New frameworks for studying and assessing the development of computational thinking

Karen Brennan (kbrennan@media.mit.edu)
Mitchel Resnick (mres@media.mit.edu)
MIT Media Lab

Brennan, K., & Resnick, M. (2012). Using artifact-based interviews to study the development of computational thinking in interactive media design. Paper presented at annual American Educational Research Association meeting, Vancouver, BC, Canada.

Abstract

Computational thinking is a phrase that has received considerable attention over the past several years – but there is little agreement about what computational thinking encompasses, and even less agreement about strategies for assessing the development of computational thinking in young people. We are interested in the ways that design-based learning activities – in particular, programming interactive media – support the development of computational thinking in young people. Over the past several years, we have developed a computational thinking framework that emerged from our studies of the activities of interactive media designers. Our context is Scratch – a programming environment that enables young people to create their own interactive stories, games, and simulations, and then share those creations in an online community with other young programmers from around the world.

The first part of the paper describes the key dimensions of our computational thinking framework: *computational concepts* (the concepts designers engage with as they program, such as iteration, parallelism, etc.), *computational practices* (the practices designers develop as they engage with the concepts, such as debugging projects or remixing others' work), and *computational perspectives* (the perspectives designers form about the world around them and about themselves). The second part of the paper describes our evolving approach to assessing these dimensions, including project portfolio analysis, artifact-based interviews, and design scenarios. We end with a set of suggestions for assessing the learning that takes place when young people engage in programming.

Designing interactive media

"Fireflies" is a music video created by Tim, who is 8 years old. He selected one of his favorite pop songs, carefully listened to the lyrics, and imagined how the words could be represented visually. He drew the characters and programmed their behavior, assembling the pieces in a timed sequence.

"Countries" is a simulation created by Shannon, who is 14 years old. She loves SimCity, a computer game that simulates multiple dimensions of a city for the player to control, and has spent hundreds of hours playing. Based on her interest in SimCity and what she was learning in

history class, she developed a simulation about virtual countries, with the player making decisions about how to respond to economic, agricultural, and political crises.

"10 Levels" is a game created by Renita, who is 10 years old, and her younger brother. She played a similar game on a popular game site and decided to design her own version of the game, which involves navigating the main character from the start of the level to the end of the level without colliding with hazards (such as spikes, fireballs, and trapdoors).

All three of these projects were created by young people using Scratch, a computational authoring environment developed by the Lifelong Kindergarten research group at the MIT Media Lab. With Scratch, young people can design their own interactive media – including stories, games, animations, and simulations – by snapping together programming-instruction blocks, just as one might snap together LEGO bricks or puzzle pieces (Resnick et al., 2009).

In addition to the authoring environment, there is an online community where young people can share their projects, just as videos are shared on YouTube. The Scratch online community, which was launched in May 2007, has grown steadily over the past five years; hundreds of thousands of young creators (mostly between the ages of 8 and 16) have shared more than 2.5 million projects. Community members can interact with projects (try them out, or download to see how they work) and with other members (leave comments, or mark someone as a friend) (Brennan, Resnick, & Monroy-Hernandez, 2010; Brennan, Valverde, Prempeh, Roque, & Chung, 2011).

Computational thinking

How do we describe what Tim, Shannon, and Renita are learning as they participate as designers of interactive media with Scratch? What is the learning that is supported by programming interactive media, as opposed to making a video with editing software or playing a video game?

We have been intrigued by the phrase *computational thinking* as a device for conceptualizing the learning and development that take place with Scratch. Although computational thinking has received considerable attention over the past several years, there is little agreement on what a definition for computational thinking might encompass (Allan et al., 2010; Barr & Stephenson, 2011; National Academies of Science, 2010). Cuny, Snyder, and Wing (2010) define computational thinking as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" – a description that aptly (if somewhat tersely) frames the work of computational creators.

The phrase *computational thinking* helps us think about learning with Scratch, and, in turn, we believe that programming with Scratch provides a context and set of opportunities for contributing to the active conversations about computational thinking. We are interested in the ways that design-based learning activities – in particular, programming interactive media – support the development of computational thinking in young people. Part of this interest is fuelled by the growing availability of tools that enable young people to design their own interactive media. But, more importantly, this interest is rooted in a commitment to learning

through design activities, a constructionist approach to learning that highlights the importance of young people engaging in the development of external artifacts (Kafai & Resnick, 1996).

Over the past several years, by studying activity in the Scratch online community and in Scratch workshops, we have developed a definition of computational thinking that involves three key dimensions: *computational concepts* (the concepts designers employ as they program), *computational practices* (the practices designers develop as they program), and *computational perspectives* (the perspectives designers form about the world around them and about themselves). Observation and interviews have been instrumental in helping us understand the longitudinal development of creators, with participation and project portfolios spanning weeks to several years, and workshops have been instrumental in understanding the practices of the creator-in-action.

Computational thinking concepts

As young people design interactive media with Scratch, they engage with a set of computational concepts (mapping to Scratch programming blocks) that are common in many programming languages. We have identified seven concepts that are highly useful in a wide range of Scratch projects, and which transfer to other programming (and non-programming) contexts: *sequences, loops, parallelism, events, conditionals, operators,* and *data*. For each concept, we provide a definition of the concept and a concrete example from a Scratch project.

Concept: Sequences
A key concept in programming is that a particular activity or task is expressed as a series of individual steps or instructions that can be executed by the computer. Like a recipe, a sequence of programming instructions specifies the behavior or action that should be produced. For example, the cat object can be programmed to move a short distance across the screen and declare, "I'm programming!" with the sequence of instructions shown in Figure 1.

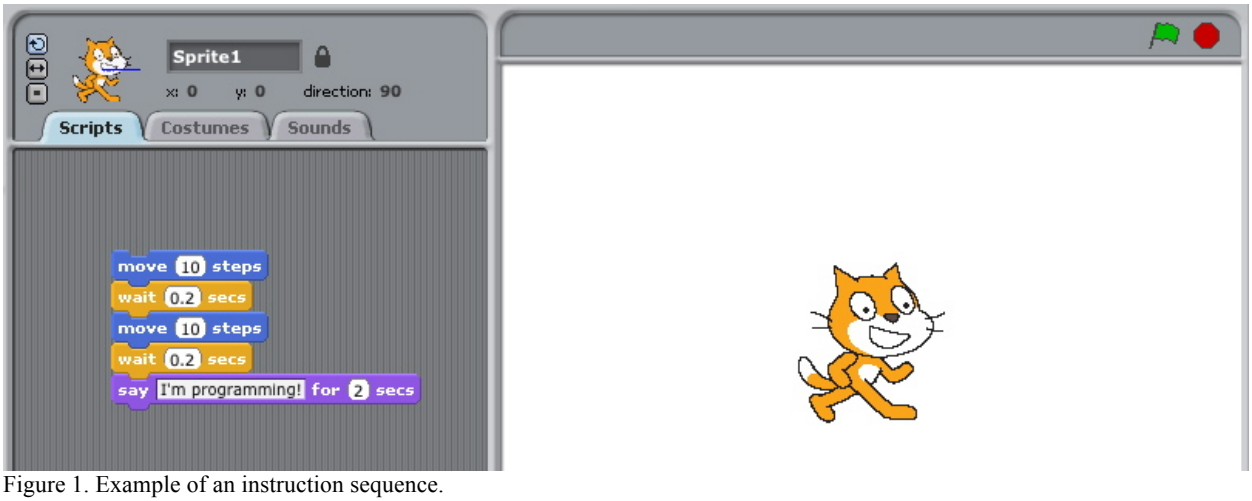

Figure 1. Example of an instruction sequence.

Concept: Loops
In the previous example, the cat was programmed to move 10 steps, wait 0.2 seconds, and then to repeat the action – moving another 10 steps, and waiting another 0.2 seconds. What if, instead

of a single repetition of the action, we want the cat to move and wait three more times? We could easily add more *move* and *wait* blocks. But what if we wanted the cat to move and wait 50 or 100 or 1000 more times? Loops are a mechanism for running the same sequence multiple times. Figure 2 illustrates how a loop can be used to express a sequence of instructions more succinctly. Instead of moving and waiting with 8 consecutive blocks, we use three blocks: *move 10 steps*, followed by *wait 0.2 secs*, enclosed by *repeat* with the desired number of iterations.

