# How Broad is Computational Thinking? A Longitudinal Study of Practices Shaping Learning in Computer Science

Chris Proctor, Paulo Blikstein, Stanford University
Email: cproctor@stanford.edu, paulob@stanford.edu

**Abstract**: Computer science is becoming a mainstream school subject, yet we know relatively little about teaching, learning, and assessing computer science at the primary and secondary level. Few studies have followed the long-term trajectories of early computer science learners. We present a longitudinal study of a school cohort (N=48) across a three-year computer science curriculum in grades 6-8. We analyzed students' Scratch projects in terms of elaboration and computational thinking content, and modeled their association with performance on a summative open-ended assessment of computational thinking. Both metrics were associated with performance on the summative task, but engagement had a much more substantial effect. This supports the idea that early computer science experience should be designed to support students in working on personally-meaningful projects. Developing computational literacy practices may be more important for long-term growth in computational thinking than a primary emphasis on content knowledge.

## Introduction

Computer science is becoming a mainstream school subject in the United States. It is being incorporated as a course in the regular curriculum (Goode & Margolis, 2011), in bite-size chunks (Wilson, 2014), and through informal communities which partially overlap with schools. For example, Scratch 2.0 topped 350,000 monthly active users for several months in 2017. The interest in computer science is spurred by a recognition of the economic opportunities afforded by computing careers as well as our societal reliance on computational media for commerce, news, work, and everyday social life. While pioneers in educational technology have argued for computers in schools for decades, the last few years have seen several well-funded, broad-reaching initiatives.

The uptake has outpaced the research. We still know relatively little about teaching and learning computer science at the primary and secondary level (Guzdial, 2008; Grover & Pea, 2013; Blikstein, 2018). Two National Research Council reports (2010, 2011) sought to define the core practices of computer science using the term *computational thinking*, but surfaced substantial disagreement on how broad to make the definition and how it should be measured. Computational thinking certainly includes practices such as framing problems in terms of computational models and working with algorithms, but should it also include literacy practices enacted through computational media? These might include self-expression, collaboration, and developing a critical consciousness. The K-12 CS Standards (2016) adopt a compromise position by naming seven core computer science practices and designating four of them as computational thinking, "the heart of the computer science practices" (2016, p. 67). The definition of computational thinking is arbitrary but it will have real impacts on how computer science courses are assessed and taught. To find the most useful definition of computational thinking, it would be helpful to have longitudinal accounts of how students learn computer science over time. However, very little such research exists.

This paper presents a three-year longitudinal study exploring the extent to which middle-school students' programming in Scratch predicts later computational thinking skill. We developed two metrics to assess students' projects: *elaboration* measures how much students built out their projects, and *computational content* measures how much students used core computer science ideas such as control flow, events, and modularization. These metrics correspond to two different views on the nature of computational thinking: that it is broadly inclusive of literacy practices, or that it is a narrower collection of disciplinary skills.

## Background

In this study, we model long-term computer science learning as growth in computational thinking, a construct whose definition is under active debate. One goal of the study is to compare alternative definitions of computational thinking. One tradition following Papert (1980) sees computational thinking as the recruitment of computers for problem-solving, but also for learning more broadly, including the learner's relationship to knowledge and sociocultural factors. diSessa (2001) gives a three-part definition of computational literacy: knowing how to interact with computational media (the material), the cognitive abilities supported by

computational media (the cognitive), and the social arrangements enacted through computational media (the social). In this view, the work of a computer scientist is inseparable from how she positions herself socially, establishes her competence, and participates in communities engaged in computational thinking practices. Research on equity and inclusion in computer science (Margolis, 2003; Barron, 2004; Margolis et al, 2010; Kafai & Peppler, 2011) tends to take this broader, practice-based view of computational thinking. These practices include computational thinking-specific practices such as incremental and iterative development, testing and debugging, reusing and remixing, abstracting and modularizing (Brennan & Resnick, 2012), as well as related sociocultural practices such as collaborating, participating in an inclusive computational culture, and communicating about computing (K-12 CS Framework, 2016).

