Identifying Elementary Students' Pre-Instructional Ability to Develop Algorithms and Step-By-Step Instructions

Hilary Dwyer‡, Charlotte Hill†, Stacey Carpenter‡, Danielle Harlow‡, and Diana Franklin†
† Department of Computer Science ‡ Gevirtz Graduate School of Education
UC Santa Barbara

ABSTRACT

The desire to expose more students to computer science has led to the development of a plethora of educational activities[16, 7, 15, 4] and outreach programs to broaden participation in computer science. Despite extensive resources (time and money), they have made little impact on the diversity of students pursuing computer science. To realize large gains, computational thinking must be integrated into K-12 systems, starting with elementary school. In order to do so, existing resources need to be adapted for a school setting.

In order to make a curriculum with lessons that build on each other over several years, and accountability for student learning, we need standards, an understanding of how students learn, and identification of what students know before exposure to the curriculum.

In this paper, we present our detailed findings of what fourth graders know before encountering a computational thinking curriculum. Groups of students participated in activities modified from CS Unplugged[4] in order to discover their knowledge (rather than provide instruction). We identify aspects of the activities students were able to complete successfully, and where they will need further instruction. We then explain how we used these results to modify our pilot curriculum.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education; K.4.m [**Computers and Society**]: Miscellaneous—*Diversity and Outreach*

General Terms

Design, Human Factors

Keywords

diversity, K-12 education, outreach, assessment

1. INTRODUCTION

For decades, computer scientists have been developing computer science activities for young students in an effort to engage them early. Responding to a national need for computer scientists, the National Science Foundation has funded the Broadening Participation in Computing program, and later the CE21 program, to increase the diversity, and thus absolute numbers, of students pursuing computer science.

This and other efforts have led to many educational platforms, activities [16, 7, 15, 4] and outreach programs [3, 2, 12]. Unfortunately, very little progress has been made in broadening computer science participation. In fact, the percentage of females has been dropping almost monotonically since the early 1980's[1], and underrepresented ethnic minorities continue to make up a very small percentage of students[20, 24].

Outreach programs reach a small percentage of the population, and they require enormous portions of facilitators' time. To diversify and increase the computer science population, as well as those equipped to innovate existing software, computer science must find a place in K-12 schools. That requires developing standards, curricula, and assessments.

In 2011, the CSTA released a set of K-12 standards [10] and scaffolded charts to illustrate intermediate points. While an important step, much work remains before computer science catches up to its STEM counterparts in developing research-based curricula. At a high level, two large gaps remain - empirical evidence to confirm and/or refine the information presented in the scaffolding charts, and lower anchor points to determine the proper starting points for the curriculum.

In this paper, we analyzed groups of fourth grade students participating in two activities modified from CS Unplugged - creating step-by-step instructions to draw a picture and sorting a small set of canisters. Our purpose was to identify what students were able to complete successfully without our help, and with what concepts students required assistance. We present these findings, as well as how we have used these to modify our initial curriculum.

The rest of the paper is organized as follows. We provide a background on theories of learning that guide our work in Section 2 with a brief summary of related work in Section 3. Sections 4 and 5 present our curriculum from a high level and the methodologies derived from educational research used in our study. We present our results - what we found in our focus groups and how that informed our learning progressions and curriculum - in Section 6. Finally, Section 7 contains our conclusions and future work.

2. BACKGROUND

Our work is informed by both constructivist theories of learning and constructionist perspectives on instruction. Constructivism is a theory of learning that explains learning as a process of making sense of new information through the lens of what one already knows [11]. This implies that curriculum developers must be cognizant of learners' prior knowledge and design experiences that allow them to build on and challenge their existing ideas. Further, it places the instructor in the role of guide and designer of educational experiences rather than someone who simply imparts knowledge.

Constructionism is a theory of learning and teaching proposed by Papert who, building on constructivist ideas, posited that people learn best "in a context where the learner is consciously engaged in constructing a public entity"[18]. That is, people learn by making something. Constructionism has been embraced by many educational scholars[14] who argue that a "design mentality" can support students' development of ideas.

