

DOI:10.1145/1592761.1592779

“Digital fluency” should mean designing, creating, and remixing, not just browsing, chatting, and interacting.

BY MITCHEL RESNICK, JOHN MALONEY, ANDRÉS MONROY-HERNÁNDEZ, NATALIE RUSK, EVELYN EASTMOND, KAREN BRENNAN, AMON MILLNER, ERIC ROSENBAUM, JAY SILVER, BRIAN SILVERMAN, AND YASMIN KAFAI

Scratch: Programming for All

WHEN MOSHE Y. VARDI, Editor-in-Chief of *Communications*, invited us to submit an article, he recalled how he first learned about Scratch: “A colleague of mine (CS faculty),” he said, “told me how she tried to get her 10-year-old daughter interested in programming, and the only thing that appealed to her was Scratch.”

That’s what we were hoping for when we set out to develop Scratch six years ago. We wanted to develop an approach to programming that would appeal to people who hadn’t previously imagined themselves as programmers. We wanted to make it easy for everyone, of all ages, backgrounds, and interests, to program their own interactive stories, games, animations, and simulations, and share their creations with one another.

Since the public launch in May 2007, the Scratch Web site (<http://scratch.mit.edu>) has become a vibrant online community, with people sharing,

discussing, and remixing one another’s projects. Scratch has been called “the YouTube of interactive media.” Each day, Scratchers from around the world upload more than 1,500 new projects to the site, with source code freely available for sharing and remixing. The site’s collection of projects is wildly diverse, including video games, interactive newsletters, science simulations, virtual tours, birthday cards, animated dance contests, and interactive tutorials, all programmed in Scratch.

The core audience on the site is between the ages of eight and 16 (peaking at 12), though a sizeable group of adults participates as well. As Scratchers program and share interactive projects, they learn important mathematical and computational concepts, as well as how to think creatively, reason systematically, and work collaboratively: all essential skills for the 21st century. Indeed, our primary goal is not to prepare people for careers as professional programmers but to nurture a new generation of creative, systematic thinkers comfortable using programming to express their ideas.

In this article, we discuss the design principles that guided our development of Scratch and our strategies for making programming accessible and engaging for everyone. But first, to give a sense of how Scratch is being used, we describe a series of projects developed by a 13-year-old girl with the Scratch screen name BalaBethany.

BalaBethany enjoys drawing anime characters. So when she started using Scratch, it was natural for her to program animated stories featuring these characters. She began sharing her projects on the Scratch Web site, and other members of the community responded positively, posting glowing comments under her projects (such as “Awesome!” and “OMG I LUV IT!!!!!!”), along with questions about how she achieved certain visual effects (such as “How do you make a sprite look see-through?”). Encouraged, BalaBethany then created and shared new Scratch projects on a regular basis, like episodes in a TV series.

PHOTOGRAPH ILLUSTRATION: SCRATCH PROJECTS FROM TOP LEFT TO RIGHT: SACKBOY2, ATYO-DICKERSON, KURI, NEVIT, SHANNAPAL, SADOWFIRE11, ZADARM03, KGR0DON, SHANESTA, MTEBOM, NATE.

ask What's your name? and wait

say join Hello answer for 2 secs

y position -150

wait until touching > 80

switch to costume jump ▾

say game over





Figure 1. Screenshots from BalaBethany's anime series, contest, and tutorial.

She periodically added new characters to her series and at one point asked why not involve the whole Scratch community in the process? She created and uploaded a new Scratch project that announced a “contest,” asking other community members to design a sister for one of her characters (see Figure 1). The project listed a set of requirements for the new character, including “Must have red or blue hair, please choose” and “Has to have either cat or ram horns, or a combo of both.”

The project received more than 100 comments. One was from a community member who wanted to enter the contest but said she didn’t know how to draw anime characters. So BalaBethany produced another Scratch project,

a step-by-step tutorial, demonstrating a 13-step process for drawing and coloring anime characters.

