Title

Mapping the development of students' ideas in order to understand learning in a collaborative programming environment

Journal

Computer Science Education, 24(2-3)

Authors

Harlow, Danielle Boyd
Leak, Anne E

Publication Date

2014

Peer reviewed

Mapping Students' Ideas to Understand Learning in a Collaborative Programming Environment

Danielle Boyd Harlow and Anne Emerson Leak

Department of Education, Gevirtz School of Education, University of California – Santa Barbara, Santa Barbara, CA, USA

Contact: dharlow@education.ucsb.edu, (805) 893-8139

Mapping the Development of Students' Ideas in order to Understand Learning in a Collaborative Programming Environment

Recent studies in learning programming have largely focused on high school and college students; less is known about how young children learn to program. From video data of 20 students using a graphical programming interface, we identified ideas that were shared and evolved through an elementary school classroom. In mapping these ideas and their resulting changes in programs and outputs, we were able to identify the contextual features which contributed to how ideas moved through the classroom as students learned. We suggest this process of idea mapping in visual programming environments as a viable method for understanding collaborative, constructivist learning as well as a context under which experiences can be developed to improve student learning.

Keywords: programming; computational thinking; elementary students; constructionism; collaborative learning; memes

Introduction

Children in primary schools today will need not only the skills to use technology, but also the ability to interact with it in ways that allow them to innovate on existing technology and create their own technology. Two important developments in educational technology and software have impacted how computer science topics can be addressed at the elementary school level. For one, the increasing availability of portable classroom technology (e.g., mobile devices, tablets, and laptops) has added flexibility to the ways that technology can be integrated into the classroom (Dunleavy & Heinecke, 2007; Topper & Lancaster, 2013). Teaching computer science to elementary school students is no longer limited to teaching skills individually or to an entire class in a fixed computer lab environment. Instead, teachers can now use the mobility of students and technology to foster collaborative learning. The second important innovation is the development of graphical programming interfaces such as Scratch (Resnick et al., 2009) and the related program TurtleArt (Bontá, Papert, & Silverman, 2010). These types of interfaces have lowered cognitive barriers to computer programming to a level that allows even young children access. Children now have structured opportunities to learn basic programming skills and create functional outputs in the form of drawings and simple games.

A recent New York Times article (Richel, 2014) reported that 20,000 K-12 teachers have introduced coding in their classrooms in the past five months and 30 school districts planned to add coding in the fall, evidence that programming has gained an increased presence in K-12 schools. Yet, despite the increased use of computers in elementary classrooms and calls for young children to learn computer science and programming, we know very little about how children develop these skills in classroom environments. In order to facilitate students' learning, it is important to

understand how their ideas develop and what instructional contexts are key to this development. Our study has the primary goal of understanding the interactional contexts that facilitated idea development. To do so required developing a means of tracking ideas during innovative collective idea development that makes learning in collaborative learning contexts visible. Thus, we developed a method of representing the trajectory of ideas as they were communicated through technology and student talk and replicated or changed by students. In the following section, we describe constructionism, computational thinking and graphical programming environments.

Background

A Theory of Learning: Constuctionism

In the classroom studied here, children used a graphical programming interface to produce drawings through instruction that was consistent with constructionist ideals. Constructionism is a theory of learning and teaching that was first articulated by Papert (1980). He posited that people learn "especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe" (Papert, 1991, p.1). Constructionism is embraced by many educational scholars (e.g. Gee, 2003; Wiggins & McTighe. 2005; Hayes & Games, 2008) who argue that a "design mentality" can support students' development of ideas. The design cycle of trying out and testing ideas "can be seen as a type of play: children play out their ideas with each new creation. In design activities, as in play, children test boundaries, experiment with ideas, and explore what's possible" (Resnick, 2006, p. 196). When programing with graphical interfaces, children try out ideas and see what happens. Children also use creative appropriation (remixing), building off of others' programs to create something new (Monroy-Hernández &

Resnick, 2008). In graphical programming environments, the output and the process (the program) are often displayed side-by-side, making both the program and output immediately apparent to the student who created the program and any onlookers, making the ideas available for adaption and alteration.

Programming and Computational Thinking

Successfully developing ideas that are novel and useful requires domain specific knowledge and the ability to apply that knowledge flexibly (e.g., Hatano and Inagaki, 1986; Sternberg, 1999). The domain specific knowledge we are concerned with here is Computational Thinking, "an approach to solving problems that can be implemented with a computer. … [Computational thinking includes] abstraction, recursion, and iteration, to process and analyze data, and to create real and virtual artifacts" (Computer Science Task Force, 2011, p. 10). Computational thinking, however, is not limited to work on a computer. It is a problem solving practice useful across disciplines. The Computer Science Teachers Association (CSTA) recently proposed standards for what students should learn at the elementary and secondary levels (Computer Science Task Force, 2011), bringing attention to the importance of considering what young learners can and should learn about computational thinking.

While standards for what children should learn about computational thinking are new, attempts to teach programming and computational thinking to children has been going on since the 1980's. In 1980 Papert wrote, "In my vision, *the child programs the computer* and, in doing so, both acquires a sense of mastery over a piece of the most modern and powerful technology and establishes an intimate contact with some of the deepest ideas from science, from mathematics, and from the art of intellectual model building." (Papert, 1980, p. 5, emphasis in original).