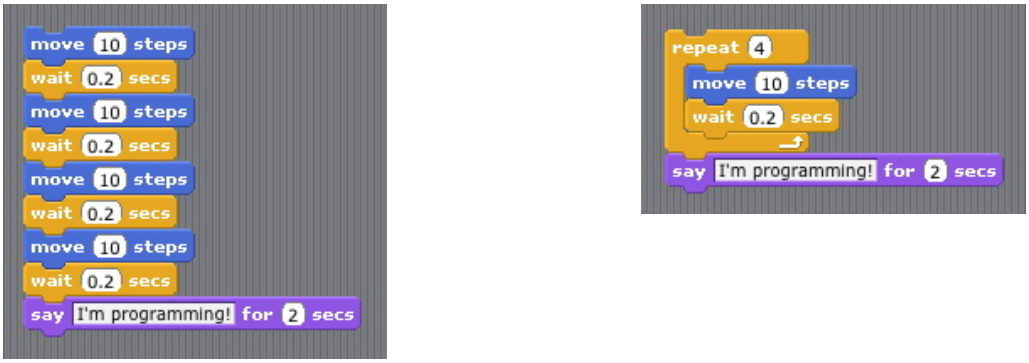

Figure 2. A sequence of repeated instructions expressed as a loop.

Concept: Events

Events – one thing causing another thing to happen – are an essential component of interactive media. For example, a start button triggering the beginning of a music video, or the collision of two objects causing a game's score to increase. Figure 3 illustrates different situations in which an event will produce an action: (1) when the green flag is clicked, the object will turn forever in 15 degree increments, (2) when the space key is pressed, the object will move up and down, and (3) when the object is clicked with the mouse, it will display a speech bubble for 2 seconds that says, "Hello!"

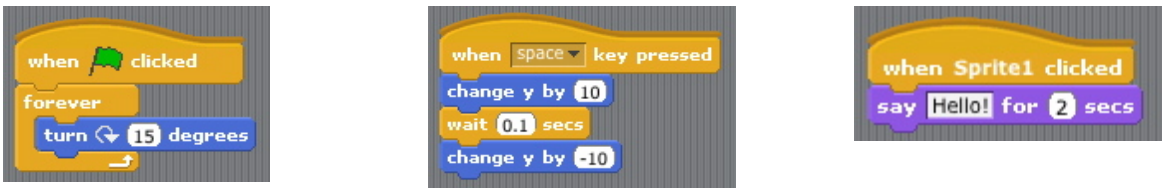

Figure 3. Examples of events producing actions.

Concept: Parallelism

Most modern computer languages support parallelism – sequences of instructions happening at the same time. Scratch supports parallelism across objects. For example, a dance party scene might involve several characters dancing simultaneously, each with a unique sequence of dance instructions. Scratch also supports parallelism within a single object. In Figure 4, the Scratch cat has been programmed to perform three sets of activities in parallel in response to the *when green flag clicked* event: (1) continuously play a background soundtrack, (2) continuously dance back and forth, and (3) introduce itself and its interests.

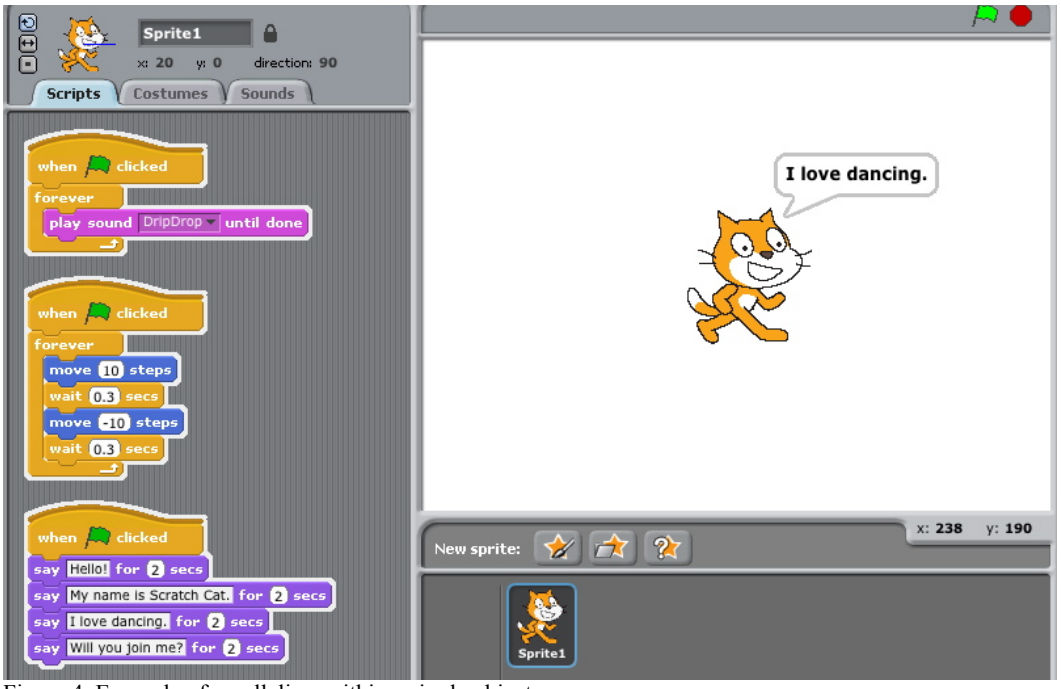

Figure 4. Example of parallelism within a single object.

Concept: Conditionals

Another key concept in interactive media is conditionals – the ability to make decisions based on certain conditions, which supports the expression of multiple outcomes. Figure 5 illustrates the use of a conditional – the *if* block – to determine the visibility of an object. If the cube is *touching color yellow*, then it should fade out and reappear for the next level of the game; otherwise, it should remain visible.

Figure 5. Example of conditionals.

Concept: Operators

Operators provide support for mathematical, logical, and string expressions, enabling the programmer to perform numeric and string manipulations. Scratch supports a range of mathematical operations (including addition, subtraction, multiplication, division, as well as

functions like sine and exponents) and string operations (including concatenation and length of strings). Figure 6 illustrates Scratch's operator blocks.

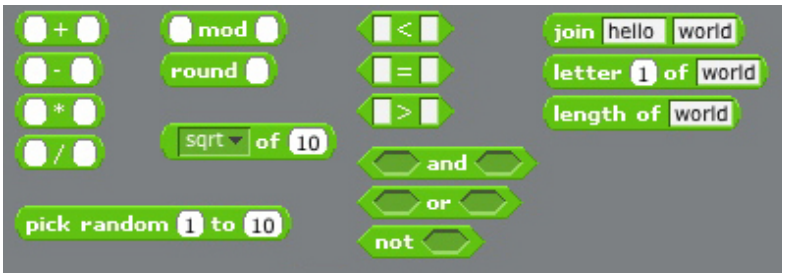

Figure 6. Operator blocks.

Concept: Data

Data involves storing, retrieving, and updating values. Scratch currently offers two containers for data: variables (which can maintain a single number or string) and lists (which can maintain a collection of numbers or strings). Keeping score in a game is a frequent motivator for young designers to explore variables. Figure 7 demonstrates how a variable is used to keep score in a game; for each little fish eaten by the large fish, the score increases by 1.

Figure 7. Using a variable to keep score.

Computational thinking practices

From our interviews with and observations of young designers, it was evident that framing computational thinking solely around *concepts* insufficiently represented other elements of designers' learning and participation. The next step in articulating our computational thinking framework was to describe the processes of construction, the design *practices* we saw kids

engaging in while creating their projects. Computational practices focus on the process of thinking and learning, moving beyond *what* you are learning to *how* you are learning.

Although the young people we interviewed had adopted a variety of strategies and practices for developing interactive media, we observed four main sets of practices: *being incremental and iterative*, *testing and debugging*, *reusing and remixing*, and *abstracting and modularizing*. Interactive media creation is a powerful context for developing these practices, which are useful in a variety of design activities, not just programming. To illustrate these practices in action, we use the case of Renita and her younger brother, and their process for developing the multi-stage obstacle/adventure game "10 Levels".

Practice: Being incremental and iterative
Designing a project is not a clean, sequential process of first identifying a concept for a project, then developing a plan for the design, and then implementing the design in code. It is an adaptive process, one in which the plan might change in response to approaching a solution in small steps. In conversations with Scratchers, they described iterative cycles of imagining and building – developing a little bit, then trying it out, and then developing further, based on their experiences and new ideas. Renita described this process with "10 Levels", and how she used each iteration as an opportunity to solicit feedback and new ideas:

> *I: OK, this is a complicated program. How long have you been working on it?*
> R: Maybe three, or maybe two, weeks.
> *I: Are you working on it every day?*
> R: Like off and on, maybe even a month. Whenever I finished one of the levels, I would show it to my brother.
> *I: You talked a bit about how you did a lot of the programming and your brother helped with the concept of the project. What was your process like?*
> R: We first came up with it on the way, but for levels 8, 9, and 10 we actually planned beforehand. My brother had this great idea about level 10 having pins and bowling balls. He said, "That should be level 8!" and I said, "No, no that should be level 10, that's really hard." and he said, "OK, OK, OK."