Other researchers, largely within the field of computer science, define computational thinking more narrowly, as the distinctive skills and knowledge gained through programming experience, and which comprise the disciplinary subject matter of computer science. In this view, computational thinking is primarily concerned with designing, using, and reasoning about computational models (Aho, 2011). Wing's argument that computational thinking "represents a universally applicable attitude and skill set" (2006) frames a narrow set of skills as widely applicable. In particular, the jobs-oriented argument for expanding access to computer science tends to emphasize skills valuable to employers and not practices which deepen self-knowledge, civic engagement, or critical awareness.

These epistemological stances lead to different ways of assessing computational thinking. Studies focused on engagement, belonging, and participation in computational thinking practices tend to seek evidence of learning by analyzing artifacts designed by students (Brennan & Resnick, 2012; Fields, et al, 2016), interviewing students (Barron et al, 2013), and by asking students to engage in scenarios where students design solutions (Brennan & Resnick, 2012) or fix solutions containing errors (Werner et al, 2012). Research adopting the narrower definition of computational thinking tends to use positivistic definitions of the skills and knowledge that make up computational thinking, and seek standardized assessments whose purpose is to measure learning apart from students' situated practices (Tew & Guizdal, 2011; Tew & Dorn, 2013). Denning (2017) argues for assessing students' *competencies*, which he defines as "ability accompanied by sensibilities." Research comparing the effectiveness of different teaching tools or approaches often relies on un-situated measures of learning to compare learning across contexts (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Weintrop, 2016).

In this study, our goal was to measure the association between students' practices over time and their computational thinking skills at the end of the three-year curriculum sequence. The summative assessment of computational thinking (described in more detail below) required students to solve a computational problem using any tools of their choice. Students were presented with multiple cases of the same problem with larger and larger datasets, so that they became intractable without implementing an algorithmic solution. The practices assessed were comfortably within all the definitions of computational thinking discussed above.

Each of the metrics by which students' Scratch projects are evaluated may be seen as grounded in one view of computational thinking. *Elaboration* measures how much detail students add to their projects, regardless of whether it has anything to do with computer science concepts. Students who create elaborate projects are likely imbuing them with personal significance and taking them up in their broader social practices. On the other hand, the narrower view of computational thinking would view much of projects' elaboration as irrelevant. C*omputational content* measures the density of blocks that map directly to core computer science concepts. For example, when a student's project contains a higher density of function definitions and function invocations, it is reasonable to assume she is exploring modularity. By comparing the association of each metric with students' later performance on the summative computational thinking task, this study explores which practices contribute to long-term growth in computational thinking.

## Methods

The participants in this study were a cohort of students at an independent all-girls' middle school in the western United States, where computer science is a core required class for all three years. The teaching philosophy of the school and the computer science classes is constructionist (Papert, 1981). During much of their school day, students work on personally meaningful projects (Papert, 1991) using tools ranging from one-to-one laptops to the school woodshop and metal-working shop. They are accustomed to seeking help from each other, from teachers, and from other resources at school and online. While some tests and quizzes are given, most summative assessments take the form of projects, presentations, portfolios, and reflections. Teachers give narrative evaluation rather than letter grades, and student self-assessment appears on report cards alongside teachers' evaluations. This school environment makes it more likely that students' projects embody authentic practices and that their performance on the summative task reflects their full capabilities.

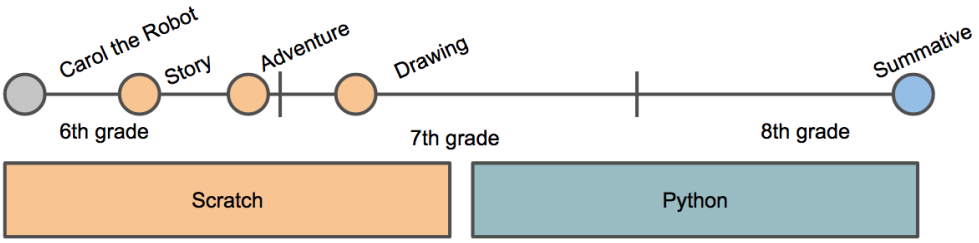


Figure 1. Middle-school computer science curriculum showing study measures.