A constructivist perspective on learning requires us to first identify the pre-instructional ideas that students develop about computational thinking. These ideas will inform how we develop instruction. Our ultimate goal is to construct a *learning progression* for computational thinking at the upper elementary school level. A learning progression [9] is an empirically-determined model of how students learn a particular set of ideas and skills. Learning progressions are bounded by lower anchor points which describe ideas that students are expected to have prior to instruction, and upper anchor points which describe the expected goals or standards students should meet by the end of instruction. Between the upper and lower points are multiple intermediate levels. Initial hypothetical learning progressions proposed by researchers serve as models for student learning and are repeatedly tested against evidence and revised.

3. RELATED WORK

The development and refinement of outreach programs using curricula based on CS Unplugged, Scratch, and Alice have matured in their goals and assessments through time. These assessments have had various time / information-yielded trade-offs. Early assessments used surveys to gauge student attitudes, and later assessments inspected final projects or tests to determine what students knew at the end. These have been very helpful in giving a glimpse of what students are capable of at different ages. The next step is to create a picture of how students learn in order to inform future curricula. We want to be able to design the curriculum, demonstrate that it worked and explain how and why it worked.

Researchers have mined the wealth of completed Scratch projects to determine what concepts were displayed in completed projects. Over 500 Scratch projects created by urban youth during an afterschool program showed that youth learned key programming concepts without specific instructional interventions or experienced mentors[17]. Wilson et al[22] adapted a prior coding scheme[8] to find the most common programming concepts used by children who created games with Scratch.

Another effort [19] used existing projects across different grade levels to identify how computational thinking concepts varied by level in their Progression of Early Computational Thinking (PECT) Model. Finally, Franklin et al[13] used a combination of field notes, hand analysis and automated analysis[5] of Scratch projects to determine which concepts students learned initially and were able to transfer to a culminating project from a 2-week summer camp for middle school students.

There have also been attempts to seek deeper information through more detailed assessments, but they still focus on what students understand at the conclusion of instruction. As part of the effort to understand variation in computational thinking across students, Werner et al[21] developed a performance assessment tool to assess algorithmic thinking and effective use of abstraction and modeling among middle school students. Finally, Brennan et al [6] used portfolio analysis, artifact-based interviews, and design scenarios to assess children's computational thinking based on a framework comprised of computational concepts, practices, and perspectives.

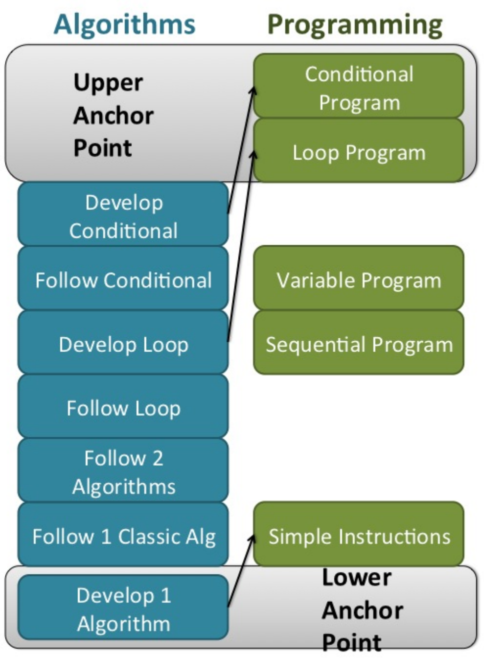


Figure 1: The two strands of our hypothetical learning progression that are the subject of this study. We focus on the lower anchor points.

Our work is distinct because it focuses on what students know before they encounter a formal computational thinking curriculum. This is just a first step towards the larger goal of understanding how students learn computational thinking. Once a more detailed model is developed for students progressing through the curricula, we will better target and provide appropriate instruction.

4. CURRICULUM

Research on how children learn often depends on the curriculum that supports that learning. This is true of our work on learning progressions. We began with a pilot curriculum which we will iteratively revise as we adjust our hypothetical learning progression. Figure 1 shows our hypothetical learning progressions for two aspects of computational thinking: algorithms and programming. Because this study focuses on identifying lower anchor points, we focus on those elements in Figure 2.