Practice: Testing and debugging
Things rarely (if ever) work just as imagined; it is critical for designers to develop strategies for dealing with – and anticipating – problems. In interviews, Scratchers described their various testing and debugging practices, which were developed through trial and error, transfer from other activities, or support from knowledgeable others. Initially, Renita could not think of a time during the process of developing "10 Levels" when she had gotten stuck on a problem with the project. After a few moments of quiet contemplation, she started to list the various testing and debugging practices she had used in this (and other) projects:

> *identify (the source of) the problem*
> *read through your scripts*
> *experiment with scripts*
> *try writing scripts again*
> *find example scripts that work*

tell/ask someone else about the problem
take a break

Practice: Reusing and remixing

Building on other people's work has been a longstanding practice in programming, and has only been amplified by network technologies that provide access to a wide range of other people's work to reuse and remix. One goal of the Scratch online community is to support young designers in reusing and remixing, by helping them find ideas and code to build upon, enabling them to potentially create things much more complex than they could have created on their own. Reusing and remixing support the development of critical code-reading capacities and provoke important questions about ownership and authorship. What is reasonable to borrow from others? How do you give appropriate credit to others? How do you assess cooperative and collaborative work? Renita's project benefitted from reuse and remixing in at least two ways. The project *idea* was a remix of a project she had first seen on a popular gaming website, and later found on the Scratch website (Figure 8).

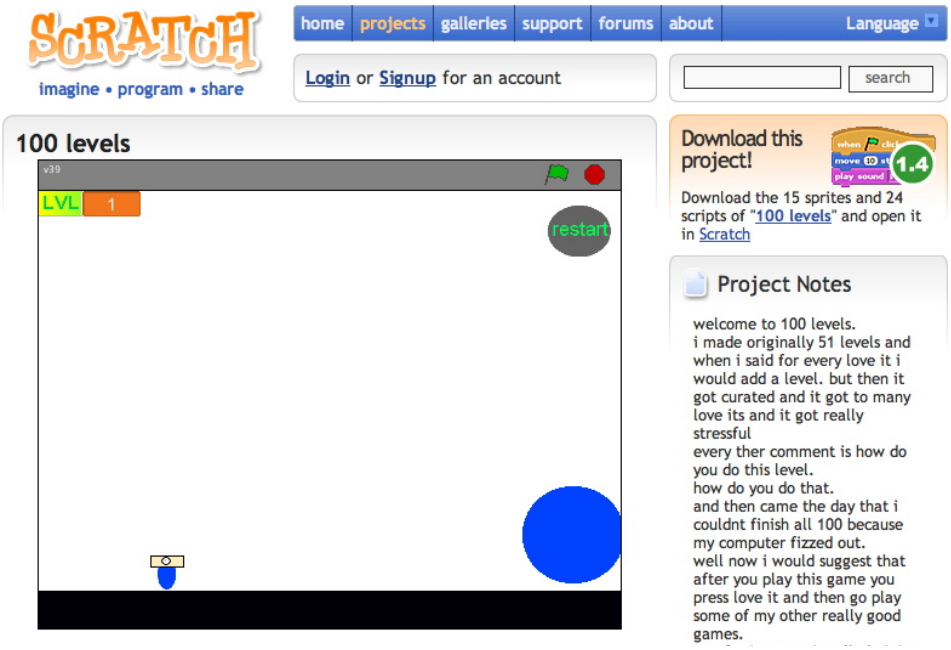

Figure 8. Project that served as inspiration for reuse/remix.

She also reused and remixed at the code level, incorporating a sprite from the Scratch library that included jet-pack simulation code (JetPack Girl, shown in Figure 9) as the main character in her game.

Figure 9. Reusing code from the Scratch library.

Practice: Abstracting and modularizing

Abstracting and modularizing, which we characterize as building something large by putting together collections of smaller parts, is an important practice for all design and problem solving. In Scratch, designers employ abstraction and modularization at multiple levels, from the initial work of conceptualizing the problem to translating the concept into individual sprites and stacks of code. Figure 10 shows one way in which Renita employed modularization and abstraction in "10 Levels" by separating out the different behaviors or actions of her central object that is navigating the obstacles. The first stack of code controls the object's on-screen movement. The second stack of code controls the object's visual appearance. The third stack of code controls the various events associated with obstacles, such as resetting the level if the object collides with a hazard. Modularizing the object's behaviors made it easier for Renita to think about (and test/debug) the different parts, and for others to read.

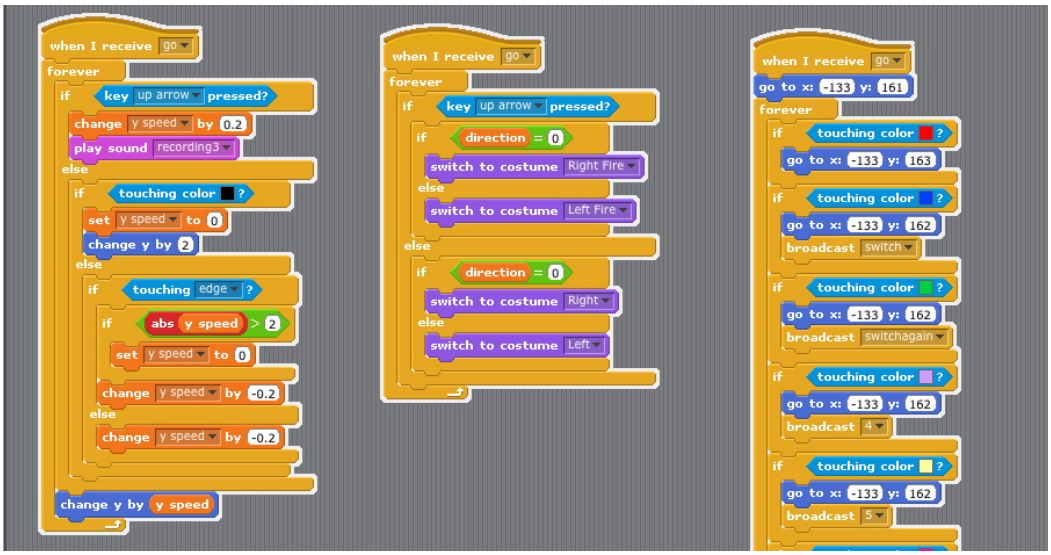

Figure 10. A complex set of instructions organized by functionality into three separate code stacks.

Computational thinking perspectives

In our conversations with Scratchers, we heard young designers describe evolving understandings of themselves, their relationships to others, and the technological world around them. This was a surprising and fascinating dimension of participation with Scratch – a dimension not captured by our framing of concepts and practices. Thus, as the final step in articulating our computational thinking framework, we added the dimension of *perspectives* to describe the shifts in perspective that we observed in young people working with Scratch.

Perspective: Expressing
People are surrounded by interactive media, but most of our experiences with interactive media are as consumers. We spend time pointing, clicking, browsing, and chatting – activities that are important for learning to use technology, but not sufficient developing as a computational thinker. A computational thinker sees computation as more than something to consume; computation is something they can use for design and self-expression. A computational thinker sees computation as a medium and thinks, "I can create." and "I can express my ideas through this new medium." They see it as a medium that is different from other things, as expressed by a 13-year-old girl from the United States:

> I like Scratch better than blogs or social networking sites like Facebook because we're creating interesting games and projects that are fun to play, watch, and download. I don't like to just talk to other people online. I like to talk about something creative and new.

and as a medium with considerable opportunities, as expressed by a 9-year-old girl from Australia:

> *I: What's your favorite part about Scratch?*
> Scratcher: Um, maybe that once you upload the whole working thing that you have a project. Or maybe it's just the creativity of Scratch.
> *I: What do you mean by that? Can you tell us a bit more about what it means to be creative with Scratch?*
> Scratcher: Well, it's just that there's endless possibilities. It's not like you can just make this project or this project and that's all that you can make.

Perspective: Connecting
Creativity and learning are deeply social practices, and so designing computational media with Scratch is unsurprisingly enriched by interactions with others. In interviews and observations, we noted the wide variety of ways in which an individual Scratcher's creative practice benefitted from access to others, through face-to-face interactions or (particularly in the case of the Scratch online community) online networks (Brennan & Resnick, in press). Young people described the power of having access to new people, projects, and perspectives via these networks, a shift in perspective expressed succinctly as, "I can do different things when I have access to others."