Over the course of a year, BalaBethany programmed and shared more than 200 Scratch projects, covering a range of project types (stories, contests, tutorials, and more). Her programming and artistic skills progressed, and her projects clearly resonated with the Scratch community, receiving more than 12,000 comments.

Why Programming?

It has become commonplace to refer to young people as “digital natives” due to their apparent fluency with digital technologies.¹⁵ Indeed, many young people are very comfortable sending text messages, playing online games, and browsing the Web. But does that really make them fluent with new technologies? Though they interact with digital media all the time, few are able to create their own games, animations, or simulations. It’s as if they can “read” but not “write.”

As we see it, digital fluency requires not just the ability to chat, browse, and interact but also the ability to design, create, and invent with new media,¹⁶ as BalaBethany did in her projects. To do so, you need to learn some type of programming. The ability to program provides important benefits. For example, it greatly expands the range of what you



can create (and how you can express yourself) with the computer. It also expands the range of what you can learn. In particular, programming supports “computational thinking,” helping you learn important problem-solving and design strategies (such as modularization and iterative design) that carry over to nonprogramming domains.¹⁸ And since programming involves the creation of external representations of your problem-solving processes, programming provides you with opportunities to reflect on your own thinking, even to think about thinking itself.²

Previous Research

When personal computers were first introduced in the late 1970s and 1980s, there was initial enthusiasm for teaching all children how to program. Thousands of schools taught millions of students to write simple programs in Logo or Basic. Seymour Papert’s 1980 book *Mindstorms*¹³ presented Logo as a cornerstone for rethinking approaches to education and learning. Though some children and teachers were energized and transformed by these new possibilities, most schools soon shifted to other uses of computers. Since that time, computers have become pervasive in children’s lives, but few learn to program. Today, most people view computer programming as a narrow, technical activity, appropriate for only



Figure 2. Sample Scratch scripts.

a small segment of the population.

What happened to the initial enthusiasm for introducing programming to children? Why did Logo and other initiatives not live up to their early promise? There were several factors:

- Early programming languages were too difficult to use, and many children simply couldn't master the syntax of programming;

- Programming was often introduced with activities (such as generating lists of prime numbers and making simple line drawings) that were not connected to young people's interests or experiences; and

- Programming was often introduced in contexts where no one could provide guidance when things went wrong—or encourage deeper explorations when things went right.

Papert argued that programming languages should have a “low floor” (easy to get started) and a “high ceiling” (opportunities to create increasingly complex projects over time). In addition, languages need “wide walls” (supporting many different types of projects so people with many different interests and learning styles can all become engaged). Satisfying the triplet of low-floor/high-ceiling/wide-walls hasn't been easy.³

In recent years, new attempts have sought to introduce programming to children and teens.⁷ Some use professional programming languages like Flash/ActionScript; others use new languages (such as Alice⁷ and Squeak Etoys⁵) developed specifically for younger programmers. They have inspired and informed our work on Scratch. But we weren't fully satisfied with the existing options. In particular, we felt it was important to make the floor even lower and the walls even wider while still supporting development of computational thinking.

To achieve these goals, we established three core design principles for Scratch: Make it more tinkerable, more meaningful, and more social than other programming environments. In the following sections, we discuss how each of these principles guided our design of Scratch.

More Tinkerable

Our Lifelong Kindergarten research group at the MIT Media Lab ([\[llk.media.mit.edu\]\(http://llk.media.mit.edu\)\) has worked closely with the Lego Company \(<http://www.lego.com/>\) for many years, helping develop Lego Mindstorms and other robotics kits.¹⁷ We have always been intrigued and inspired by the way children play and build with Lego bricks. Given a box full of them, they immediately start tinkering, snapping together a few bricks, and the emerging structure then gives them new ideas. As they play and build, plans and goals evolve organically, along with the structures and stories.](http://</p>
</div>
<div data-bbox=)

We wanted the process of programming in Scratch to have a similar feel. The Scratch grammar is based on a collection of graphical “programming blocks” children snap together to create programs (see Figure 2). As with Lego bricks, connectors on the blocks suggest how they should be put together. Children can start by simply tinkering with the bricks, snapping them together in different sequences and combinations to see what happens. There is none of the obscure syntax or punctuation of traditional programming languages. The floor is low and the experience playful.