Our curriculum helps students develop ideas about computational thinking with two types of activities: activities done off the computer and programming activities done on the computer. The first, which we call FiredUp, are loosely modeled after CS Unplugged activities[4]. During these activities, students encounter computational thinking concepts in the context of their everyday lives. FiredUp activities are followed by WiredUp activities - projects programmed in the Scratch programming language[23]. These relate to content included in 3rd or 4th grade, such as planets, geography, and erosion both to increase the chances that the curriculum will be adopted by elementary schools, and to show students how computational thinking can be integrated into multiple subjects. We also include bonus projects for students who finish early and would like to explore more creatively.

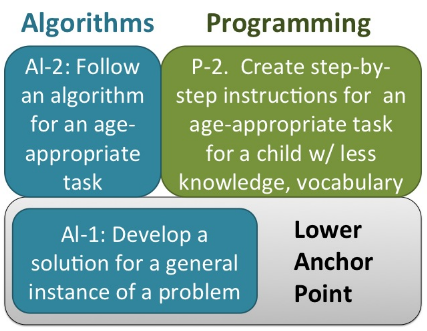

Algorithms
Programming
Al-2: Follow an algorithm for an age-appropriate task
P-2. Create step-by-step instructions for an age-appropriate task for a child w/ less knowledge, vocabulary
Al-1: Develop a solution for a general instance of a problem
Lower Anchor Point

Figure 2: The expanded description for the items within and directly above the lower anchor points.

In this paper, we describe activities conducted in focus interviews that were related to FiredUp activities. We focused on fourth graders who had not been formally taught computational thinking or computer science. Thus, our results informed our *lower anchor points*. From our interviews, we identified what students were already able to do as well as what they were not able to do. This helped to better define our lower anchor points and inform our curriculum development.

5. METHODOLOGY

There are several steps to the iterative process of testing and modifying hypothetical learning progressions. We first present the methodology that we plan to use for the entire process. We then describe the methodology used for the particular findings we present in this paper.

We began by studying the CSTA K-12 Computational Thinking Standards, attending closely to the K-6 standards. This was done in two parts - first, reviewing the standards (an end-point or *upper anchor point*) and second, examining the scaffolding charts (not dissimilar to learning progressions, but as yet untested). We chose a subset of standards to target and designed a detailed learning progression for those subjects. Figure 1 shows the hypothetical learning progression relevant to the subjects we address in this paper.

Educational researchers use a combination of methods to discover details on how students learn. These methods include focus group discussions, observations of individual and group activities, talk-alouds in which students describe their thought process as they work, and analysis of student projects. Typically, pre- and post- assessments are used to evaluate the curriculum itself, whereas more detailed and frequent methods are used to discover how students learn - the paths students take in their journey from the lower anchor point to the upper anchor point.

In this study, we are identifying *lower anchor points*, or the starting point, for fourth graders before instruction. We conducted fifteen focus group interviews with fourth grade students at four different elementary schools in Southern California during May 2013. A total of 55 fourth graders participated in these interviews. The schools were selected to represent a broad range of demographics. The percentage of English Language Learners (ELL) ranged from 22% at Vista Grande[1] to 81% at Cabrillo, and students receiving free or reduced lunch ranged from 29% at Vista Grande to 100% at Cabrillo.

Each focus group interview lasted approximately 30 minutes and was moderated and filmed by two graduate students. The interviews focused on three themes (knowledge about computers, complex decisions, and sequential procedures), and ended with one of four activities that were designed to elicit ideas related to aspects of computational thinking.

The discussions from the activities were transcribed and analyzed for discourse that supported or challenged our proposed model. Text was coded by its connection to the hypothesized learning progressions in our model of computational thinking. All results were independently coded by a computer science graduate student and education graduate student.

[1]All school and student names changed for confidentiality

6. RESULTS

In our initial focus groups, we identified some aspects of what students were able to do and what they were not able to do related to two strands of our learning progression: step-by-step instructions and algorithmic development. These were used to refine our initial activities.

For each of the topic areas, we first present the activity, then what we discovered from the students, and finally how we used that information to make changes to our initial curriculum.

6.1 Step-by-Step Instructions

Giving step-by-step instructions is a base-line skill that contributes to two areas of computational thinking: algorithms and programming. The activity we used to elicit this was based on the CS Unplugged activity Marching Orders[4], in which students are given a picture and asked to give instructions to another student, who draws the picture based on the directions. The purpose of the activity is to show students the importance of providing precise instructions to others, and the challenging nature of such a task when one has been restricted to only written instructions. We adapted this exercise to discover how well students could a) initially create directions, b) analyze the strengths and weaknesses of a set of instructions and c) improve upon the original attempt.