In interviews, having access to others was described in two ways: the value of creating *with* others, and the value of creating *for* others. By creating *with* others, young Scratchers described how they were able to do more than they could have on their own, whether by having questions

answered in online forums (such as getting help fixing a particular bug in a project), or studying and remixing others' code (such as finding a side-scrolling project base to build from), or establishing intentional partnerships and collaborations (such as Scratchers who form "design studios" or "production companies" to create projects together). By creating *for* others, young Scratchers experienced the value of authentic audience. They appreciated that others were engaging with and appreciating their creations, whether by *entertaining* others (such as building up an audience of followers for a series of soap opera-esque projects), *engaging* others (such as designing a survey for other community members to respond to), *equipping* others (such as developing assets for other Scratchers to use in their own projects), or *educating* others (such as making tutorial projects that help other Scratchers learn something about Scratch, like how to use trigonometry in physics simulations or how to make popular projects).

Perspective: Questioning

As Bandura (2001) observed, "everyday life is increasingly regulated by complex technologies that most people neither understand nor believe they can do much to influence" (p. 17). With the computational perspective of questioning, we look for indicators that young people do not feel this disconnect between the technologies that surround them and their abilities to negotiate the realities of the technological world. Young people should feel empowered to ask questions about and with technology – "I can (use computation to) ask questions to make sense of (computational things in) the world." As one example of this shift in worldview, an 11-year-old described the new perspective with which he sees the objects surrounding him:

> I love Scratch. Wait, let me rephrase that – Scratch is my life. I have made many projects. Now I have what I call a "programmer's mind." That is where I think about how anything is programmed. This has gone from toasters, car electrical systems, and soooo much more.

Questioning involves interrogating the taken-for-granted, and, in some cases, responding to that interrogation through design. For example, Scratch the programming environment is a computational artifact that has certain design affordances and limitations. Some young members of the community questioned those limitations and teamed up to make a derivative version of Scratch that included blocks they felt should be included and developed a website where other people could download their modified version of Scratch. This involved not only recognizing that Scratch is a designed artifact in the world that can be modified, but also that they, as designers of computational media, were empowered to modify it.

Assessing learning through design

Having articulated our framework for computational thinking (*concepts*, *practices*, and *perspectives*), we now describe three approaches to assessing the development of computational thinking in young people who are engaging in design activities with Scratch. For each assessment approach, we describe the details of the assessment process, provide example data where available, and discuss the approach's strengths and limitations.

Approach #1: Project portfolio analysis

Each member of the Scratch online community has a profile page that displays their creations, as well as other dimensions of participation, such as projects they have favorited and Scratchers they follow. For example, Figure 11 shows the profile page of a 17-year-old Scratcher who has been a member of the community for more than 3 years and has posted 49 projects.

Figure 11. Scratch community member profile page.

Researchers at the College of New Jersey have developed a set of visualizations called Scrape (http://happyanalyzing.com/) that analyze the programming blocks within Scratch projects (Wolz, Hallberg, & Taylor, 2011). Our first approach to assessing the development of computational thinking involved using a Scrape tool (the "User Analysis" tool) to analyze the portfolio of projects uploaded by a particular community member and generate a visual representation of the blocks used (or not used) in every project.

The visualization for the Scratch community member featured in Figure 11 is shown in Figure 12. Each column represents a project and all of the blocks it contains, while each row represents a specific type of Scratch block. A darker shade indicates more frequent use of a block within the project. The final column identifies blocks that have never been used.

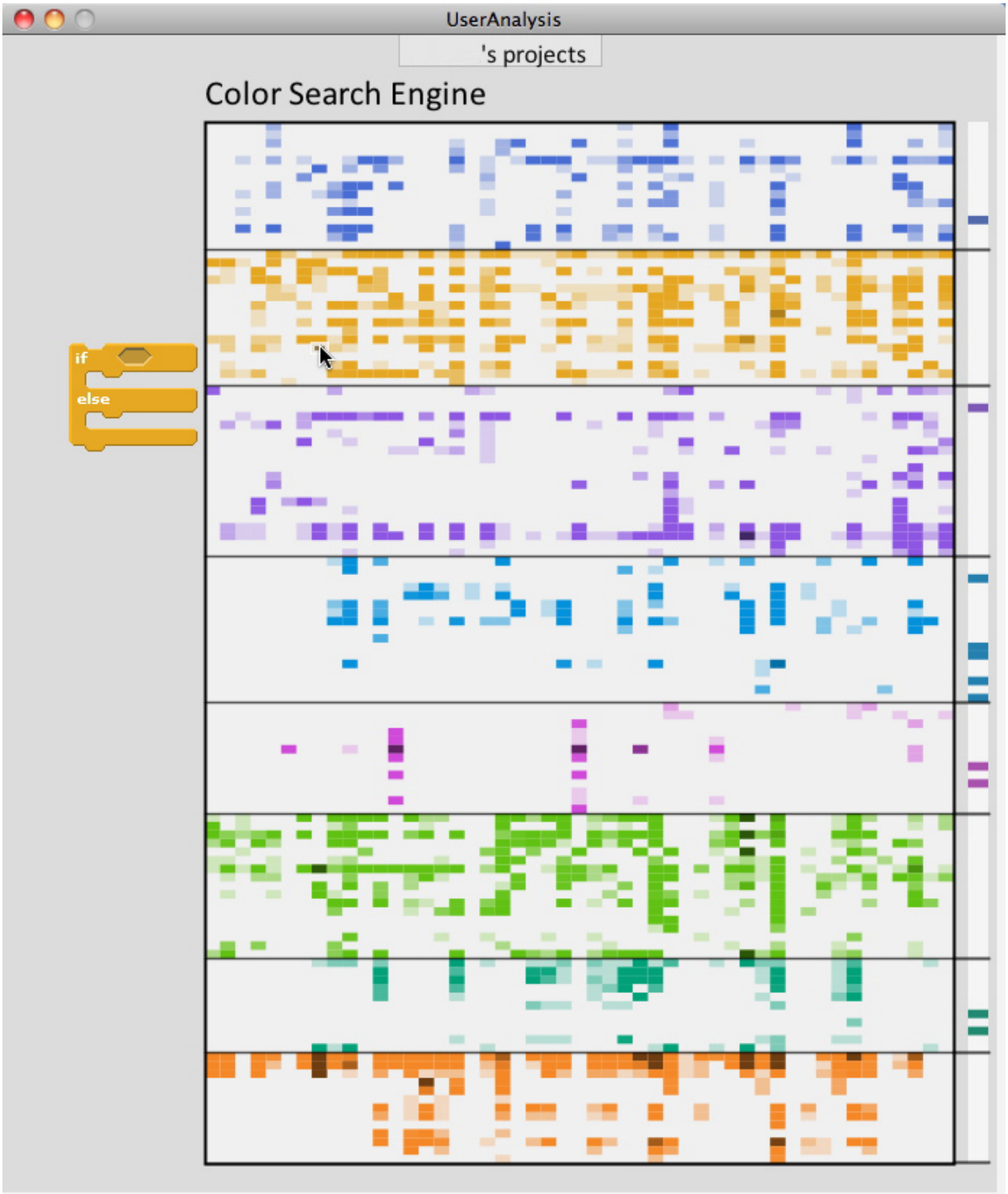

Figure 12. Scrape User Analysis visualization for an experienced Scratcher.

In comparison, Figure 13 is the Scrape User Analysis visualization of a novice Scratcher. This 11-year-old member has been a member for one week and has created 19 projects. But unlike the member in Figure 12, this Scratcher has not experimented with the majority of Scratch blocks.