We collected student work and written reflections from this cohort throughout their three years of computer science, and gave them a summative assessment at the end of 8th grade. Students worked primarily in Scratch in 6th and 7th grade. Each of the three projects analyzed from each student was the result of a curriculum unit spanning 4-6 weeks. At the end of each unit, students' projects were assessed on the use of computational thinking concepts such as control flow, events, using variables to process data, and decomposing problems with subroutines. Students also shared their projects in class and informally through their networks of followers on Scratch. Thus, in their projects, students had an incentive to attend to both subject-matter goals and to the enactment of literacy practices such as self-positioning, attending to audience, and taking up and interpreting socially-important narratives.

Students transitioned to working in Python in 7th and 8th grades (See Figure 1). Accompanying the change in programming interface was a change in emphasis from personal expression and narrative (and closer integration with their humanities classes) to modeling and conceptual exploration (and closer integration with mathematics and science classes). The summative assessment was distant from students' Scratch projects not just chronologically, but also in terms of the problem domains with which they were engaging. 48 of 67 students returned consent forms and were included in the study.

## Measures

### Scratch

We evaluated Scratch projects on two dimensions. *Elaboration* is defined the natural log of the total number of blocks in the project. *Computational content* is defined as the ratio of blocks from certain categories (data, events, control, sensing, and functions) to the total number of blocks (Brennan & Resnick, 2012). These two dimensions capture different kinds of interactions students had with their projects: sometimes students created detailed images or narratives, using many blocks in a straightforward (often sequential) manner. Other times, students were more focused on how their projects worked than on the end result, and tended to have fewer blocks but more concise and expressive code.

To analyze the projects, we wrote a Python package which fetches a representation of a project from the Scratch backend server and then maps each sprite, script, statement, and expression to a Python class instance, similar to the approach used by Fields et al (2016). Figure 2 shows elaboration and computational thinking distributions for each of the three Scratch projects analyzed. We calculated an *elaboration* and *computational content* score for each student by averaging the student's normalized score on the metric from each of the three projects. In the first two projects students worked from models and starter code provided by the

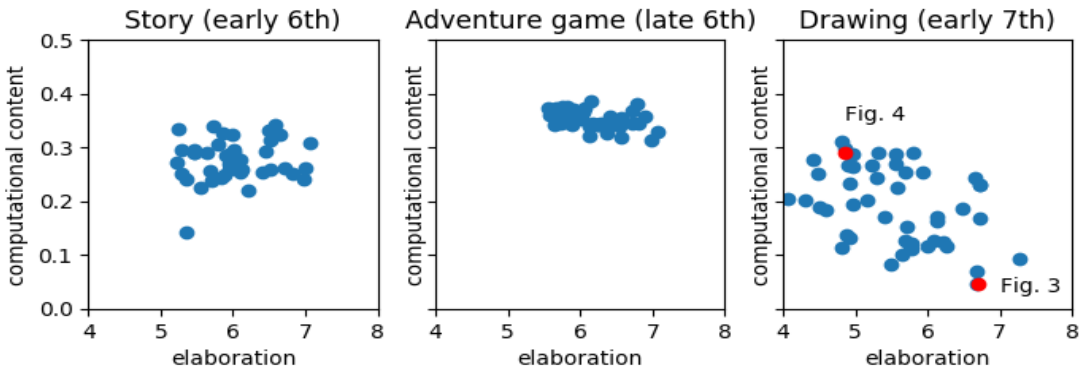


Figure 2. Elaboration and computational thinking content in three Scratch projects.

teacher. Using normalized values allowed us to exclude the starter code from analysis, and to weight students' work equally across the three projects. Each of these metrics represents a hypothesis about what might lead to long-term effects: the extent of block-based programming practice, or the richness of the practice in terms of computational content.