With Logo, one of the first programming languages developed for children, users controlled a turtle on the screen with simple typed commands such as **FORWARD 100** and **RIGHT 90.** Children could use these simple commands to draw geometric patterns on the screen. More recent interfaces remove the requirement of typing in text-based commands by using graphical interfaces consisting of commands in blocks or puzzle pieces that snap together to create code. In TurtleArt, the interface used here, a graphical turtle creates drawings according to scripts that the user creates using snap-together visual commands. Figure 1 shows a bank of commands (far left). The user drags the commands onto the programming area (right) and snaps them together to create a script. When the script is clicked, the turtle draws an image.

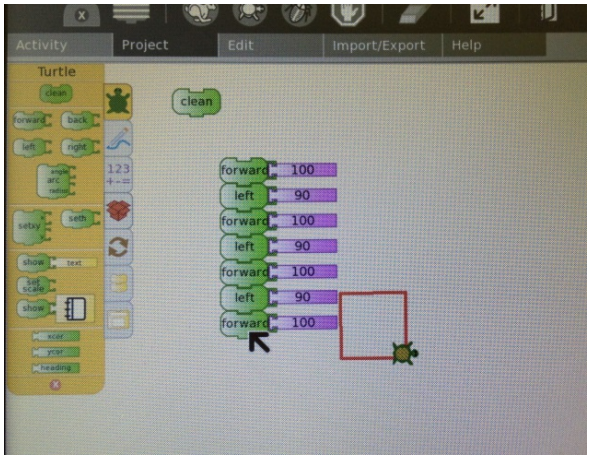

Figure 1. Screen shot of a program to create a square (puzzle blocks) and output (square).

The user can select commands that do things like move the pen up and down (to begin and end writing), turn the turtle at a particular angle, and commands that repeat designated groups of commands. The TurtleArt programming interface allows students to program without the syntax difficulties common in traditional computer languages, and provides feedback in the form of art. TurtleArt also limits the problem space.

Problems that are solvable through TurtleArt are limited to "How do I draw something?" and outputs are graphical drawings.

TurtleArt uses an interface similar to the Scratch Programming interface (Resnick et al., 2009) which was designed to provide children with introductory computer experiences in programming and animation. These activities make creating with technology accessible and fun for children. Brennan and Resnick (2012) identified seven concepts that are used across many Scratch programs: sequences, loops, events, parallelism, conditionals, operators, and data. They also identify practices and thinking perspectives. Work on Scratch has identified that students in after school and summer programs learned programming concepts (Maloney, Peppler, Kafai, Resnick, & Rusk, 2008; Franklin, Conrad, Boe, & Nilsen, 2013). In fact, developers of Scratch Jr. (Flannery, Kazakoff, Bontá, Silverman, Bers, & Resnick, 2013) contend that, with an appropriately designed tool, children much younger than our population are capable of learning computational thinking skills.

Our study is novel in that it takes place in a classroom where children were free to move about the classroom and used technology (laptops) that was also mobile. We used conversations between students with their peers with their teacher to map communicated ideas as they move through the classroom. This method has potential use in studying how ideas spread and develop in collaborative and constructionist learning contexts.

Representing Collective Idea Development

In order to facilitate students' learning in a mobile classroom culture, it is important to understand how their ideas are expressed and developed through interactions. Our goal was to understand the ideas that children developed as they shared ideas and built on

the ideas of others to form new ideas. This generated a need for an efficient way to follow ideas as they were expressed, shared, and changed in the classroom. As such, we developed a method of representing the development of ideas. We used the theoretical construct of memes to narrow our focus.

A meme is a cultural analog to the gene (Dawkins, 1976). Unlike genes, memes are ideas; like genes, memes are replicated and changed. To be replicated and changed, an idea must first be visible. If a child thinks of a new way of doing something, it is not visible until she shares that idea in some way (e.g., drawing, talking). Only shared ideas are available for others to replicate or adapt. That is, memes do not exist until they are visible through spoken language or otherwise (Brilliant-Mills, 1994; Yoon, 2007). Dawkins's description of memes highlights that they change or evolve through cultural transmission. Memes are visible ideas that are taken up and replicated. Brilliant-Mills (1994) claimed that memes are produced every time we speak, and are transmitted by radio and television, in written words, and other media; yet, many of these are never replicated. Memes are not only passive; memes also shape minds. Csikszentmihalyi (1993), explains, "Once electricity is discovered, for example, it begins to suggest hundreds of new applications. So even though memes are initially shaped by the mind, they soon turn around and begin to shape minds" (p. 120). The existence of a meme allows individuals to think of new ideas that were impossible before the existence of a particular meme.

As a theoretical construct, memes are useful to our study because the construct helps us narrow our focus to particular kinds of ideas: those that are communicated, replicated, and changed. It also guides our data analysis and research questions to focus on replication and changing ideas across a classroom or group of students rather than examine how any one individual develops his or her ideas. We asked two questions,

- How can collective idea and skill development be mapped when children are replicating and changing (remixing) others' ideas? and
- What classroom interactions lead to skill development and which lead to innovation in a collaborative classroom environment?