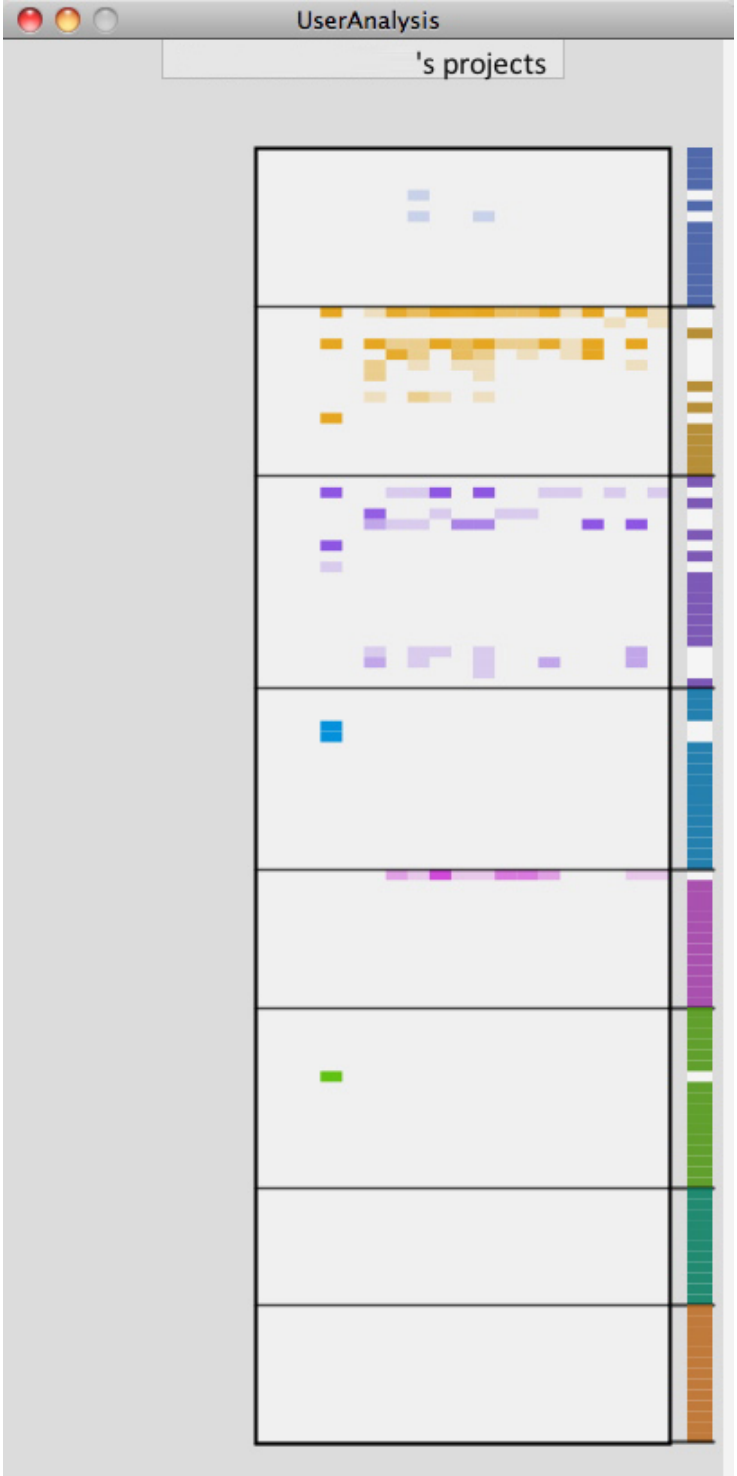

Figure 13. Scrape User Analysis visualization for a novice Scratcher.

Strengths

As described in the first part of the paper, our computational thinking framework maps concepts to particular Scratch blocks. So, the "User Analysis" approach of analyzing a project's blocks provides a record of computational concepts that are being encountered by a Scratcher. We find the formative nature of this assessment particularly appealing. The "User Analysis" tool focuses

on a collection of work over time, which emphasizes the evolving and developing nature of a Scratchers' portfolio rather than, for example, a summative examination of a single final project.

Limitations
Our initial approach of using project content analysis as a means of assessing computational thinking quickly revealed its limitations. This approach is entirely product-oriented, and reveals nothing about the process of developing projects, and, in turn, nothing about the particular computational thinking *practices* that might have been employed. This lack of process information has an impact on assessment of the *concepts*, as it is unknown what the creator was able to do on their own (as opposed to getting help from other people or other projects) and the extent to which they understand the concepts associated with particular blocks (as opposed to concepts they have been "exposed" to).

We learned through interviews and observations that many young people do not post all of their projects to the Scratch online community. In particular, in-progress projects and abandoned projects were often not posted, or were posted to an alternative, test account. These projects could be particularly interesting from a developmental perspective, as they might highlight areas of conceptual confusion or challenge.

Finally, this approach focuses on the blocks in projects, but there are other (less automated) ways of studying a Scratch community member's profile. For example, the genres of projects being generated could be analyzed. Are all of the projects the same type of project (e.g. all games or all stories)? Does the creator demonstrate an ability to develop different genres of projects in this medium? Beyond projects, analysis could be expanded to study a Scratcher's comments. What does the creator say about their own work? What does the creator say in response to others' projects? To what extent are connections being made with other creators?

Approach #2: Artifact-Based Interviews
Studying Scratchers' online project portfolios invited numerous questions – questions we thought would best be explored in conversation with Scratchers directly. Our second approach to assessing the development of computational thinking was an artifact-based interview approach. Over the course of a year, we interviewed 31 Scratchers, who represented a range of ages (8-17), geographic locations (including North America, Europe, Asia), durations of participation (from 1 month to 4 years), technical/aesthetic sophistication (from beginners to experts), and 40% of whom were female (reflecting participation in the online community). The majority of these Scratchers were selected through random sampling, but others were selected after they responded to a community-wide invitation.

Interviews ranged in duration from 60 to 120 minutes. The interview protocol was organized into four major sections:

1. Background
    a. Introduction to Scratch: *How did you find out about Scratch? What is Scratch?*
    b. Current practices: *Where do you use Scratch? What do you do with it? Do other people help you? Do you help other people?*
2. Project Creation

   a. Project framing: *How did you get the idea for your project?*
   b. Project process: *How did you get started making your project? What happened when you got stuck?*
3. Online Community
   a. Introduction to the online community: *What do you do in the online community? What is the Scratch online community?*
   b. Other people, other projects: *How do you find interesting people and interesting projects? How do you interact with other Scratchers?*
4. Looking Forward
   a. Scratch: *What do you dis/like about Scratch? What would you keep, add, change?*
   b. Technology: *What are other tech-related things you like to do?*
   c. Beyond technology: *What are other non-tech-related things you like to do?*

Section 2 (Project Creation) was most significant for assessing computational thinking concepts and practices. We asked the interviewees to select two projects that they would find interesting to discuss. For each project, we started by asking about the history and motivation for the project. Then, we ran the project to see how it worked. We asked the creator to discuss the process of developing the project: how they got started, how the project evolved during development, what was important for them to know in order to make the project, what problems they encountered throughout the process, and how they dealt with those problems. Finally, we ended the discussion about the project with some reflections on the artifact, such as what they were most proud of, what they might want to change, and what surprised them. With this approach, we were able to have detailed discussions with Scratchers about particular programming elements in a project (such as asking how a certain stack of code functions or why a particular block was used), and to develop rich descriptions of their development practices.

This approach highlighted a weakness in our previous blocks-based project portfolio analysis approach. Consider the analysis of a 13-year-old Scratcher who had been using Scratch for 3.5 years and had created 163 projects (Figure 14). From the visualization, we saw that this Scratcher had developed numerous projects, and had experimented with a variety of blocks.

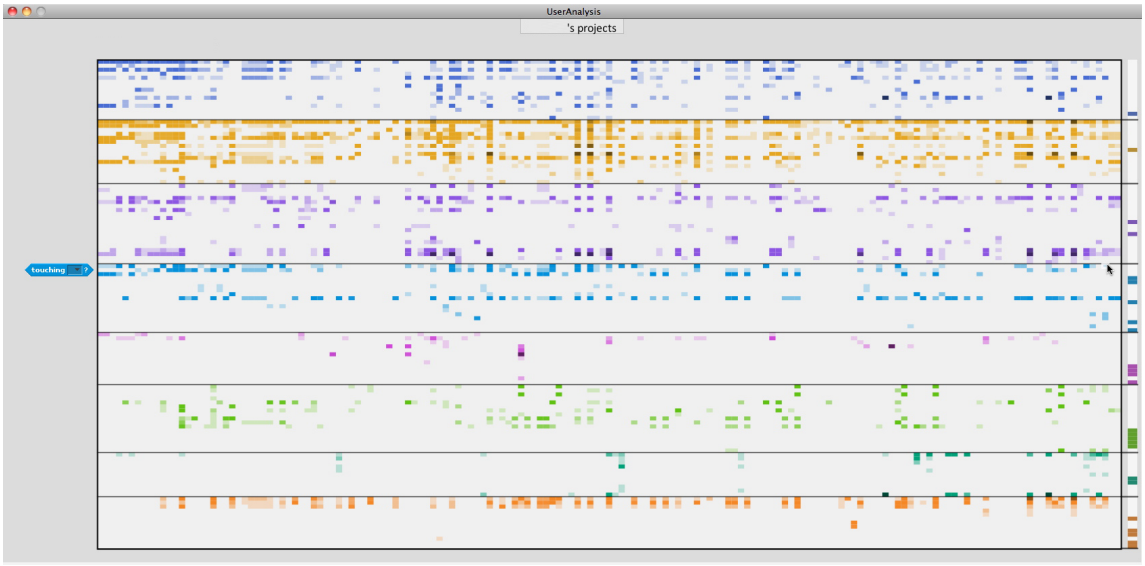

Figure 14. Scrape User Analysis visualization of an apparently-fluent Scratcher.