Scratch blocks are shaped to fit together only in ways that make syntactic sense. Control structures (like *for-every* and *repeat*) are C-shaped to suggest that blocks should be placed inside them. Blocks that output values are shaped according to the types of values they return: ovals for numbers and hexagons for Booleans. Conditional blocks (like *if* and *repeat-until*)

have hexagon-shaped voids, indicating a Boolean is required.

The name “Scratch” itself highlights the idea of tinkering, as it comes from the scratching technique used by hip-hop disc jockeys, who tinker with music by spinning vinyl records back and forth with their hands, mixing music clips together in creative ways. In Scratch programming, the activity is similar, mixing graphics, animations, photos, music, and sound.

Scratch is designed to be highly interactive. Just click on a stack of blocks and it starts to execute its code immediately. You can even make changes to a stack as it is running, so it is easy to experiment with new ideas incrementally and iteratively. Want to create parallel threads? Simply create multiple stacks of blocks. Our goal is to make parallel execution as intuitive as sequential execution.

The scripting area in the Scratch interface is intended to be used like a physical desktop (see Figure 3). You can even leave extra blocks or stacks lying around in case you need them later. The implied message is that it's OK to be a little messy and experimental. Most programming languages (and computer science courses) privilege top-down planning over bottom-up tinkering. With Scratch, we want tinkerers to feel just as comfortable as planners.

The emphasis on iterative, incremental design is aligned with our own development style in creating Scratch. We selected Squeak as an implementation language since it is well-suited for

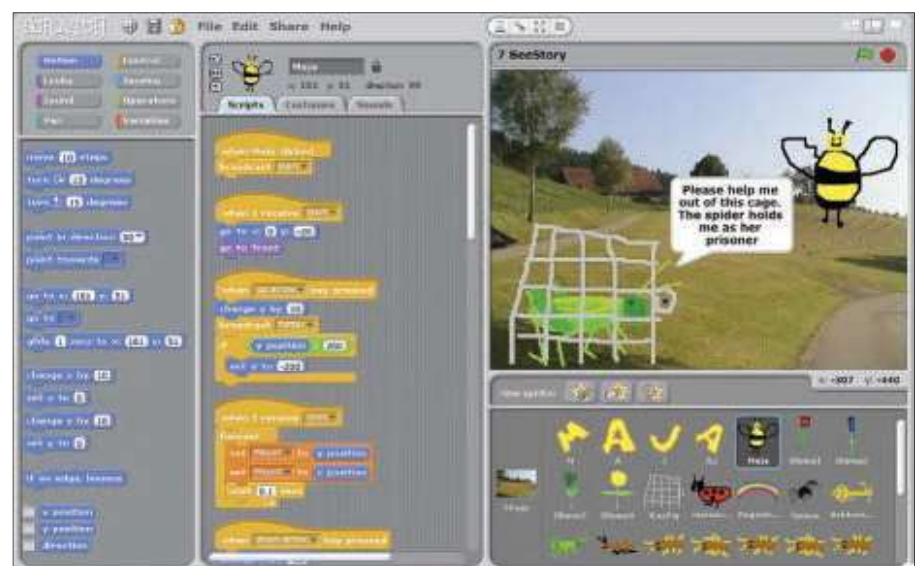


Figure 3. Scratch user interface.