Study Design

Research Context

The study took place at "Elm Elementary[1]" in California. Our participants included 20 third grade students (8-9 years old) and their teacher. Nearly one fifth of Elm's students were identified as English learners (19%) and one quarter of the students (24%) qualified for free or reduced lunches. Under-represented ethnic groups constituted nearly one-third of the student population (31.5%), of which the majority was Hispanic/Latino. In interviews at the beginning of the school year, the classroom teacher, Mr. Mills, discussed that he intentionally set up a classroom culture of sharing ideas and that he had particular interests in how the program mediated his students' sense of ownership of ideas and their collaboration with one another. The teacher used TurtleArt as a tool for teaching geometry, but also gave students free time to explore the programming platform.

Data Collection and Analysis Method

We collected data over one school year. As researchers, we were participant-observers in the classroom (Spradley, 1980). This means that we were collecting data and also working with children as they requested our help or our attention. Our primary data

[1] The name of the school, teacher, and all students are pseudonyms

source was video recordings taken once a week at a time when the students were regularly using laptops. Two video cameras recorded classroom interactions. During whole class discussions, one camera focused on the teacher and the other remained on a group. When students were working in small groups, each camera focused on a different table group of four focus students. In this classroom, students frequently got up from their group to talk to students in another group. The camera, however, remained focused on the table. This means that students who left the group were not captured during the time they were away and that students who visited the focus group from another group were captured. In this way we could observe how ideas entered groups as visitors brought them in or when a student left and came back with a new idea learned at another table.

Although we collected data when the students were using a variety of programs, we selected periods of time during which students were using TurtleArt because it provided a useful platform for children to express and develop ideas. Because the commands are entered through puzzle pieces (blocks) that remain visible on the screen, onlookers see both the user's process (code) and output (drawing). Video recordings of classroom time when students were using TurtleArt were event mapped (Collins & Green, 1992) to show major points of analytical interest (approximately six hours of video data representing 3 hours of classroom time). Selected episodes were then transcribed and coded based on verbal and non-verbal interactions between students and their peers, teachers, and laptops. Coding was iterative and recursive across time, events and ideas (Agar 1994).

Analysis stage 1: Identifying replicated ideas

We used an observable idea (meme) as the tracer unit (Green, 1983). Using an idea as a tracer unit means that our analysis followed ideas rather than individuals. In

particular, we were interested in ideas that were replicated either in their original form or replicated with changes from individual to individual.

In our first round of coding, we coded each turn of speech according to 1) What was being discussed (e.g., an output or the program to create it); 2) whether the student was talking to someone at their local table group, to someone visiting from another table group, or to someone else non-verbally by sharing their laptop screen; 3) who was promoting the idea (teacher or student).

We then made decisions about when memes began. We identified episodes in the video where students shared ideas. Sharing was accomplished in a number of ways: requesting others to look at his or her computer screen (e.g., "Look what I just made. It's awesome!"), requesting that other students look at a classmate's screen (e.g., "Oooh! Look what he drew"), and responding to a request from other students (e.g., "How did you make it go forward?") or the teacher (e.g. "Everyone, Tom has something interesting to show you"). A final method of sharing was through what we called "clustering," instances when multiple students gathered around to look at one student's screen, whether or not that student had called for attention. We marked these sharing events as analytically interesting points of meme communication in transcripts. We then used abductive reasoning to map the meme evolution following points of idea replication or change. Abductive reasoning is the process of examining a series of facts to suggest a theory of how they relate (Anderson, 1986). We then carefully inspected the transcript and video to identify ideas preceding and following these ideas to make inferences about the progression and evolution of ideas and how they might connect.

Our next step was to identify whether ideas were replicated. If other students took up the idea, we traced that idea forward in the event maps and reviewed the video to see how other students took up the idea. Specifically we looked to see if it changed

(and how so) as it passed through multiple students. In contrast, if the idea was not replicated, we did not pursue the idea.

Analysis stage 2: Representing idea evolution.

We used the analysis steps above to inform our representations of idea development, which we call Program-Output maps. Program-Output maps are derived from prior work by Green and Wallat's (1981) discourse flow maps and modifications to this work by Eshach (2010). Discourse flow maps show how talk is related to proceeding talk and follows analytically interesting points of discourse. Eshach used discourse flow maps to show how classroom discourse led to students' developing an agreed-upon understanding of science concepts among a classroom of students. With these maps, he was able to show how multiple ideas presented in a class discussion were taken up and combined with other ideas. We sought to map a similar phenomenon; however rather than converge or evolve along a single conceptual line or even a few conceptual lines, we expected ideas to diverge.

When looking at whether and how ideas changed, we looked specifically at two aspects - the graphical output (the drawing) and the set of commands (program) that the student used to create the output. We then represented changes in programming and changes in graphical output on orthogonal axes. This allowed us to follow an idea that was shared and to consider how that particular idea developed to be more efficient (changes in the program) or how students innovated (changes in the output) on the idea. This follows the representation used by Schwartz and colleagues (Schwartz, Bransford, & Sears, 2004) who placed efficiency and innovation on two separate perpendicular axes rather than on a single continuum. Though, we clarify that we do not use their meanings of innovation and efficiency.

As children developed facility and skill with the TurtleArt programming interface, they also used existing knowledge of the interface to create novel outputs. We considered ideas to evolve along the horizontal axis (programming) if the output (drawn image) remained relatively unchanged, but the process by which it was constructed became more streamlined. We considered this a development of programming facility. For example, in Figure 1, a square is drawn in eight steps (like command lines of code); yet, as shown in Figure 2, an identical square can be drawn in just three steps. This sort of change would be represented as a change in the horizontal axis because the output does not change but the method of achieving the output (the program) becomes more efficient.