But in the interview, it became apparent that, despite the Scratcher's apparent fluency, there were significant conceptual gaps. For example, one of the projects selected was a graph, drawn in real-time, that is proportional to the loudness measured by the computer's microphone – the louder the sound, the larger the spike on the graph (Figure 15).

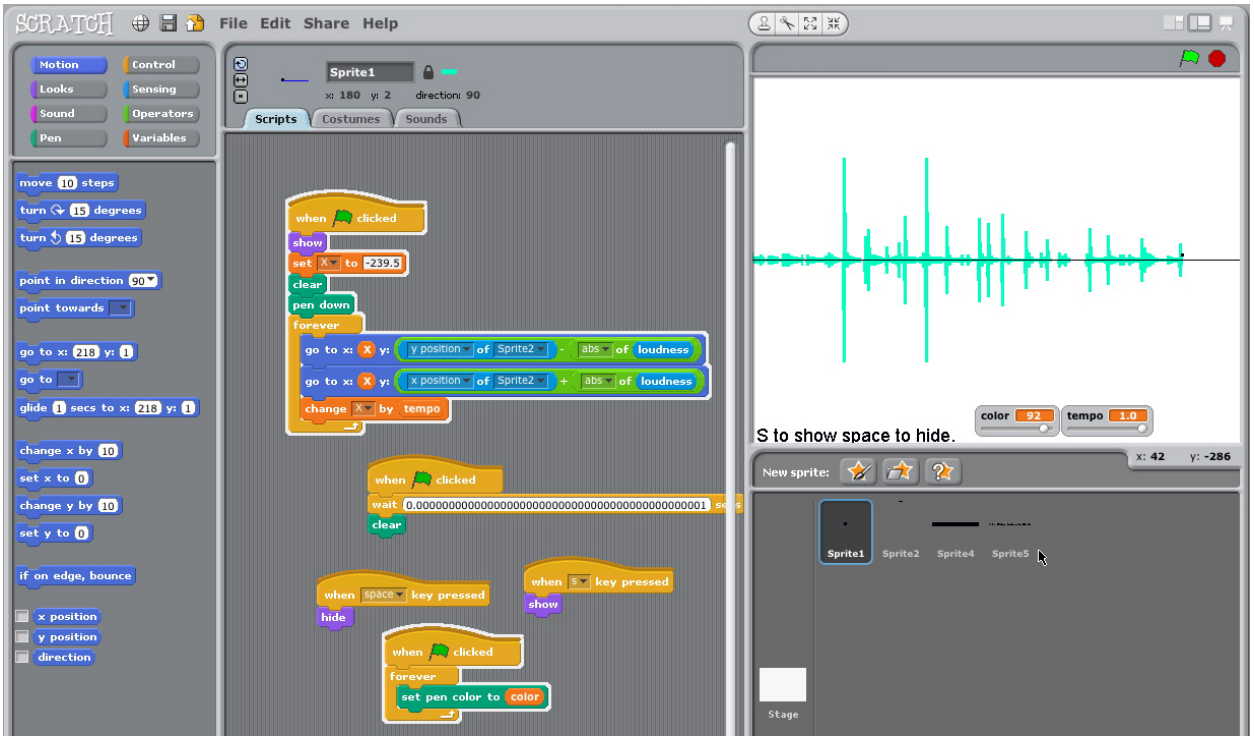

Figure 15. A project created by the Scratcher illustrated in Figure 14.

As we discussed the project and how it works, one of the interviewers wanted to know more about a particular code excerpt (Figure 16). The interviewer asked, "How does this work?" The Scratcher was unable to explain any part of it. The Scratcher explained that they had seen a project like it on the website, downloaded the project to view its code, and had pulled out matching blocks until it somewhat worked the same.

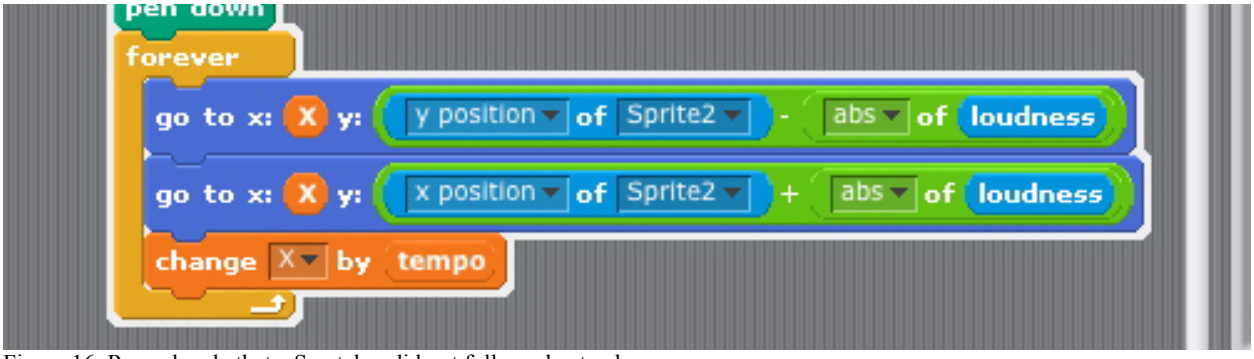

Figure 16. Reused code that a Scratcher did not fully understand.

Strengths

Our exchange with this Scratcher served as a strong reminder that the presence of a code element in a project is not necessarily an indicator that the designer possesses a deep understanding of the

code element, and underscored the strength of the artifact-based interview approach to develop better understandings of a Scratcher's fluency with particular concepts. The concepts are no longer just *there* or *not there*, as in the first approach; a more nuanced characterization emerges, with Scratchers being at different points on a trajectory of understanding. For example, consider a Scratcher who is able to explain what a particular concept or block is, but is unable to meaningfully use it in context. Or a Scratcher who is able to read someone else's code and explain how it functions, but is uncertain how to independently select and employ the same concepts in a new context.

The discussion-based format also enabled an expansion of focus from exclusively product to include process. This gave us opportunities to assess how young people were employing computational thinking practices while developing projects. We analyzed for several indicators of fluency with practices: Were they aware of and able to articulate their design practices and strategies? Did they have a range of strategies in their repertoire? Did the strategies they developed assist them in achieving their design goals?

Limitations
The artifact-based interview approach is time consuming, requiring at least an hour with the Scratcher. Further adding to the time burden, interviews would ideally be repeated at several points over time for a developmental portrait.

Although we were able to discuss the process of developing particular projects, the discussion was limited by what the Scratcher was able to remember and did not typically explore practices in real time. Some Scratchers, when asked to describe a situation in which they "got stuck" while developing their project, would instantaneously respond, "I never got stuck!" For some, this quick initial response reflected a limitation of memory, while for others it reflected a desire to communicate expertise or mastery.

The discussion of process was also constrained by the two projects the Scratcher selected to discuss. When asked why a project had been selected, many Scratchers said that the project was particularly "awesome" or "cool" or "popular" – a project that they were especially proud of. Of course, these projects sometimes included challenges or difficulties, but those challenges or difficulties had invariably been overcome, and were not aspects with which they were actively struggling. There were few cases in which the Scratcher being interviewed asked for help with their projects (although in one interview, the Scratcher began by giving the interviewers a list of Scratch-related concepts to explain.)

Approach #3: Design scenarios
Our third approach to assessment was the development of design scenarios. These scenarios were developed in collaboration with researchers at Education Development Center (EDC) as part of an NSF grant focused on the development of computational thinking through Scratch programming activities. Unlike the other assessment approaches described in this paper, these design scenarios were used exclusively in classroom settings. Researchers from EDC tested this assessment tool with a small number of students in a variety of schools, across grades and disciplines.