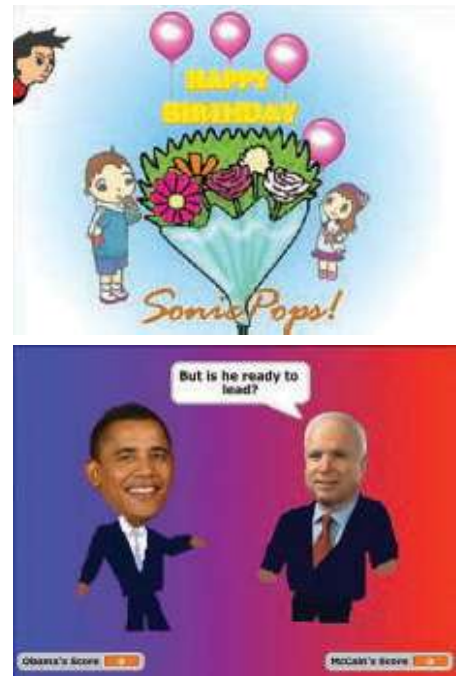


Figure 4. Screenshots from sample Scratch projects.

rapid prototyping and iterative design. Before we launched Scratch in 2007, we continually field-tested prototypes in real-world settings, revising over and over based on feedback and suggestions from the field.⁴

More Meaningful

We know that people learn best, and enjoy most, when working on personally meaningful projects. So in developing Scratch, we put a high priority on two design criteria:

Diversity. Supporting many different types of projects (stories, games, animations, simulations), so people with

widely varying interests are all able to work on projects they care about; and

Personalization. Making it easy for people to personalize their Scratch projects by importing photos and music clips, recording voices, and creating graphics.¹⁴

These priorities influenced many of our design decisions. For example, we decided to focus on 2D images, rather than 3D, since it is much easier for people to create, import, and personalize 2D artwork. While some people might see the 2D style of Scratch projects as somewhat outdated, Scratch projects collectively exhibit a visual diversity and

personalization missing from 3D authoring environments.

The value of personalization is captured nicely in this blog post from a computer scientist who introduced Scratch to his two children: “I have to admit that I initially didn’t get why a kids’ programming language should be so media-centric, but after seeing my kids interact with Scratch it became much more clear to me. One of the nicest things I saw with Scratch was that it personalized the development experience in new ways by making it easy for my kids to add personalized content and actively participate in the development process. Not only could they develop abstract programs to do mindless things with a cat or a box, etc... but they could add their own pictures and their own voices to the Scratch environment, which has given them hours of fun and driven them to learn.”

We continue to be amazed by the diversity of projects that appear on the Scratch Web site. As expected, there are lots of games, ranging from painstakingly recreated versions of favorite video games (such as *Donkey Kong*) to totally original games. But there are many other genres, too (see Figure 4). Some Scratch projects document life experiences (such as a family vacation in Florida); others document imaginary wished-for experiences (such as a trip to meet other Scratchers). Some Scratch

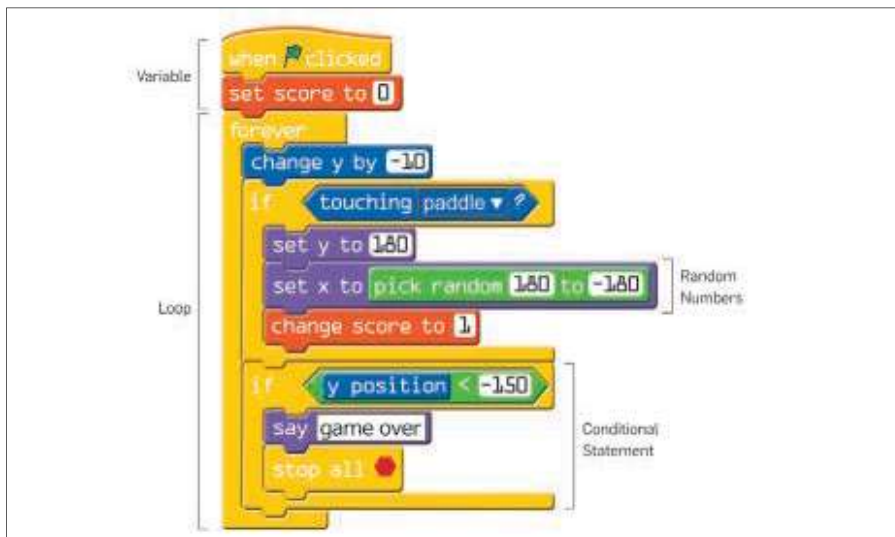


Figure 5. Sample Scratch script (from Pong-like paddle game) highlighting computational and mathematical concepts.

projects (such as birthday cards and messages of appreciation) are intended to cultivate relationships. Others are designed to raise awareness on social issues (such as global warming and animal abuse). During the 2008 U.S. presidential election, a flurry of projects featured Barack Obama and John McCain and later a series of projects promoted members of the Scratch online community for the not-quite-defined position of “President of Scratch.”

