Some Reflections on Designing Construction Kits for Kids

Mitchel Resnick
MIT Media Lab
20 Ames Street
Cambridge, MA 02139 USA
+1 617 253 9783
mres@media.mit.edu

Brian Silverman
Playful Invention Company
2075 University Street, Suite 1208
Montreal, Quebec H3A 2L1 Canada
+1 514 282 4994
brian@playfulinvention.com

ABSTRACT
In this paper, we present ten guiding principles for designing construction kits for kids, informed by our experiences over the past two decades:

* Design for Designers
* Low Floor and Wide Walls
* Make Powerful Ideas Salient – Not Forced
* Support Many Paths, Many Styles
* Make it as Simple as Possible – and Maybe Even Simpler
* Choose Black Boxes Carefully
* A Little Bit of Programming Goes a Long Way
* Give People What They Want – Not What They Ask For
* Invent Things That You Would Want to Use Yourself
* Iterate, Iterate – then Iterate Again

While these principles apply especially to the development of construction kits, we believe that they could be useful for everyone who designs new technologies for kids.

Keywords
Design, metadesign, construction kits, education, learning

INTRODUCTION
Over the past 20 years, the two of us have worked together on the design of a variety of new technologies for kids. Most of our creations can be viewed as "construction kits" – that is, systems that engage kids in designing and creating things, sometimes on the screen, sometimes in the physical world, sometimes both. Kids around the world have used our construction kits to create their own animated stories, simulations, robotic constructions, interactive sculptures, scientific instruments, and multimedia presentations.

In designing these construction kits, we have had several overarching goals: to help kids become more fluent and expressive with new technologies (and with "old" technologies too); to help them explore important concepts (often in the domains of mathematics, science, and engineering) through their expressive activities; and, most broadly, to help them become better learners.

In this paper, we make a first attempt to articulate our "guiding principles" for designing construction kits for kids. These principles have emerged through our collaboration, with a large number of colleagues, in the development of many different construction kits (including LEGO/Logo [5], Microworlds, StarLogo [6], Scratch [10], Programmable Bricks [7], and others). When we design new technologies, we do not explicitly refer to this list of principles, as if checking off items on an agenda. Rather, the principles are always sitting in the back of our minds, subtly (and sometimes not so subtly) informing each decision we make.

These guiding principles are influenced (some might say biased) by our focus on construction kits. But we feel that the principles could be useful for everyone who designs new technologies for kids – and, perhaps, those who design for adults too.

1. DESIGN FOR DESIGNERS
Probably the most important unifying thread in all of our projects is our emphasis on "learning through designing." Seymour Papert has served as our most important intellectual mentor, and we have been deeply influenced by his Constructionist approach to learning and education [3]. Like Papert, we believe that the best learning experiences, for most people, come when they are actively engaged in designing and creating things, especially things that are meaningful to them or others around them.

If our goal is to engage kids in meaningful design experiences, then it makes sense for us to design for designers – that is, to design things that will enable kids to design things. We see the traditional LEGO construction kit as a model for what we are trying to achieve with new technologies. We provide kids with a simple set of parts (in the spirit of LEGO bricks) that they can use to design and create a diverse collection of constructions. But our new construction kits allow new types of creations: while kids use traditional LEGO bricks primarily for static, structural creations (such as houses and castles), they use our new building blocks for dynamic, interactive creations (such as animations in a virtual world or kinetic sculptures in the physical world).

The analogy with LEGO kits also suggests an important counter-example. In recent years, a growing number of LEGO kits highlight a specific construction (such as a Star Wars spaceship or a Harry Potter castle), with many specialized pieces. Although it is possible to use these kits to create a variety of constructions, many kids build the model suggested on the package, or perhaps slight variants, and nothing more. This type of activity might qualify as

"hands-on learning" or "learning-by-doing," but it is not what we mean by "learning-through-designing." Our goal is to develop technologies that not only engage kids in constructing things, but also encourage (and support) them to explore the ideas underlying their constructions.

2. LOW FLOOR AND WIDE WALLS

The Logo programming language is often described as having a low floor and high ceiling: it is easy for novices to get started (low floor) and possible for experts to work on increasingly sophisticated projects (high ceiling).