We developed three sets of Scratch projects with increasing complexity. Within each set, there were two projects; the projects engaged the same concepts and practices, but had different aesthetics to appeal to different interests. In a series of three interviews, students were presented with the design scenarios, which were framed as projects that were created by another young Scratcher. The students were then asked to select one of the projects from each set, and (1) explain what the selected project does, (2) describe how it could be extended, (3) fix a bug, and (4) remix the project by adding a feature. The combination of these four activities emerged from several independent activities (presentations, critiques, debugging, challenges, and remixing) that we had been experimenting with in workshops for young Scratchers and educators. The genres of projects and sequencing of concepts followed the framing outlined in the Scratch curriculum guide (Brennan, 2011). The projects can be downloaded from http://bit.ly/ScratchDesignScenarios

Set 1: Name and Performance

In the *Name* project, Dean (the project creator) has designed an animated project that features his name. How could we extend this project? Dean wants the N to appear after the A, not at the same time. What is the bug? How do we fix the bug? Dean wants the N to do something interesting (like the other letters), but only when the N is clicked. How do we add this feature?

In the *Performance* project, Keely has designed an animated performance project. How could we extend this project? Keely wants the singer to sing while she is moving, not after. What is the bug? How do we fix the bug? Keely wants each drum to start only if it is clicked. How do we add this feature?

Both of these projects (Figure 17) feature the computational thinking concepts of sequence, loops, parallelism, and events.

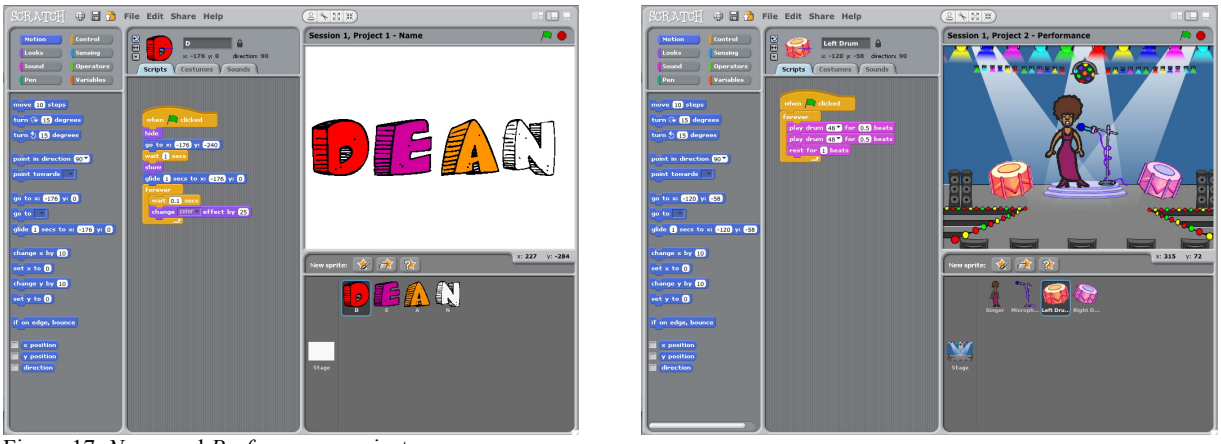

Figure 17. *Name* and *Performance* projects.

Set 2: Underwater Conversation and Sports Scenes

In the *Underwater Conversation* project, Miguel has designed a project that features a conversation between a whale and an octopus. How could we extend this project? Miguel wants the whale to say, "Not much!" after the octopus says, "Hey whale! What's up?" What is the bug? How do we fix the bug? Miguel wants the crab to appear after the whale says, "Not much!" How do we add this feature?

In the *Sports Scenes* project, Gracie has designed a project that helps people learn about sports. How could we extend this project? Gracie wants to show the Tennis background when the Tennis button is clicked. What is the bug? How do we fix the bug? Gracie wants the baseball to appear and say, "Do you know who invented baseball?" when the Baseball button is clicked. How do we add this feature?

Both of these projects (Figure 18) feature the computational thinking concepts of sequence, loops, parallelism, and events.

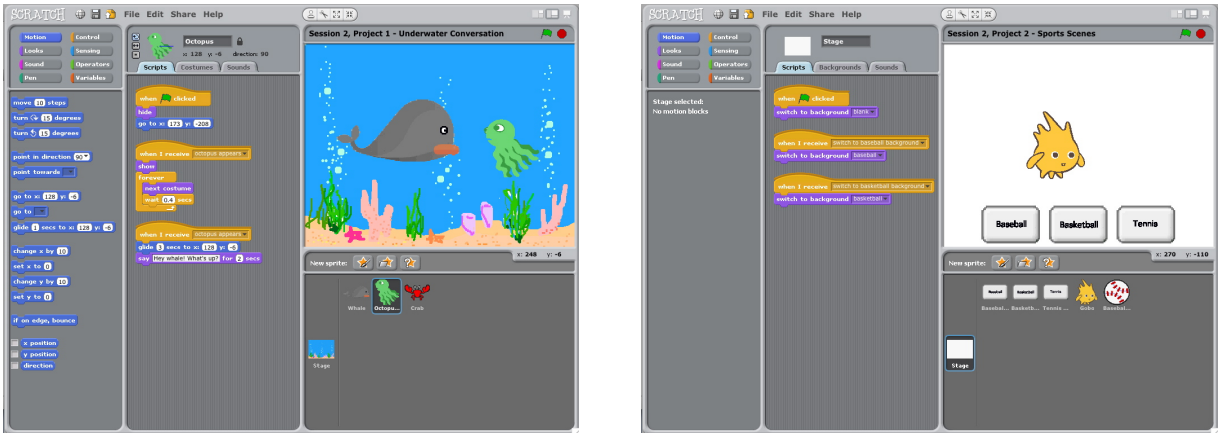

Figure 18. *Underwater Conversation* and *Sports Scenes* projects.

Set 3: Jump and Fruit Quiz

In the *Jump* project, Amaya has designed a three-level obstacle jumping game. How could we extend this project? Amaya wants the game to stop after the three levels are completed. What is the bug? How do we fix the bug? Amaya wants to add another level to her jumping game. How do we add this feature?

In the *Fruit Quiz* project, Mylo designed a fruit quiz with an apple, a banana, and an orange. How could we extend this project? Mylo wants to display "Perfect!" at the end of the project if all three fruit were correctly identified, and "Almost!" otherwise. But the project always displays "Perfect!" What is the bug? How do we fix the bug? Mylo wants to add another fruit to his quiz. How do we add this feature?

Both of these projects (Figure 19) feature the computational thinking concepts of sequence, loops, parallelism, events, conditionals, operators, and data.

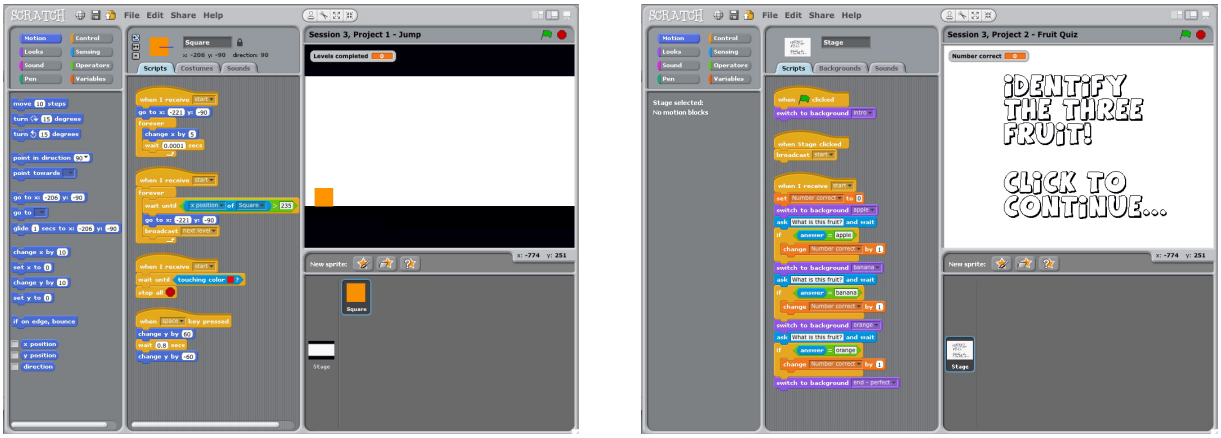

Figure 19. *Jump* and *Fruit Quiz* projects.

Strengths

As an approach to assessment, the design scenarios have three major strengths. First, the scenarios, with their four sets of questions about each project, offer an opportunity to systematically explore different ways of knowing, such as critiquing, extending, debugging, and remixing, as well as fluency with different concepts and practices. Second, the design scenarios are intended to be used at three waypoints over time, an approach that highlights a developmental or formative approach. Finally, the scenarios emphasize process-in-action, rather than process-via-memory. In the artifact-based interview, Scratchers were asked to describe a situation in which they identified and solved (or not) a software bug, relying on their memories or reflections on the experience. In the design scenarios, interviewers (or evaluators or teachers) are able to observe a Scratchers' design practices and debugging strategies, for example.