We adapted the activity in the following ways. First, we did the activity in three rounds, each progressively more difficult than the former. In the first round, the picture was simple: a circle, square, and triangle vertically arranged and touching. In the second, the picture was a square with lines bisecting different sections. In the final round, the picture included multiple shapes and lines, haphazardly organized. We also adapted the activity so that students participated in different ways during each round. In the first two rounds, one student gave directions to the other students. In the third round, students worked together and created directions for the interviewer, who would draw a picture based on their description. Six focus groups (17 students) participated in the drawing activity. As this was a semi-structured interview, if a student wanted to try giving alternate directions for the same picture, we allowed it and considered these included in the same round.

Findings Related to Step-by-step Instructions.

Four distinct ideas related to step-by-step instructions emerged from our data. The first two describe things that the students were able to do prior to any instruction and the third is related to what they could not do. Below we list all four findings for this topic and then discuss the data that led to these findings.

- Finding 1a: Fourth graders recognized the need for specific instructions.
- Finding 1b: Fourth graders were able to identify what was lacking in instructions.

- Finding 1c: Fourth graders described more attributes following critique and discussion.
- Finding 1d: Fourth graders struggled to employ precision and cover all attributes consistently.

From our data, students' initial attempts lacked many attributes of specific instructions. In particular, during the drawing activity, many students identified included shapes to be drawn and explained roughly their position; however, they did not consistently include sufficient details (such as absolute position, size, or orientation) needed so that other students (or the interviewer) could draw the object.

The subsequent discussion illustrated that students seemed to recognize the need for specific instructions, identified when others' instructions were vague and provided suggestions about how to amend the instructions. Our first two findings emerged when the fourth graders were asked to compare the drawings they created when following someone else's directions to the original drawing. They recognized that the drawings did not match and identified aspects of the directions that should have been clarified. For example, one student provided the instructions, "There's a circle on the top." followed by "There's a triangle - there's a square in the middle. There's a triangle at the bottom." Other students recognized that knowing whether the shapes were organized in such a way that the shapes were "touching" was a key piece of information that would have improved their drawings.

This left a gap between what students can identify about others' instructions and what students produce themselves. We then attempted to narrow down this gap by allowing the students to produce instructions for a final drawing, with the only new information being their own discussion about what should have been clarified about the first attempt. We found that they still struggled to produce specific instructions themselves even after discussing how to improve their peers' instructions. After the discussion, students improved their instructions in one regard - they described additional attributes (Finding 1c). Table 1 shows how many groups mentioned an attribute at least once during a round (this was per round, not per shape). We see that during initial rounds, several groups failed to mention certain attributes such as orientation and absolute position. After the discussion, almost every group remembered to mention all attributes except for orientation.

What the table does not express is how consistently students remembered to mention every attribute (once per round vs for every shape) or to describe precisely each attribute (the line is short vs the width of my thumb). We found that students struggled on both accounts. So while there was improvement, their final instructions were far from complete (Finding 1d).

Implications for Curriculum.

Since students demonstrated difficulty improving their instructions after analyzing their and others' first attempt, we greatly expanded the "discussion" portion of the activity. We now have a complete lesson on procedural writing, including the importance of sequencing, the concepts that need to be expressed, and vocabulary words one can use.

6.2 Algorithm Development

Our second set of findings relates to algorithm development. We consider an algorithm to be broader than what can be solved by a computer; it refers to the ability to develop a sequence of steps or other problem-solving operations. As such, we were interested in identifying how well students could develop a sequence of steps to most efficiently solve a problem. To elicit this, we used an activity that was based on the CS Unplugged activity Lightest and Heaviest[4], in which students were given eight canisters and a scale. They were asked to perform a series of order-based tasks, starting with finding the lightest and ending with sorting all eight canisters. The initial tasks of the original CS Unplugged activity were intended to build the knowledge students will need to solve the sorting problem efficiently. We adapted this activity to discover how well students could a) create an algorithm for a small problem without initial instruction, b) extrapolate how their algorithm would work on a larger number of canisters, and c) analyze how their algorithm could be sped up. Below we describe how we adapted the activity for our purposes and the findings we inferred from the data.