Some Scratch projects grow out of school activities. For an Earth-science class, a 13-year-old boy from India created a project in which an animated character travels to the center of the Earth, with a voice-over describing the different layers along the way. As part of a social-studies class, a 14-year-old boy from New Jersey created a simulation of life on the island of Rapa Nui, designed to help others learn about the local culture and economy.

As Scratchers work on personally meaningful projects, we find they are ready and eager to learn important mathematical and computational concepts related to their projects (see Figure 5). Consider Raul, a 13-year-old boy who used Scratch to program an interactive game in his after-school center.⁹ He created the graphics and basic actions for the game but didn’t know how to keep score. So when a researcher on our team visited the center, Raul asked him for help. The researcher showed Raul how to create a variable in Scratch, and Raul immediately saw how he could use it for keeping score. He began playing with the blocks for incrementing variables, then reached out and shook the researcher’s hand, saying “Thank you, thank you, thank you.” The researcher wondered how many eighth-grade algebra teachers get thanked by their students for teaching them about variables?

More Social

Development of the Scratch programming language is tightly coupled with development of the Scratch Web site.¹² For Scratch to succeed, the language needs to be linked to a community where people can support, collaborate, and critique one another and build on one another’s work.¹

The concept of sharing is built into the Scratch user interface, with a prom-

Three core design principles for Scratch: Make it more tinkerable, more meaningful, and more social than other programming environments.

inent “Share” menu and icon at the top of the screen. Click the Share icon and your project is uploaded to the Scratch Web site (see Figure 6) where it is displayed at the top of the page, along with the “Newest Projects.” Once a project is on the Web site, anyone can run it in a browser (using a Java-based player), comment on it, vote for it (by clicking the “Love It?” button), or download it to view and revise the scripts. (All projects shared on the site are covered by Creative Commons license.)

In the 27 months following the Scratch launch, more than 500,000 projects were shared on the Scratch Web site. For many Scratchers, the opportunity to put their projects in front of a large audience—and receive feedback and advice from other Scratchers—is strong motivation. The large library of projects on the site also serves as inspiration. By exploring projects there, Scratchers get ideas for new projects and learn new programming techniques. Marvin Minsky once said that Logo had a great grammar but not much literature.¹¹ Whereas young writers are often inspired by reading great works of literature, there was no analogous library of great Logo projects to inspire young programmers. The Scratch Web site is the beginning of a “literature” for Scratch.

The site is also fertile ground for collaboration. Community members are constantly borrowing, adapting, and building on one another’s ideas, images, and programs. Over 15% of the projects there are remixes of other projects on the site. For example, there are dozens of versions of the game Tetris, as Scratchers continue to add new features and try to improve gameplay. There are also dozens of dress-up-doll projects, petitions, and contests, all adapted from previous Scratch projects.

At first, some Scratchers were upset when their projects were remixed, complaining that others were “stealing” from them. That led to discussions on the Web site’s forums about the value of sharing and the ideas behind open source communities. Our goal is to create a culture in which Scratchers feel proud, not upset, when their projects are adapted and remixed by others. We have continually added new features to the site to support and encourage this mind-set. Now, when someone remixes

a project, the site automatically adds a link back to the original project, so the original author gets credit. Also, each project includes links to its “derivatives” (projects remixed from it), and the “Top Remixed” projects are featured prominently on the Scratch homepage.

Some projects focus on the site itself, providing reviews and analyses of other projects there. One early example was called SNN, for Scratch News Network, featuring the Scratch cat (the default character in Scratch) delivering news about the Scratch community, much like a CNN anchor. At first, we saw it as a “simulated newscast” but then realized it was a real newscast, providing news of interest to a real community—the Scratch online community. The SNN project inspired others, leading to a proliferation of online newsletters, magazines, and TV shows, all programmed in Scratch, reporting on the Scratch community.