Limitations

As with artifact-based interviews, design scenarios are time consuming, particularly the debugging and extension activities. How much time should be given to work through the activities? Should assistance be given if they are (consciously or not) stuck or following an unproductive path?

Further, the nature of the questions and the use of externally-selected projects may not connect to personal interests and the learner's sense of intrinsic motivation. Even though the design scenarios are framed as assisting another Scratcher, for some, feelings of helping someone else – particularly if they are unable to develop a response – may be displaced by feelings of judgment or testing.

Connecting Framework to Assessment

Returning to our computational thinking framework, how do these different approaches support the assessment of computational **concepts**, **practices**, and **perspectives**? Table 1 summarizes the strengths and limitations of each approach, as described in the previous section.

Table 1. Strengths and limitations of assessment approaches.

	Concepts	**Practices**	**Perspectives**
Approach #1: Project Analysis	presence of blocks indicates conceptual encounters	N/A	N/A (possibly by extending analysis to include other website data, like comments)
Approach #2: Artifact-Based Interviews	nuances of conceptual understanding, but with limited set of projects	yes, based on own authentic design experiences, but subject to limitations of memory	maybe, but hard to ask directly
Approach #3: Design Scenarios	nuances and range of conceptual understanding, but externally selected projects	yes, in real-time and in a novel situation, but externally selected projects	maybe, but hard to ask directly

In general, we felt that the progression from the first approach to the third approach was productive – mainly by leading to more nuanced understandings of a Scratcher's fluency with computational concepts and having access to richer data about a Scratcher's computational practices. None of the three approaches were particularly effective for understanding changes in computational thinking perspectives. It is challenging to explicitly ask a Scratcher how participation in an activity like programming with Scratch has contributed to a shift in understanding oneself or the world. Social and psychological insights often emerged serendipitously through interview conversations with a Scratcher or with others close to the Scratcher (such as a parent or teacher), as opposed to direct questioning.

Given that no single approach proved sufficient, a combination of approaches could be appropriate. We acknowledge, however, that certain constraints (e.g. time available to conduct assessment, number of learners) make this difficult, if not impossible.

Six Suggestions for Assessing Computational Thinking via Programming

In general, we need to think about how developing as a computational thinker takes place in different contexts, on different timescales, with different motivations, and with different structures and supports – and how these differences lead to different approaches to assessment. Despite variations in learning environments (which may not even include computational thinking as an explicit framework for learning), we end with a set of general suggestions for assessing the learning that takes place when young people engage in programming, which we argue is a valuable setting for developing capacities for computational thinking.

Suggestion #1: Supporting further learning
We believe that the best forms of assessment are those that are useful to the learners. The three approaches to assessment described here only tangentially connect to the learners' interests and goals – the Scrape tool chain is publicly available and could be used by Scratchers to discover new blocks, the interviews are often thought-provoking opportunities for reflection, the design scenarios are engaging intellectual puzzles for some learners – and more could be done to make assessment contextualized and meaningful for the learner. This is an interesting challenge, as what we are most easily able to assess may not be most valuable to the learner. For example, a young person creating an interactive game wants to add a score, but is unaware of the computational concepts of variables and data, and a list of unused variable blocks may be insufficient support for achieving their goal.

Suggestion #2: Incorporating artifacts
Assessments should involve creating and critically examining projects. Individual projects are rich, concrete, and contextualized examples that can be explored and analyzed in a variety of ways. A collection of projects makes assessment even richer, providing an opportunity to see how understanding develops over time.

Suggestion #3: Illuminating processes
We found ourselves limited with the blocks-based analysis in our first assessment approach, and realized that rich conversations about development processes go hand-in-hand with artifacts that have been developed. Focusing on process presents an opportunity to explore the computational thinking that is incompletely represented by blocks: What understanding does the designer have about particular concepts? What practices did they employ? Assessment of process can take multiple forms, and need not involve real-time observation. Young computational creators can document their processes through comments in their code or in project notes, talk about their experiences in presentations, embed audio-recorded descriptions in their Scratch projects, screen record their development process, teach others what they know, or engage in a retrospective or real-time interview. Whatever the form, conversations about their work engage young people in a meta-cognitive activity, encouraging them to think about their thinking, a capacity important to developing as a self-regulating learner.

Suggestion #4: Checking in at multiple waypoints
Computational thinking is not a binary state of *there* or *not there* at a single point in time, and any approach to assessment should strive to describe where a learner has been, is currently, and might go. Adopting a formative approach to assessment involves checking in at multiple points across a computational learning experience, and may also involve checking in during a particular design activity (like checking in with a colleague to discuss progress while creating).

Suggestion #5: Valuing multiple ways of knowing
The intersection of computational thinking concepts and computational thinking practices leads to multiple ways of knowing. It is not enough to be able to define a concept, such as "What is a loop?" Is the learner able to meaningfully put the concept to use in design? Is the learner able to read how others employed the concept and then remix it to the learner's own end? Is the learner able to analyze and critique their own and others' code? Is the learner able to debug problematic code? Assessments should explore these multiple ways of knowing.

Suggestion #6: Including multiple viewpoints
Our second and third approaches to assessment were greatly enriched by moving beyond assessment solely from the researcher's viewpoint. In interviews, for example, new insights about a Scratcher's development were provided by parents or siblings who interjected during conversations. In the Scratch online community, peer feedback and critique is highly valued. Assessment should embrace this multiplicity of viewpoints, engaging self, peer, parent, teacher, and researcher assessments as possible and appropriate.

These suggestions are based on what we saw, analytically, as the most productive components of our three approaches to assessment and on what we have learned through conversations with young Scratchers and Scratch educators. We hope that others will take these suggestions, as well as our three example approaches, and remix them to create new forms of assessment.

Additional Resources

- Scratch curriculum guide
  http://scratched.media.mit.edu/resources/scratch-curriculum-guide-draft
- Computational thinking concepts webinar
  http://scratched.media.mit.edu/resources/computational-thinking-concepts-march-2011-webinar
- Computational thinking practices webinar
  http://scratched.media.mit.edu/resources/computational-thinking-practices-april-2011-webinar
- Computational thinking perspectives webinar
  http://scratched.media.mit.edu/resources/computational-thinking-perspectives-may-2011-webinar
- Assessing computational thinking webinar
  http://scratched.media.mit.edu/resources/assessing-computational-thinking-may-2012-scratched-webinar

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1019396. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Allan, W., Coulter, B., Denner, J., Erickson, J., Lee, I., Malyn-Smith, J., Martin, F. (2010). Computational thinking for youth. *White Paper for the ITEST Small Working Group on Computational Thinking (CT)*.

Bandura, A. (2001). Social cognitive theory: An agentic perspective. *Annual Review of Psychology, 52*, 1-26.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48- 54.

Brennan, K. (2011). *Creative computing: A design-based introduction to computational thinking*. Retrieved May 9, 2012, from http://scratched.media.mit.edu/sites/default/files/CurriculumGuide-v20110923.pdf

Brennan, K., & Resnick, M. (in press). Imagining, creating, playing, sharing, reflecting: How online community supports young people as designers of interactive media. In N. Lavigne & C. Mouza (Eds.), *Emerging technologies for the classroom: A learning sciences perspective*. Springer.

Brennan, K., Resnick, M., & Monroy-Hernandez, A. (2010). Making projects, making friends: Online community as catalyst for interactive media creation. New Directions for Youth Development, 2010(128), 75-83.

Brennan, K., Valverde, A., Prempeh, J., Roque, R. & Chung, M. (2011). More than code: The significance of social interactions in young people's development as interactive media creators. In T. Bastiaens & M. Ebner (Eds.), Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2011 (pp. 2147-2156). Chesapeake, VA: AACE.

Cuny, J., Snyder, L., & Wing, J.M. (2010). *Demystifying computational thinking for non-computer scientists*. Unpublished manuscript in progress, referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

Kafai, Y. B., & Resnick, M. (Eds.). (1996). *Constructionism in practice: Designing, thinking, and learning in a digital world*. Hillsdale, NJ: Erlbaum.

National Academies of Science. (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington DC: National Academies Press.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. Communications of the ACM, 52(11), 60-67.

Wolz, U., Hallberg, C., & Taylor, B. (March, 2011). *Scrape: A tool for visualizing the code of Scratch programs*. Poster presented at the 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX.