriculum designed to develop CT and how can this be evaluated? These are perhaps among the most important questions that need answering before any serious attempt can be made to introduce curricula for CT development in schools at scale.”

The issue of assessment is particularly thorny, especially because of the variety of disparate programming tools available and in use. Most of the recent research on assessment of computational learning is tied very closely to specific programming environments (for example, Alice in the case of studies done by Denner, Warner, et.al., Agentsheets in Repenning et al.’s work, faulty e-textile projects using Lilypad Arduino
by Kafai, Fields, et al. and analytics-based diagnostics around programming challenges in Processing & Javascript in the case of Khan Academy (and similarly for Tynker). Can the CS Education research community identify and develop assessments measures involving “patterns” that transcend programming languages and environments? These patterns could then form the basis for different versions of assessments that are programming-language dependent. SRI’s work around assessments for ECS may form a starting point for this. Additionally, could there be language-agnostic assessments to measure algorithmic and computational understanding for students at various levels of K-12?

All these aforementioned threads of inquiry are necessary albeit challenging to pursue. However, in order to impact large-scale and successful rollout of curricula in K-12 classrooms that is guided by prior research in both computing education and the learning sciences, these are important to be answered and supported by empirical inquiry. Some are more tractable than others – studying the use of different curricula (Khan Academy, CodeHS, Tynker, Stanford OpenEdX course) with students randomly selected to different treatment groups and testing the outcomes on a common assessment, for example, may be more easily studied empirically than others to establish which curriculum works best for that set of, or similar, students. Or the same Khan Academy curriculum could be tested with students of different genders or SES backgrounds, or from very different contexts to compare how it works for those 2 disparate student groups. Clearly there is a prioritization exercise involved, but perhaps that could be part of the mandate for a select group that is convened to brainstorm and deliberate on these questions.

Citations