Other Scratchers formed online “companies,” working together to create projects that their individual members could not have produced on their own. One company got its start when a 15-year-old girl from England, with screen name BeeBop, created a project full of animated sprites and encouraged others to use them in their projects or place special requests for custom-made sprites. She was setting up a no-fee consulting business. A 10-year-old girl, also from England, with screen name MusicalMoon, liked BeeBop’s animations and asked if she’d be willing to create a background for one of her projects. This collaboration gave rise to Mesh Inc., a self-proclaimed “miniature company” to produce “top quality games” in Scratch. A few days later, a 14-year-old boy from New Jersey, screen name Hobbit, discovered the Mesh Inc. gallery and offered his services, saying, “I’m a fairly good programmer, and I could help with debugging and stuff.” Later, an 11-year-old boy from Ireland, with screen name Marty, was added to the Mesh Inc. staff due to his expertise in scrolling backgrounds.

Such collaborations open opportunities for many different types of learning. Here’s how a 13-year-old girl from California, who started a Scratch company called Blue Elk Productions, described her experience:

“What is fun about Scratch and



The Scratch Web site has become a vibrant online community, with people sharing, discussing, and remixing one another’s projects.



about organizing a company to write games together is that I’ve made a lot of friends and learned lots of new things. I’ve learned a lot about different kinds of programming by looking at other games with interesting effects, downloading them, and looking at and modifying the scripts and sprites. I really like programming! Also, when I started with Scratch I didn’t think I was a very good artist. But since then, just by looking at other people’s art projects, asking them questions, and practicing drawing using programs like Photoshop and the Scratch paint editor, I’ve gotten a lot better at art... Another thing I’ve learned while organizing Blue Elk is how to help keep a group of people motivated and working together... 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.”

To encourage international sharing and collaboration, we’ve placed a high priority on translating Scratch into multiple languages. We created an infrastructure that allows the Scratch programming blocks to be translated into any language with any character set. A global network of volunteers has provided translations for more than 40 languages. Children around the world now share Scratch projects with one another, each viewing the Scratch programming blocks in their own language.

Future Directions

A growing number of K–12 schools around the world, and even some universities (including Harvard and the University of California, Berkeley),⁸ use Scratch as a first step into programming. A natural question is What comes next? In the Scratch discussion forums, there are ongoing debates about what programming language should be used after Scratch. We receive many requests to add more advanced features to Scratch (such as object inheritance and recursive list structures), hoping that Scratch itself could be the “next step.”

We plan to keep our primary focus on lowering the floor and widening the walls, not raising the ceiling. For some Scratchers, especially those who want to pursue a career in programming or com-

puter science, it is important to move on to other languages. But for many other Scratchers, who see programming as a medium for expression, not a path toward a career, Scratch is sufficient for their needs. With Scratch, they can continue to experiment with new forms of self-expression, producing a diverse range of projects while deepening their understanding of a core set of computational ideas. A little bit of programming goes a long way.


As we develop future versions, our goal is to make Scratch even more tinkerable, meaningful, and social. With our Scratch Sensor Board (http://info.scratch.mit.edu/Sensor_Boards), people can create Scratch projects that sense and react to events in the physical world. We are also developing a version of Scratch that runs on mobile devices and a Web-based version that enables people to access online data and program online activities.

Probably the biggest challenges for Scratch are not technological but cultural and educational.¹⁰ Scratch has been a

success among early adopters, but we need to provide better educational support for it to spread more broadly. We recently launched a new online community, called Scratch-Ed (<http://scratched.media.mit.edu>), where educators share their ideas, experiences, and lesson plans for Scratch. More broadly, there needs to be a shift in how people think about programming, and about computers in general. We need to expand the notion of “digital fluency” to include designing and creating, not just browsing and interacting. Only then will initiatives like Scratch have a chance to live up to their full potential.

Acknowledgments

Many people have contributed to the development of Scratch and even more to the ideas underlying Scratch. We’d like to thank friends and former members of the Lifelong Kindergarten group who have worked on Scratch, especially Tammy Stern, Dave Feinberg, Han Xu, Margarita Dekoli, Leo Burd, Oren Zuckerman, Nick Bushak, and Paula Bonta.

We are grateful to Kylie Peppler, Grace Chui, and other members of Yasmin Kafai’s research team, who conducted and participated in field studies in Scratch’s early development. Scratch was deeply influenced and inspired by the work of Seymour Papert and Alan Kay. We appreciate financial support from the National Science Foundation (grant ITR-0325828), Microsoft, Intel Foundation, Nokia, and MIT Media Lab research consortia. The names of all children mentioned here are pseudonyms. 

References

1. Bransford, J., Brown, A., and Cocking, R. *How People Learn: Mind, Brain, Experience, and School*. National Academies Press, Washington, D.C., 2000.
2. diSessa, A. *Changing Minds: Computers, Learning, and Literacy*. MIT Press, Cambridge, MA, 2000.
3. Guzdial, M. Programming environments for novices. In *Computer Science Education Research*, S. Fincher and M. Petre, Eds. Taylor & Francis, Abingdon, U.K., 2004, 127–154.
4. Kafai, Y., Peppler, K., and Chiu, G. High-tech programmers in low-income communities: Seeding reform in a community technology center. In *Communities and Technologies*, C. Steinfield, B. Pentland, M. Ackerman, and N. Contractor, Eds. Springer, New York, 2007, 545–564.
5. Kay, A. Squeak etoys, children, and learning; <http://www.squeakland.org/resources/articles>.
6. Kelleher, C. and Pausch, R. Using storytelling to motivate programming. *Commun. ACM* 50, 7 (July 2007), 58–64.
7. Kelleher, C. and Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys* 37, 2 (June 2005), 83–137.
8. Malan, D. and Leitner, H. Scratch for budding computer scientists. *ACM SIGCSE Bulletin* 39, 1 (Mar. 2007), 223–227.
9. Maloney, J., Peppler, K., Kafai, Y., Resnick, M., and Rusk, N. Programming by choice: Urban youth learning programming with Scratch. *ACM SIGCSE Bulletin* 40, 1 (Mar. 2008), 367–371.
10. Margolis, J. *Stuck in the Shallow End: Education, Race, and Computing*. MIT Press, Cambridge, MA, 2008.
11. Minsky, M. *Introduction to LogoWorks*. In *LogoWorks: Challenging Programs in Logo*, C. Solomon, M. Minsky, and B. Harvey, Eds. McGraw-Hill, New York, 1986.
12. Monroy-Hernández, A. and Resnick, M. Empowering kids to create and share programmable media. *Interactions* 15, 2 (Mar.–Apr. 2008), 50–53.
13. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, 1980.
14. Peppler, K. and Kafai, Y. From SuperGoo to Scratch: Exploring creative media production in informal learning. *Journal on Learning, Media, and Technology* 32, 7 (2007), 149–166.
15. Prensky, M. Digital natives, digital immigrants. *On the Horizon* 9, 5 (Oct. 2001), 1–6.
16. Resnick, M. Sowing the seeds for a more creative society. *Learning and Leading with Technology* (Dec. 2007), 18–22.
17. Resnick, M. Behavior construction kits. *Commun. ACM* 36, 7 (July 1993), 64–71.
18. Wing, J. Computational thinking. *Commun. ACM* 49, 3 (Mar. 2006), 33–35.

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, and Jay Silver are all researchers and members of the Scratch Team (<http://scratch.mit.edu>) at the Media Laboratory of the Massachusetts Institute of Technology, Cambridge, MA. **Brian Silverman** is president of the Playful Invention Company, Montreal, Quebec, Canada. **Yasmin Kafai** is a professor in the Graduate School of Education of the University of Pennsylvania, Philadelphia, PA.



Figure 6. Scratch Web site.