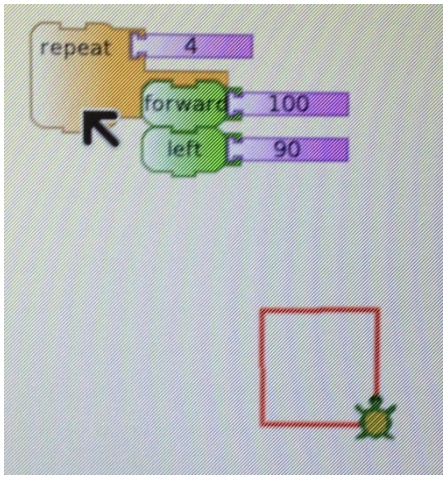

Figure 2. Screen shot of a square drawn with three steps.

Conversely, ideas were considered to move along the vertical axis (output) if the output changed but the program remained largely unchanged. For example, changing only the numerical values in the program in Figure 2 could result in drawing a star or octagon or other shape, a significant change in the output. Examples of Program-Output maps will be shown and discussed in the findings section.

Analysis stage 3: Exploring the actions and discourse between ideas

To understand the context leading to and facilitating the changes in output and programs, we looked at the classroom actions and discourse leading to and supporting or constraining the development of communicated ideas. Inspecting the videos across the duration of time between the instances when ideas were communicated provided insight into how the idea changed and to the classroom context and expectations that led to these particular changes. After mapping classroom discourse, we re-examined the transcripts and video to identify contextual patterns across instances where ideas were taken up and evolved. For each instance included on the meme map, we identified whether sharing the idea was teacher-prompted (e.g., the teacher pointing out a students' idea), or student-prompted (e.g., a child drawing attention to their own idea or that of a peer). We also noted whether these ideas were shared between students at the same table group or whether students were leaving their table (or collaborating with students from another table) when the shift occurred. These identifications were made by returning to the initial coding that was done on each turn of speech of the transcript.

Results

As described earlier, to identify which ideas students found valuable enough to replicate or change, we examined transcripts for phrases involving requests for attention or help and also by looking at the video for "clustering," times when student(s) were grouped around and looking at a peer's computer screen. The ideas that students shared consisted of visible graphical outputs such as shapes (e.g., stars, block letters, spirals), processes for creation such as programming functions (e.g., using the repeat command), and mathematical ideas (e.g., calculating the degrees in an angle). We then mapped the ideas according to how they evolved.

On each map, evolution toward a more efficient means of programming a particular output is mapped on the horizontal axis. More efficient uses often meant that the number of programming steps was smaller to create the same output. Evolution toward more innovative use of the same method (a different graphical output) is mapped on the vertical axis. More innovative uses often meant that very small changes in the program resulted in significant changes in the output. Ideas are represented as a series of Program/Output nodes. Each node (depicted as text in a rounded square) contains two lines of text. The first line (smaller font) is a brief description of *what* the student created (the Output), for example "Make Rectangle." The text in the lower half of the node in larger font briefly describes *how* the student programmed the output (Program), for example "multiple independent commands". Transitions between these nodes are assigned a number to facilitate discussion about the figure. Further, a black arrowhead indicates transitions that were prompted by the teacher or researcher while a grey arrowhead indicates those that were prompted by the students. We present three examples of the maps and associated descriptions below.

Example Classroom Events and Program-Output Maps

Example 1: Circles and rectangles (October 23, 2009)

Figure 3 depicts the ideas shared by the students in one of the focus groups on the first day that they used the TurtleArt program. On this day, students figured out how to move the mouse and the turtle in different ways to create simple shapes like circles and rectangles.

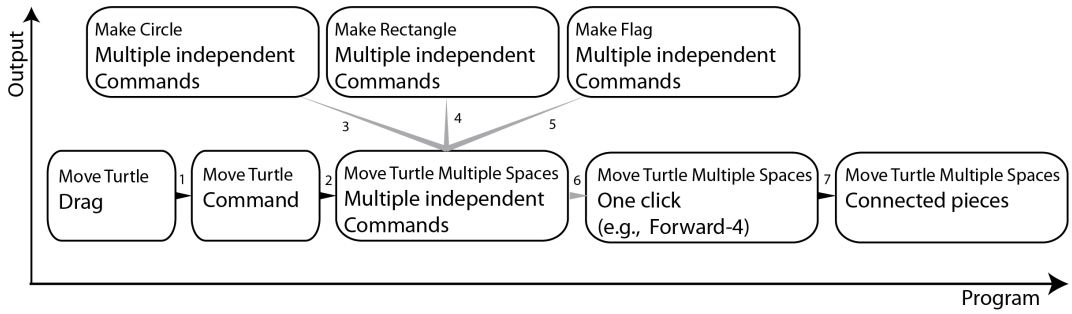

Figure 3. Program-Output Map 1, October 23, 2009. Black arrows represent transitions prompted by the teacher. Grey arrows represent transitions prompted by the students.

On this day, Mr. Mills introduced the TurtleArt program by showing his students the drawings made by third graders from the previous year using TurtleArt. After this introduction, he told his students to explore the TurtleArt program and that, "When you find something, you have to share it with everybody else. That's the only rule." The students then began working. Initially, they dragged the turtle across the screen. Even though the turtle moved, it did not draw.