In our modified version of this activity, we asked fourth graders to sort 6 film canisters from lightest to heaviest (each film canister contained different weights). The students worked as a group, and could only lift two canisters off the table at a time. Students were not provided a scale - instead, they compared the weights with their hands.

When they finished, the interviewer led a discussion about how to make the sorting faster, especially if they had more canisters to sort. Four focus groups (20 students) participated in this activity.

Findings Related to Algorithm Development.

From the focus group interviews, we identified five ideas that helped refine our learning progression for algorithm development. These findings describe what students can do as well as limitations in what they cannot do. Below we list the five findings. Following this, we discuss each one in more detail.

- Finding 2a. Fourth graders developed an algorithm that would successfully sort 6 canisters.
- Finding 2b. Fourth graders suggested many ways for speeding up the implementation of their algorithm.
- Finding 2c. Some fourth graders identified the limitations of their algorithms when scaled to a larger number of items.
- Finding 2d. Some fourth graders recognized that breaking a large problem into parts made solving it easier.
- Finding 2e. Fourth graders often bypassed necessary comparisons because they remembered canister weights from earlier trials.

In our first finding, fourth graders developed an algorithm to solve the task. Across several schools, all focus groups developed the same general algorithm. They first paired the canisters and performed an independent comparison of each pair of canisters, splitting them into two groups of three canisters. They then performed sorts within each group of three to sort them locally. They had two lines of three canisters, each sorted from left to right, lightest to heaviest. In order to merge the two groups, they placed the "light" line on the left and the "heavy" line on the right. They then performed local bubbling left and right to adjust the middle canisters.

While all students correctly sorted their set of 6 canisters, the initial algorithm chosen was not scalable (Finding 2c). Moreover, the merges were inefficient. Students assumed that placing the groups next to each other was close to the sorted order, because they had performed the initial splits into groups of heaviest and lightest. They then "cleaned up" the ordering with some final check passes. When scaling the problem, they would need to resolve this merge issue because the number of steps to fix the merge would increase substantially.

We inferred our other findings in this area from discussions following the initial sort about how to scale their algorithm. When we

Attribute	Example	Round 1	Round 2	Discussion	Round 3
Shape	circle, square	6	5	4	6
Orientation	facing upwards, horizontal, up and down	1	3	2	2
Relative Position	Touching, on top of	5	4	4	5
Absolute Position	middle of paper	3	3	0	5
Size	units, big/small	1	5	1	5

Table 1: The attributes necessary to give precise drawing directions, and the number of groups that used that attribute at least once in their directions. The groups improved after the discussion.

Type	Name	Description	Groups
Algorithmic	Split	Split into smaller piles to sort	2
	Split and Merge	Both split and proper merge suggested	1
	Insert Sort	Compare next one to existing sorted line of them until you find place	1
	Organization	Put them into a line	1
Implementation	Shake	Shake canisters by listening and feeling	2
	Scale	A scale would be faster than using their hands	2
	2+	If you can touch more than two, weighing is faster	1
	Fast Pick-up	Pick them up really fast	1
	Practice	It will be faster second time around due to familiarity	1
	Less Repetition	Only one person compares, not all	1

Table 2: Student suggestion, the type (Algorithmic or Implementation) and how many groups had this suggestion

asked students to suggest improvements if they were given many canisters, we did not specify what kind of improvements. Students provided two kinds of suggestions - implementation and algorithmic.

When asked how they could speed up their sorting, students easily provided suggestions to improve the implementation of the algorithm (Finding 2b). Table 2 shows the suggestions. We define an implementation suggestion as one that would involve speeding up the execution of the algorithm, but without changes to the order or number of comparisons. Implementation suggestions included picking up the canisters very quickly, using a scale, shaking canisters, and practicing the process. Students were much less likely to think of algorithmic suggestions.

Related to this is finding 2c. Two groups figured out that their original algorithm would not scale well. One group observed that it was wasteful to compare two new canisters each time, rather than a new canister and a previously sorted canister. They adjusted their algorithm to be a form of insertion sort. The second group noted the inefficiency in making two piles initially, and thus placed them in a line, also performing an insertion sort.