The final project, Drawing, provided students with the least scaffolding and therefore it is unsurprising that it has greater variance on both metrics than those preceding it. In this project, students were asked to create any drawing of their choice in Scratch, but the focus of the unit was on abstraction and modularization, and one of the evaluation criteria was students' use of functions and data to reuse code. The negative association between *elaboration* and *computational content* in this project corresponds with the intuition that code which makes effective use of data, control structures, and functions will be able to achieve a desired effect with fewer blocks. Projects with high computational content built up more complex drawing routines from reusable subcomponents, while some projects with low computational content also created elaborate effects, but they did so using hundreds of blocks effectively encoding point-to-point vector drawings. Figures 3 and 4 show two students' drawings and excerpts from project code.

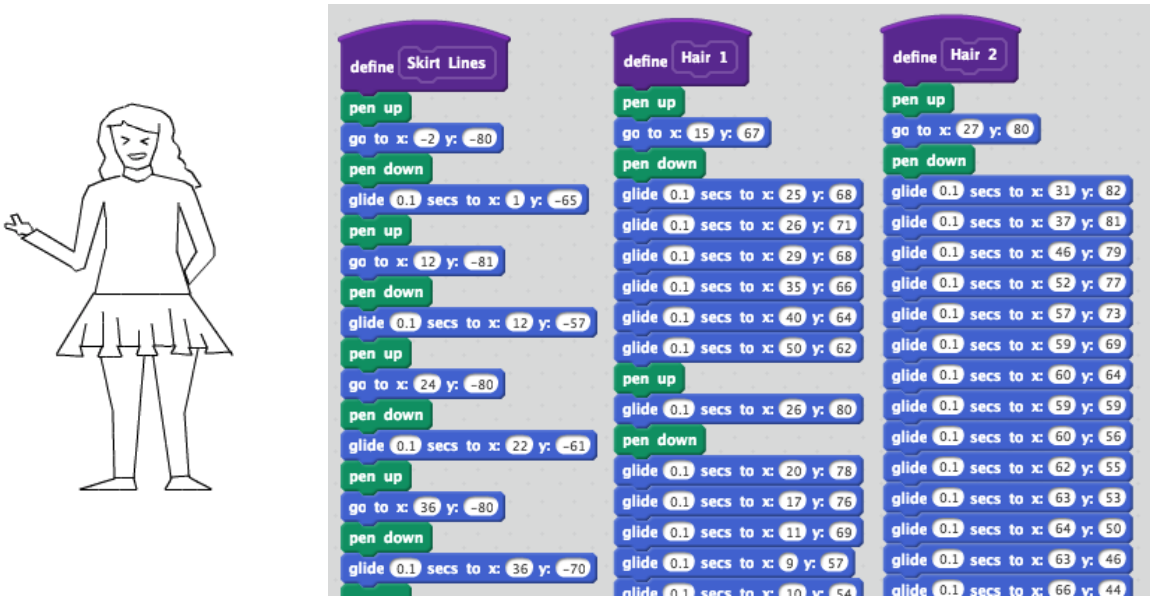


Figure 3. A student project with low *computational content* (z= -2.03) and high *elaboration* (z=1.49).

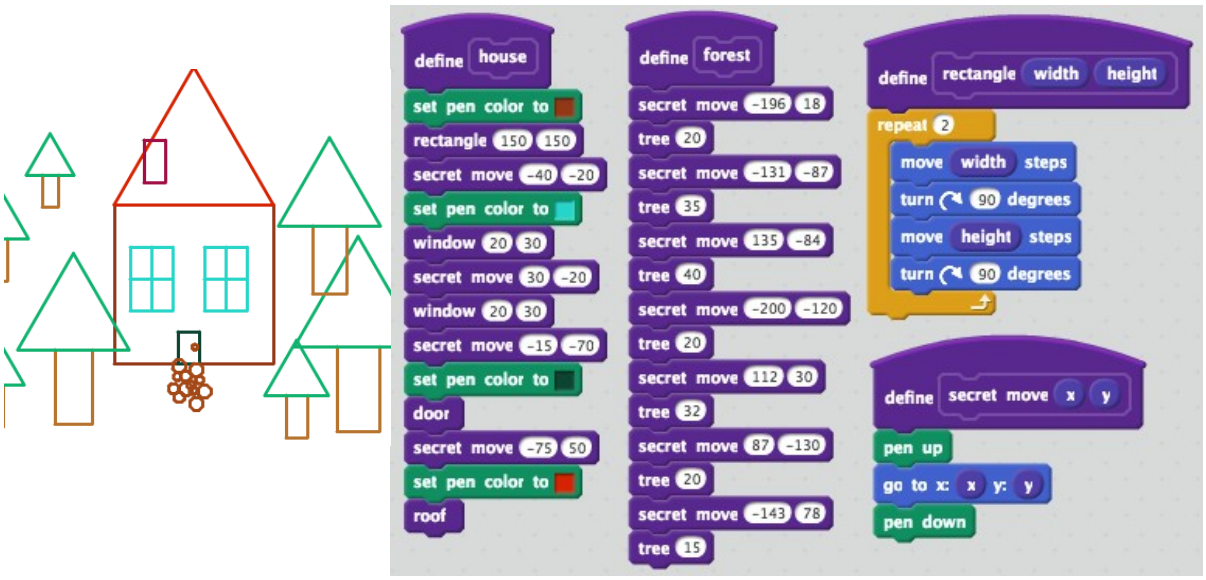


Figure 4. A student project with high *computational content* (z=1.40) and medium *elaboration* (z=0.34).

Figure 3 is characteristic of low *computational content* projects; the code excerpt shows superficial use of functions to break the program into subroutines, but the program itself is essentially a long sequence of imperative commands. At the same time, this is a self-portrait executed in code, in which the student depicts a detailed facial expression, gesture, hair, and clothing by using far more blocks than most projects. The student may have felt high social significance in each line's placement. Figure 4, meanwhile, depicts a nondescript scene very similar to the model drawn by the teacher in introducing the project. This student appears to have put her energy into the structure of her project rather than its final project; the code excerpt shows an elegant use of nested subroutines with arguments. Her project is almost entirely composed of function definitions and invocations; accordingly, her project's *computational content* was among the highest in the class.

### Summative computational task

At the end of the 8th grade year, we gave students an open-ended computational task over the course of two 45-minute class periods and analyzed the extent to which students used computational thinking to solve the task. The task presented students with a list of items for sale at a store, and asked students to spend a specified amount of money on exactly two items. There were six cases of the problem which only varied by the amount of money to spend and the length of the price list. The early cases could easily be completed by hand; the later cases had so many possibilities that they were intractable without the use of a computer. Then there were two final variations: a case in which three items should be purchased and a case in which two items must be purchased which added up to a certain price and to a certain weight. Students were provided each problem as a handout and were also given links to a starter Scratch and Python project initialized with the data. Students were encouraged to approach the problem as a fun puzzle, freely requesting help from their teachers and peers, and using any strategies of their choice. There was no grade, reward, or recognition attached to students' performance. At the end of each class session, students completed a survey in which they explained their work.

### Prior experience

Finally, we attempted to control for prior experiences by including two additional factors in some models. At the beginning of sixth grade, students began by completing a series of puzzles called *Carol the Robot*. In these puzzles, students wrote instructions to guide a robot through a maze to collect beacons without hitting walls. We used students' scores on this task as a measure of prior computational thinking skill. Additionally, at the end of sixth grade, students' teachers were asked to estimate their general quantitative skills.