One student Zoe dragged one of the commands onto her screen, clicked on it, and stated, "I get it." With this, Zoe changed from moving the turtle by dragging to moving the turtle with commands (which resulted in a drawn output). Daniel, who was sitting next to her, saw her screen and asked her how she did this (see Transcript 1).

Transcript 1: Daniel and Zoe figure out how to move Turtle

Daniel	But how do you draw though?
Researcher	[to Zoe] Do you know how you made it draw the little (points at Zoe's screen), like start moving?
Zoe	Yeah, go like back, back, back, back (clicks a puzzle piece on the screen each time she says back)

Researcher	[to Daniel] Oh, you click on it?
Daniel	Where?
Researcher	[to Daniel] So maybe try, click on the button
Zoe	Right, right, right, right, right, right. Look at this!
Daniel	It does work!

Transcript 1 is one example of a conversation that constituted transition 1 on program-output map 1 (Figure 3). Notice that the transfer of knowledge of how to click on the buttons from one child to another was facilitated by an adult in the room (the researcher in this case) and is thus represented by a black arrowhead. After this interaction, Daniel began to make his own creations. At another table, Ian called out, "Mr. Mills, you can move stuff, the numbers around," referring to numbers that could be typed into a text box within a command. These numbers determined length, angle or other variables depending on the command. In this case, the value indicated the distance the Turtle would move. Other students at his table began to explore this too, changing the number of pixels the turtle drew in one click. This led to a more efficient method of moving the turtle multiple spaces since students could now insert a number instead of click the command repeatedly. With this discovery, the students were able to make various shapes such as rectangles and circles. That multiple commands could be used together to create different shapes was shared and explained by children to their peers. In Figure 3, we represent the creation of different shapes as moves in the vertical because the focus of the students was not on the process, but on the output (transitions 3, 4, and 5). These are represented by grey arrowheads. Mr. Mills then asked the class, "What does this piece [pointing to a command] look like -- the green piece on my screen?" He led the students to recognize that the pieces resembled a puzzle and could

therefore be connected. The students continued drawing with the program, adding multiple commands together, one on top of the other, an even more efficient method of moving the turtle.

It is important to recall that this map is a map of how ideas spread across an entire classroom. It is not the evolution of ideas of a single student. Note that the first block “Move Turtle: Drag” is followed by two consecutive blocks that indicate development in facility with the programming environment. The third block, “Move Turtle Multiple Spaces: Multiple Independent Commands” is the first that leads to students innovating on the idea and producing multiple graphical outputs, innovating on each other’s program.

Example 2: Making shapes with 1 click (November 6, 2009)

Our second example occurred two weeks later, when Mr. Mills gave students another opportunity to explore TurtleArt. As with the first example, this lesson was largely unstructured; however, Mr. Mills periodically interrupted the students to make suggestions or pose challenges. Often these were based on outputs he observed on a student's screen. Program-Output Map 2 (Figure 4) focuses on the students developing the skill of creating shapes using multiple commands so that the output could be generated with "one click" (running the program). This idea also appeared at the end of the class period represented in Program-Output Map 1 (Figure 3). However, it was not generalized across the entire class at that time.

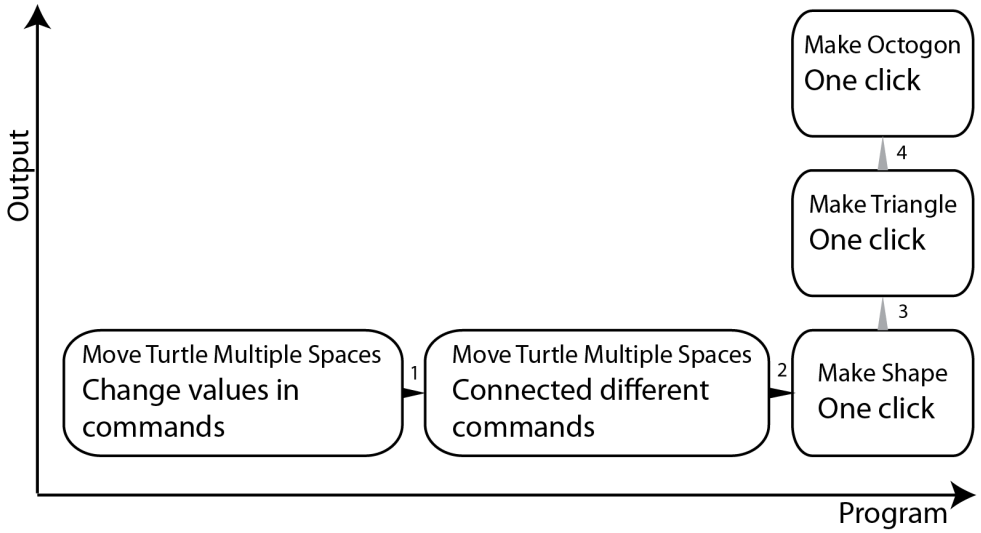

Figure 4. Program-Output Map 2, November 6, 2009.

After a few minutes exploring TurtleArt, Mr. Mills addressed the class and said, "I want you to make a rectangle." He asked the students how many sides and corners a rectangle had and explained that the "goal is to do this with one click of the mouse." Tom and Gary discussed the pieces, and the order they needed to appear in, to make a rectangle while trying out different combinations. Tom watched Gary's computer as he tried out various combinations (see Transcript 2).