Also, students suggested ways to simplify a larger problem. Two different groups suggested that if they received more canisters, they would break them up into more than 2 groups (Finding 2d). One group even suggested a better merge algorithm - taking the lightest ones from each group and comparing those.

Our final finding is not directly related to items in our existing learning progression for algorithm development but is important nonetheless. In finding 2e, children remembered weights of the original canisters and thus bypassed steps that a computer would be required to complete to solve the algorithm. In this exercise, it is useful for students to sort the canisters multiple times. We found that, while they still "weighed" the canisters, they were not "comparing" them directly the second time. Instead, they remembered approximate weights. While this is a valid algorithm if you know you will get the same weights each time, and is somewhat analogous to having some hash table with the weight mapped to the final location, this is not what we were hoping for.

Implications for Curriculum.

Our findings have led us to modify this activity for our curriculum in three ways, and we will consider more major changes depending on whether the first two are sufficient during field testing.

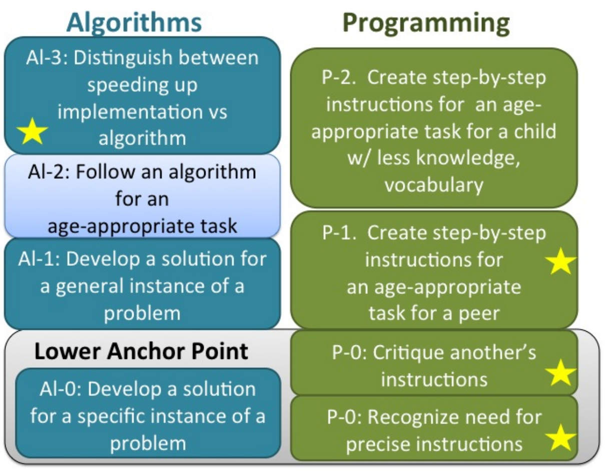
First, we need to explicitly distinguish between the speed of the algorithm and the speed of implementing the algorithm. Like the original CS Unplugged activity, we will focus on counting the number of comparisons.

Second, developing and describing algorithms takes more time and depth, and this is perhaps why students would either only describe a way to speed up execution, would develop incomplete algorithms, or would start describing an algorithm but then get sidetracked by more execution speed suggestions. We need to give the students more time to develop algorithms and encourage they test new algorithms as they arise. In addition, we need to give the students new canisters each time with different weights so they cannot use their prior knowledge. Finally, when asking about solving with more canisters, we should be ready with a large number of canisters so the exercise can be less theoretical.

Third, we need to provide students with techniques for analyzing algorithms, including examples and vocabulary. This is similar to the expanded lesson we will provide for the Marching Orders activity to teach students how to give precise instructions.

6.3 Changes to Learning Progression

Figure 3 reflects the changes, additions, and confirmations to the lower items in the learning progression, including the lower anchor points. For Algorithms, we split “Develop a solution for a general instance of a problem” into two parts - develop a solution for a specific instance of a problem and develop a solution for a general instance of the problem (i.e. N canisters rather than 6). In addition, we identified an important concept - distinguishing between efficiency of the implementation and the algorithm. In Program-



Algorithms
Programming
Al-3: Distinguish between speeding up implementation vs algorithm
Al-2: Follow an algorithm for an age-appropriate task
Al-1: Develop a solution for a general instance of a problem
Lower Anchor Point
Al-0: Develop a solution for a specific instance of a problem
P-2. Create step-by-step instructions for an age-appropriate task for a child w/ less knowledge, vocabulary
P-1. Create step-by-step instructions for an age-appropriate task for a peer
P-0: Critique another's instructions
P-0: Recognize need for precise instructions

Figure 3: Final learning progression. Al-1 was split into Al-0 and Al-1. New items are starred. Al-2 was not tested in this study, so its position is still hypothetical.

ming, we identified several precursors to creating step-by-step instructions for someone with limited knowledge and/or vocabulary. Students were not yet able to create precise step-by-step instructions for a peer, but they did recognize the need for precise instructions and were able to critique another's instructions.