## Analysis

We developed a rubric to evaluate students' use of computational thinking on the summative task (See Table 1). The rubric focuses on the core computational thinking practices of framing the problem, designing an algorithmic approach, and using a computer to implement it. In evaluating students on the rubric, we used their final submissions and their reflections, in which they explained and justified the approach they took to the problem. Scoring students solely on the number of cases students solved would not have been an adequate measure of computational thinking because some industrious students spent the entire 90 minutes doing tedious but occasionally lucky guess-and-check, while other students developed computational solutions which failed due to bugs. The least successful students used manual guess-and-check with no indication of a systematic approach; the most successful students developed correct, generalized solutions in Scratch or Python. Of the students who used programming, most used Python; only three chose to use Scratch (two of whom were successful). We were surprised at the diversity of students' approaches. Many used ad-hoc computational tools such as Excel to sort the numbers in a list or Word's search feature as part of an otherwise-manual strategy.

Table 1. Rubric used to evaluate students' use of computational thinking in the summative task.

| Level 0 | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 |
|---|---|---|---|---|---|
| Worked by hand or using a calculator. No evidence of a computational strategy. (Ex: guessed pairs of numbers over and over) | Worked by hand or using a calculator. Used a computational strategy. (Ex: decomposed the problem; systematically tested cases) | Used an ad-hoc tool (Ex: Word, Excel) to implement a computational strategy (Ex: decomposed the problem; sorting; searching) | Attempted to implement an algorithm using Scratch or Python, but did not solve all the two-item cases. | Successfully implemented an algorithm using Scratch or Python to solve all the two-item cases. | Successfully implemented a generalized algorithm solving a more complex case as well. |

Having defined metrics for students' Scratch projects and the summative measure, we used standard OLS linear regression to estimate the association between students' block-based programming practices and their performance on the summative task. Because our dependent variable is ordinal, we assume the rubric measures a latent, continuous random variable. This analysis treats the rubric categories as evenly-spaced intervals, an assumption we are comfortable making because the score distribution approximates a normal distribution (M=2.75; SD=1.47; skew=-0.22; kurtosis=-0.81). For each of the two dimensions on which we analyzed students' Scratch projects, we used the average normalized score across the three projects.

## Results

We found that both *elaboration* (r=0.512) and *computational content* (r=0.322) in block-based programming were positively correlated with higher performance on the summative assessment several years later. However, we found that project *elaboration* was a much more important metric (See Table 2). *Computational content* is also not a statistically significant predictor of summative scores when outliers are removed. The effects remained largely unchanged when controlling for prior computational thinking skill and teachers' estimates of quantitative skill. Figure 5 plots each metric against summative performance.

Table 2. Regression effect sizes (and p-values) predicting summative performance.

| *computational content* | *elaboration* | *prior ct (Carol the Robot)* | *prior quantitative* |
|---|---|---|---|
| 0.80 (0.035) | | | |
| | 1.21 (0.001) | | |
| 0.57 (0.094) | 1.10 (0.001) | | |
| 0.55 (0.11) | 1.14 (0.001) | -0.14 (0.427) | |
| 0.01 (0.074) | 1.01 (0.003) | -0.21 (0.236) | 0.49 (0.118) |