Transcript 2: Students figure out how to make a rectangle

Tom	This is going to be complicated, look.
Gary	(leans over to look at Tom's computer screen)
Tom	(puts both hands in the air) "whoa, I made a square."
Researcher	Let's see
Tom	I made a ummm, kind of weird square
Ian	(walks over to Tom)

Tom	I need to flip it up, wait. Wait, whoa, whoa, whoa, I need to flip it upside down, It needs to be hundreds, then nineties
Researcher	Cool, try that
Tom	So that's my problem
Gary	(looking at his screen) 34, 1, 1, 1
Ian	(sitting back down at his desk) [To Tom] How do you change the numbers?

After Tom explained the process he went through, the other students at the table began to make their own single-click shapes including a triangle and octagon (transitions 3 and 4). For example, Gary made an octagon and then rotated the octagons slightly each time until it began to look like a circle. When the students made different shapes, attention was focused on how the process could be used for different outputs. In these transitions, children shared ideas with other children who adapted them (and, as such are indicated by grey arrowheads). While the researcher was present in the conversation, it was the children who facilitated the knowledge transfer.

Example 3: From stars to slinkys. (January 8, 2010)

The third and most complex example we present is from a class period two months later. In this class, the students developed the idea (with the help of their teacher) that they could use factors of 360 degrees to make a variety of shapes (see Figure 5). Mr. Mills told students that if they wanted, they could open up TurtleArt from their journal where the XO laptop saves their previous creations. Cory opened one of his drawings and Tom glanced at it, saying "Woah!" and shared what Cory had done with another classmate. Cory then asked Tom to recreate his drawing so that they could make the process more efficient.

Transcript 3: Figuring out how many clicks an object requires.

Cory	Look at this! Heather, look at this
Mr. Mills	How many clicks, Cory?
Cory	Um, a lot
Tom	How many clicks did you do?
Cory	A lot
Tom	No, just tell me
Cory	I don't know, I don't remember
Tom	Clean and redo it over and then tell me how many clicks and I can make it do it in one click
Cory	I know how to do that
Tom	(leaning over and pointing to Cory's screen) repeat
Tom	No nahnahnana. You put it on there.
Cory	forward, forward, forward, forward, forward. Like that?
Tom	K good, now forward, click forward
Cory	Whoa, that's a lot [of steps]
Tom	(In a loud voice) He did it in one click, he taught himself how to do it that.

During the conversation in Transcript 3, Cory's process changed from clicking several times to using the repeat function to make the shape with one click. Tom helped him redraw his design and find the number of steps to use in the repeat sign. The number they chose was very large, though, so it took awhile to run and resulted in a many-pointed star that repeated itself many times, rotating slightly with each star. The resulting output looked like a solid circle with patterns in concentric circles. The students called this a "3-D circle." Cory shared this output, saying, "I made a 3-D

circle." Ian, Tom, and Heather all looked at Cory's screen. This is transition 2 in program-output map 3 (Figure 5).

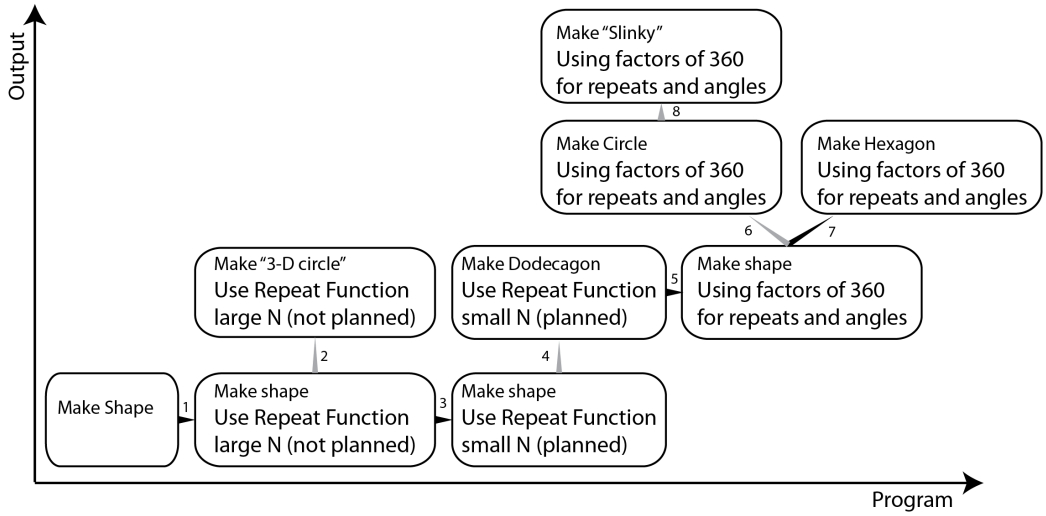

Figure 5. Program-Output Map 3, January 8, 2010.

After this conversation, Mr. Mills walked by and suggested that Cory explore different numbers that might make the turtle draw the shape in less time (transition 2). When Ava used smaller numbers for her repeat function, she made a dodecahedron and stated, “Look - it’s 12 sides. Mr. Mills, what kind of shape is this?” Mr. Mills reminded the students about math they had learned the previous day and suggested that students try using factors of 360 as the number of degrees and number of repeats. He worked with the class to draw a square and a hexagon and let them explore other combinations. Mr. Mills challenged his students to use the same commands (with different values) from their shapes to make a circle. A few students requested attention or help with making these new circles. While students were showing Mr. Mills their circles and cleaning up for recess, Heather turned her circle into a slinky and called Ava over to see her idea.