7. CONCLUSIONS AND FUTURE WORK

We have shown initial results from in-depth focus groups. These focus groups informed three lower anchor points for the learning progressions of two CSTA K-12 strands and how that knowledge informed our pilot curriculum. We found that students could analyze existing directions for deficiencies, and their analysis led to incrementally better directions, but they were unable to produce thorough, precise instructions themselves. Finally, students were able to solve a small instance of a problem, but they had difficulty describing a working algorithm when the problem was scaled. In addition, they often focused on accelerating the mechanics of the operation rather than the algorithm itself.

In the future, we will study multiple classrooms and their progress through our pilot curriculum. Our goal is to produce more complete learning progressions and refine our curriculum with the information we gather through interviews, videos, and assessments. Our end goal is a three-year curriculum for computational thinking.

8. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation BPC Award CNS-0940491 and and CE21 Award CNS-1240985.

9. REFERENCES

[1] Percent female bachelor's degrees.

[2] S. Alliance. The stars alliance: A southeastern partnership for diverse participation in computing. NSF STARS Alliance Proposal. http://www.itstars.org/.

[3] I. Arroyo et al. Effects of web-based tutoring software on students' math achievement. In *AERA*, 2004.

[4] T. Bell, I. H. Witten, and M. Fellows. *Computer Science Unplugged*. 2006.

[5] B. Boe, C. Hill, M. Len, G. Dreschler, P. Conrad, and D. Franklin. Hairball: Lint-inspired static analysis of scratch projects. In *SIGCSE '13*, March 2013.

[6] K. Brennan and M. Resnick. New frameworks for studying and assessing the development of computational thinking. In *AERA*, 2012.

[7] W. Dann, S. Cooper, and R. Pausch. Making the connection: programming with animated small world. *ITiCSE*, 2000.

[8] J. Denner, L. Werner, and E. Ortiz. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers and Education*, 58:240–249, January 2011.

[9] R. A. Duschl, H. A. Schweingruber, and A. W. Shouse. *Taking science to school: Learning and teaching science in grades K-8*. 2007.

[10] C. S. T. Force. *CSTA K-12 Computer Science Standards*. Association for Computing Machinery, 2011.

[11] C. Fosnot. *Constructivism: A psychological theory of learning*. Teachers College Press, 1997.

[12] D. Franklin, P. Conrad, G. Aldana, and S. Hough. Animal tlatoque: attracting middle school students to computing through culturally-relevant themes. In *SIGCSE '11*, 2011.

[13] D. Franklin, P. Conrad, B. Boe, K. Nilsen, and C. H. et al. Assessment of computer science learning in a scratch-based outreach program. In *SIGCSE '13*.

[14] J. Gee. What video games have to teach us about learning and literacy. *Computers in Entertainment (CIE)*, 1(1):20–20, 2003.

[15] C. S. Hood and D. J. Hood. Teaching programming and language concepts using legos. In *ITiCSE*, June 2005.

[16] J. Maloney et al. The scratch programming language and environment. *Trans. Comput. Educ.*, 10(4):16:1–16:15, Nov. 2010.

[17] J. Maloney, K. Peppler, Y. B. Kafai, M. Resnick, and N. Rusk. Programming by choice: Urban youth learning programming with scratch. In *SIGCSE '08*. ACM Press, 2008.

[18] S. Papert. *Constructionism: research reports and essays 1985-1990*. Ablex Publishing Corporation, 1991.

[19] Seiter and Forman. Modeling the learning progressions of computational thinking of primary grade students. In *International Computing Education Research (ICER)*. ACM, 2013.

[20] J. Vegso. Cra taulbee trends: Ph.d. programs and ethnicity. *Computing Research News*, 2007.

[21] L. Werner, J. Denner, S. Campe, and D. C. Kawamoto. The fairy performance assessment: Measuring computational thinking in middle school. In *SIGCSE12*. ACMPress, 2012.

[22] A. Wilson, T. Hainey, and T. M. Connolly. Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *6th European Conference on Games-based Learning (ECGBL)*, 2012.

[23] G. C. Y B Kafai, K A Peppler. High tech programmers in low income communities: Creating a computer culture in a community technology center. *Proceedings of the Third International Conference on Communities and Technology*, pages 545–562, 2007.

[24] S. Zweben. Computing degree and enrollment trends. *Computing Research News*, 2011.