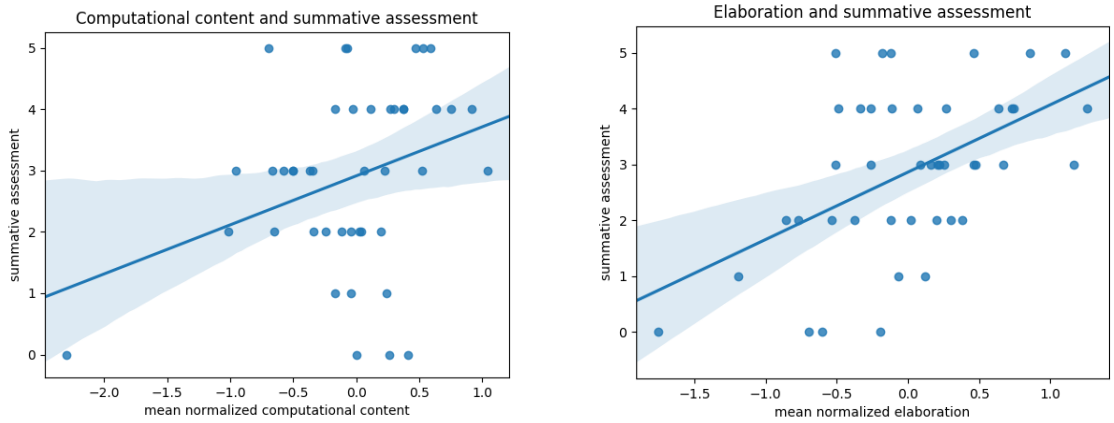


Figure 5. Scores for each metric plotted against summative scores with regression and 95% confidence interval.

## Discussion

We found that students who engage more deeply in creating Scratch projects, measured in either of two ways, are significantly more likely to perform well on a test of computational thinking skill several years later, even though all but three students chose to use tools other than Scratch in the summative assessment. Prior research has found little evidence of novices learning across programming interfaces (Armoni, Meerbaum-Salant, and Ben-Ari, 2015; Weintrop, 2016). There is also little evidence that programming experience develops generalized mental functions (Pea & Kurland, 1984), or for far transfer of thinking skills more generally (Barnett & Ceci, 2002). Our findings are an important result in an area where very little longitudinal research is available.

Comparing the two metrics of students' Scratch projects, we were surprised to find that students' project *elaboration* was much more associated with summative scores than *computational content*. This supports the idea that early computer science experience should be designed to support students in working on robust and sustained programming projects. Working on personally-meaningful projects may be more important than ensuring all students have a uniform foundation in basic computer science concepts. This is well-aligned with

sociocultural research showing the importance of supporting the development of student interest and identification with computer science, particularly for marginalized students. Students need to feel a sense of belonging to learn effectively, and computer science is pervasively stereotyped as being a subject most suitable for white, male, high-achieving students (Margolis 2003; Margolis et al, 2010) Fostering communities of creative media production can help to dismantle stereotypes and increase participation (Kafai & Peppler, 2011). The importance of these literacy practices is not reflected in several national initiatives to teach computer science. Future research on the reflections and self-assessments students submitted with their projects will allow us to corroborate our interpretation of the *elaboration* metric and to study students' process as well as product.

Our findings suggest that computational literacy practices are associated with long-term growth in computational thinking, even under the narrower definition of computational thinking. While the definition of computational thinking is arbitrary, our study suggests that students' sociocultural practices may play an important role in learning even core computer science content. Therefore, a broader definition, which recognizes computational thinking as a situated set of practices, might be the most useful. This is in line with Kafai & Burke's (2013) call to reframe computational thinking as computational participation, and diSessa's (2017) argument for extending computational thinking to computational literacy.

## Conclusions

This study offers one of the only longitudinal accounts of early computer science learning, and provides strong evidence that students' programming practices lead to long-term growth in computational thinking. The breadth of our central construct, computational thinking, is still taking shape. This study also offers an empirical exploration of the effects of operating under different definitions of computational thinking. In subsequent research, we intend to strengthen these claims by analyzing this cohort's reflective writing and self-assessment over the course of their middle-school experience learning computer science.

## Acknowledgements

Funding for this study was provided by the Lemann Center for Entrepreneurship and Educational Innovation in Brazil. We are grateful to the students and teachers who shared their time with us to make this study possible.

## References

Aho, A. V. (2011). Ubiquity symposium: Computation and computational thinking. Ubiquity, 2011(January), 1.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: blocks and beyond. Communications of the ACM, 60(6), 72-80.

Barnett, S. M., & Ceci, S. J. (2002). When and where do we apply what we learn?: A taxonomy for far transfer. Psychological bulletin, 128(4), 612.

Barron, B. (2004). Learning ecologies for technological fluency: Gender and experience differences. Journal of Educational Computing Research, 31(1), 1-36.