In this final example, there are three developments in programming that led to innovation. The first and the second developments were variations on the idea of using

a repeat function. The first was not planned. The student put in a number and observed what happened. In the second case, the student thoughtfully considered how the repeat function could be used to create an intended outcome. The third development was a math idea – students used factors of 360 for the angles and number of repeating instances. Though it should be noted that this does not take into account students' ability to consider the turtle's orientation when selecting angles, an idea they did not develop until later in the year.

Contextual Patterns across the Development of Ideas

The three examples reveal a pattern regarding teacher and student-prompted sharing. Changes on the horizontal axis (developing more efficient means of programming an image) were most often prompted by questions or comments from the teacher (black arrowheads), whereas changes on the vertical axis (using the commands to create novel outputs) were most often prompted by questions or comments from students (grey arrowheads). In an example of a teacher-prompted idea, Tom and Cory were working on making a shape in one click when the teacher suggested, "explore that number because you might be able to do that same thing in a lower number that won't take 300 hours." The two students worked to make the process more efficient by using a smaller number in the repeat function before proclaiming their success and sharing the idea with others. In this case, the teacher suggested that large numbers in the repeat function may take a long time to run. The students adapted their idea to be more efficient. When trying to program particular outputs, students often prompted their peers such as when Patricia asked, "Who knows how to do a circle?" Ava raised her hand saying, "I do" and Patricia walked over to Ava's desk to work with her. In this instance, Patricia asked a question about how to make a certain output. Ava then shared the idea leading

Patricia to come up with an evolved version of her own. Other times, a student became excited by an output and shared it with a friend. While the teacher's interactions with students allowed them to develop more efficient ideas, it was students themselves that seemed to prompt the development of more innovative outputs.

Discussion

In all three cases presented, we identified instances where students began to share ideas. In these cases, students took ownership of ideas and pursued their own transformative directions. We also identified the contexts under which students began to take ownership and innovate on ideas. Our analysis showed that the third grade students in this study took up ideas that were shared and changed these ideas, using similar programs to create original outputs. In some cases, ideas developed in one class meeting were taken up again and further replicated and evolved in subsequent class periods, sometimes weeks or months later. This is consistent with the literature on memes (e.g., Blackmore 2000; Dawkins 1976). The shared ideas became part of the corpus of knowledge and ideas that the children could access when using TurtleArt.

The choice of mapping idea development along two orthogonal axes – that of developing efficiency in programming and using that programming to create new graphical outputs—allowed us to recognize that children were both developing computer programming skills and figuring out how to use these nascent skills to solve new problems within the limited problem space and set of commands from the TurtleArt program. We found this particular way of mapping classroom discourse fruitful in identifying how the children were innovating on the programming ideas they were developing while simultaneously learning the vocabulary, syntax, and programming concepts necessary to work within the TurtleArt program.

We propose that identifying instances where students *innovated* by producing novel graphical output may be productive indicators of learning in a collaborative environment. The children in this study appeared to be learning many concepts of computer science by communicating and innovating on ideas. For example, the use of the repeat function to draw complex shapes was not at all obvious to the students to begin with, but once they learned this function, they used it to create a variety of shapes. This is evidenced by the multiple meme flow maps that show how once the repeat function was grasped, students used it to create new outputs. The repeat function as it was used in this classroom is a simplified looping function (the command allows for more complicated looping functions but this possibility was not explored by the students in this study). Examining the ideas that, once grasped, were used to create novel outputs may lead to identifying key understandings in beginning computer programming.

We found that elements of the classroom context were vital to the process of identifying and representing the developing ideas of the students. First, the program and output created by individual students needed to be visible to onlookers. In our case Turtle Art was an ideal programming environment for our research precisely because others could easily observe the output and script. Other interfaces with this feature like Scratch would also be appropriate. Second, allowing children and computers to move around the classroom facilitated the movement of ideas and allowed us to observe how those ideas were changed. This may not have been possible if children were seated at stationary desktop computers. In fact, every change in innovation, except one, was prompted by students who had moved from their original seats. Adopting and adapting ideas is crucial for recognizing *which* ideas allow students to make innovative outputs.

Finally, the method of analysis that represented idea development along orthogonal axes was key to identifying places where important ideas might be found.

Our findings also show that this particular type of activity resulted in students showing significant ownership of ideas. In fact, showing ownership of ideas was one of the primary indicators we used to identify memes. Though we did not explicitly analyze idea ownership in this study, it appeared that students called more attention to their own or others' work when the evolution resulted in an innovation of an idea rather than when the evolution led to a more efficient method of making a very similar output. This leads to the hypothesis that ownership of ideas is associated with more creative and innovative work. This is an empirical question that could be investigated with further data collection and analysis.

This study is limited by its scope and focus. The location of the cameras meant that only a few students were focused on during each day of data collection. This prevented us from making conclusive claims about the ways ideas travelled across the entire class. The ideas discussed here were likely taken up and modified in additional ways by students who were not part of focal groups and thus unobserved by researchers. In fact, there is some evidence of this in the existing video when we see a student from another table walk over to one of the focal groups, look over a classmate's shoulder to view a screen, and then leave the group presumably to return to his own table and possibly share what he has learned. Future research could address this limitation by having multiple video cameras focused on each group of students to ensure that all student conversations are heard. The study is also limited to a particular type of learning that occurred while using TurtleArt. Because TurtleArt allows both the process (programming) and the output (drawing) to be visible to others, it is an unusual learning context. Adapting this process of mapping ideas according how they change in

efficiency and innovation to other learning contexts, while possible, may be difficult.