Barron, B., Wise, S., & Martin, C. K. (2013). Creating within and across life spaces: The role of a computer clubhouse in a child's learning ecology. In LOST Opportunities (pp. 99-118). Springer Netherlands.

Bell, T. C., Witten, I. H., Fellows, M. R., Adams, R., & McKenzie, J. (2015). CS Unplugged: An Enrichment and extension programme for primary-aged students.

Blikstein, P. (2018). Pre-College Computer Science Education: A Survey of the Field. Mountain View, CA: Google LLC.

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada (pp. 1-25).

Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript.

CSTA Standards Task Force. (2011). CSTA K-12 computer science Standards.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. Communications of the ACM, 60(6), 33-39.

diSessa, A. A. (2017, June). What If Your Project's Timeline is a 100 Years?: Reflections on Computational Literacies. In Proceedings of the 2017 Conference on Interaction Design and Children (pp. 1-2). ACM.

Fields, D. A., Quirke, L., Amely, J., & Maughan, J. (2016, February). Combining big data and thick data analyses for understanding youth learning trajectories in a summer coding camp. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 150-155). ACM.

Franklin, D., Conrad, P., Aldana, G., & Hough, S. (2011, March). Animal tlatoque: attracting middle school students to computing through culturally-relevant themes. In Proceedings of the 42nd ACM technical symposium on Computer science education (pp. 453-458). ACM.

Fraser, N. (2013). Blockly: A visual programming editor. *https://developers.google.com/blockly/*

Goode, J., & Margolis, J. (2011). Exploring computer science: A case study of school reform. *ACM Transactions on Computing Education (TOCE)*, 11(2), 12.

Grover, S., & Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. Educational Researcher, 42(1), 38–43. https://doi.org/10.3102/0013189X12463051

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

K–12 computer science Framework. (2016). Retrieved from http://www.k12cs.org.

Kafai, Y. B., & Burke, Q. (2013, March). The social turn in K-12 programming: moving from computational thinking to computational participation. In Proceeding of the 44th ACM technical symposium on computer science education (pp. 603-608). ACM.

Kafai, Y. B., & Peppler, K. A. (2011). Youth, technology, and DIY: Developing participatory competencies in creative media production. Review of research in education, 35(1), 89-119.

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

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch (Vol. 40, No. 1, pp. 367-371). ACM.

Margolis, J., & Fisher, A. (2003). Unlocking the clubhouse: Women in computing. MIT press.

Margolis, J., Estrella, R., Goode, J., Holme, J. J., & Nao, K. (2010). Stuck in the shallow end: Education, race, and computing. MIT Press.

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

National Research Council. (2011). Report of a workshop on the pedagogical aspects of computational thinking. National Academies Press.

National Research Council. (2013). Next generation science standards: For states, by states.

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

Papert, S. (1991). Situating Constructionism. *Constructionism*. I. Harel and S. Papert. Norwood.

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

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60-67.

Sivilotti, P. A., & Laugel, S. A. (2008, March). Scratching the surface of advanced topics in software engineering: a workshop module for middle school students. In ACM SIGCSE Bulletin (Vol. 40, No. 1, pp. 291-295). ACM.

Tew, A. E., & Guzdial, M. (2011, March). The FCS1: a language independent assessment of CS1 knowledge. In Proceedings of the 42nd ACM technical symposium on Computer science education (pp. 111-116). ACM.

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

Weintrop, D. (2016). Modality Matters: Understanding the Effects of Programming Language Representation in High School computer science Classrooms (Doctoral dissertation, Northwestern University).

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. (2015, June). 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 (pp. 199-208). ACM.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012, February). The fairy performance assessment: measuring computational thinking in middle school. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (pp. 215-220). ACM.

Wilensky, U., & Papert, S. (2010). Restructurations: Reformulations of knowledge disciplines through new representational forms. Constructionism.

Wilson, C. (2014). Hour of code: we can solve the diversity problem in computer science. *ACM Inroads*, 5(4), 22-22.

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