Concluding Thoughts

Concluding Thoughts

Understanding the processes that students go through to solve problems and create new ideas has both practical and theoretical applications regarding the ways students interact with new technologies. Learning how children take up and use these different types of ideas as they interact with new technology has implications for the types of opportunities for learning with technology we provide children.

References

Agar, M. (1994). *Language shock: Understanding the culture of conversation*. New York: Morrow.

Anderson, D.R. (1986). The evolution of Peirce's concept of abduction. *Transactions of the Charles S. Peirce Society*, 22(2), 145-164.

Blackmore, S.J. (2000). *The meme machine*. Oxford University Press.

Bontá, P., Papert, A., Silverman, B., (2010). TurtleArt. *Proceedings of Constructionism,* August 16-20, Paris, France.

Brennan, K., & Resnick, M. (2012). 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.

Brilliant-Mills, H. (1994). Becoming a mathematician: Building a situated definition of mathematics. *Linguistics and Education*, 5(3-4), 301-334.

Collins, E. & Green, J.L. (1992). Learning in classroom settings: making or breaking a culture. In Hermine Marshall (Ed.) *Redefining students learning*. Norwood, New Jersey: Ablex.

Computer Science Task Force (2011), CSTA K-12 Computer Science Standards.

Csikszentmihalyi, M., (1993). *The Evolving Self: A psychology for the third millennium.* New York: Harper Collins Publishers.

Dawkins, R., (1976). *The Selfish Gene*. New York: Oxford University Press.

Dunleavy, M., & Heinecke, W. F. (2008). The impact of 1: 1 laptop use on middle school math and science standardized test scores. *Computers in the Schools*, 24(3-4), 7-22.

Eshach, H. (2010). An analysis of conceptual flow patterns and structures in the physics classroom. *International Journal of Science Education*, 32(4), 451-477.

Flannery, L., Kazakoff, E., Bontá P., Silverman, B., Bers, M., & Resnick, M., (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children (IDC).* New York, NY.

Franklin, D., Conrad, P., Boe, B., Nilsen, K., Hill, C., Len, M., ... & Waite, R. (2013, March). Assessment of computer science learning in a Scratch-based outreach program. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 371-376). ACM.

Gee, J. P. (2003). *What video games have to teach us about learning and literacy*. New York: MacMillan.

Green, J. L., (1983). Exploring classroom discourse: Linguistic perspectives on teaching-learning processes. *Educational Psychologist*, 1532-6985, Volume 18, Issue 3, 180-199.

Green, J., & Wallat, C. (1981). Mapping instructional conversations: A sociolinguistic ethnography. In J. Green & C. Wallat (Eds.), *Ethnography and language in educational settings* (pp. 161–205). Norwood, NJ: Ablex.

Hatano, G., & Inagaki, K. (1986). Two courses of expertise. In H. A. H. Stevenson & K. Hakuta (Eds.), *Child development and education in Japan* (pp. 262-272). New York: Freeman.

Hayes, E., & Games, I. (2008). Making computer games and design thinking: a review of current software and strategies. *Games and Culture*, 3, 309-332.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: urban youth learning programming with Scratch. *ACM SIGCSE Bulletin*, 40(1), 367-371.

Monroy-Hernández, A. & Resnick, M. (2008). Empowering kids to create and share programmable media. *Interactions*, 50-53.

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

Papert, S., (1991). Situating Constructionism, in S. Papert and I. Harel, *Constructionism: research reports and essays 1985-1990*. Norwood, NJ: Ablex Publishing Corporation.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM*, November 2009.

Resnick, M., (2006). Computer as Paint Brush: Technology, Play, and the Creative Society. In Singer, D., Golikoff, R., and Hirsh-Pasek, K. (eds.), *Play = Learning: How play motivates and enhances children's cognitive and social-emotional growth.* Oxford University Press.

Richtel, M. (2014). Reading, Writing, Arithmetic, and Lately, Coding. *The New York Times.* Accessed online http://www.nytimes.com/2014/05/11/us/reading-writing-arithmetic-and-lately-coding.html

Schwartz, D., Bransford, J., & Sears, D. (2005). Efficiency and Innovation in transfer. In J. Mestre (Ed.), *Transfer of learning from a modern multidisciplinary perspective* (pp. 1-51). Greenwich: Information Age Publishing.

Spradley, J. (1980). *Participant observation*. New York, Chicago, San Francisco, Dallas, Montreal, Toronto, London, Sydney: Holt, Rinehart and Winston.

Sternberg (1999). *Handbook of creativity*. Cambridge: Cambridge University Press.

Topper, A., & Lancaster, S. (2013). Common Challenges and Experiences of School Districts That Are Implementing One-to-One Computing Initiatives. *Computers in the Schools*, *30*(4), 346-358.

Wiggins, G., & McTighe, J. (2005). *Understanding by design*. Alexandria, VA: Association for Supervision and Curriculum Development.

Yoon, S. (2007). Using memes and memetic processes to explain social and conceptual influences on student understanding about complex socio-scientific issues. *Journal of Research and Science Teaching*, 45(8), 900-921.