In our own work (especially in recent years), we have put less emphasis on high ceilings and more emphasis on what might be called "wide walls." That is, we have tried to design technologies that support and suggest a wide range of different explorations. When kids use our Programmable LEGO Bricks, for instance, they can create anything from a robotic creature to a "smart" house to an interactive sculpture to a musical instrument. We want kids to work on projects that grow out of their own interests and passions – which means that our technologies need to support a wide range of different types of projects.

When we evaluate the use of our construction kits, we consider diversity of outcomes as an indicator of success. If the creations from a class of students are all similar to one another, we feel that something has gone wrong. And if, after finishing one project, a student feels that s/he is "done" with the construction kit, again we feel as if we have failed.

We see our construction kits as defining a space to explore, not a collection of specific activities. And our hope is that kids will continually surprise themselves (and surprise us too) as they explore the space of possibilities. When we created Programmable Bricks, we didn't imagine that kids would use them to measure their speed on rollerblades, or to create a machine for polishing and buffing their fingernails.

To support and encourage this diversity, we explicitly include elements and features that can be used in many different ways. The design challenge is to develop features that are specific enough so that kids can quickly understand how to use them (low floor), but general enough so that kids can continue to find new ways to use them (wide walls).

3. MAKE POWERFUL IDEAS SALIENT – NOT FORCED

In a paper [4] written 20 years after the publication of his landmark book *Mindstorms: Children, Computers, and Powerful Ideas* [2], Papert noted that educators had reacted to the book "as if it were about *children and computers*, as if the third term [*powerful ideas*] was there as a sound bite." In fact, Papert had intended the idea of "powerful ideas" to be at the core of his book – and his work.

What is a powerful idea? In *Mindstorms*, Papert describes powerful ideas as ideas that "can be used as tools to think with over a lifetime." He points to the idea of "feedback" as an example: you can use it to understand many different types of phenomena in the world, not only in engineering, but also in biology and social sciences. Powerful ideas are ideas with leverage: they help you make sense of the world.

In designing construction kits, one of our primary goals is to help kids explore and understand powerful ideas. We have found that trying to teach powerful ideas directly is not very effective. Rather, our strategy is to provide opportunities for kids to encounter and use powerful ideas as a natural part of design experiences.

In developing StarLogo modeling software, for example, we designed the objects and commands so that kids naturally encounter the idea of "emergence" in the process of creating models. If they write rules for cars on a highway, they naturally observe how traffic jams emerge from the interactions among the cars. If they write rules for birds flying in the sky, they naturally observe how flocks emerge from interactions among the birds.

We view StarLogo as a "microworld" for exploring the idea of emergence. Similarly, the original Logo turtle served as a microworld for exploring ideas of differential geometry, and Programmable Bricks serve as a microworld for exploring the idea of feedback. Creating new microworlds is not easy. In a successful microworld, different kids engage in different design activities (e.g., one creates a bird flock, another a traffic jam), but all encounter and use the same underlying ideas as a natural and integral part of the design process. That's very different from traditional educational applications, in which all kids typically work on the same activity (e.g., solving a specific puzzle) to learn a particular idea.

4. SUPPORT MANY PATHS, MANY STYLES

When we were testing an early version of our LEGO/Logo technology, we worked with a fourth-grade class in which the students wanted to build an amusement park. One group of students decided to create a merry-go-round. They carefully drew up plans, built the mechanisms, then wrote a program to make the ride spin round-and-round whenever someone pressed a touch sensor. Within a couple hours, their merry-go-round was working.

Another group of students decided to build a Ferris wheel. But before the ride was working, they put it aside and started building a refreshment stand next to the Ferris wheel. We were concerned: the refreshment stand did not have any motor or sensors or programming. We worried that the students would miss out on some of the powerful ideas underlying the LEGO/Logo activity. But we didn't interfere. After finishing the refreshment stand, the group built a wall around the amusement park, created a parking lot, and added lots of little LEGO people walking into the park. Then, finally, they went back and finished their Ferris wheel.

These two groups represent two very different styles of playing, designing, and thinking. Sherry Turkle [13, 14] has

described these styles as “hard” (the first group) and “soft” (the second). In another classification [12], the two styles are described as “patterners” and “dramatists.”

In designing new technologies, we put a high priority on supporting users with all different styles – hards as well as softs, patterners as well as dramatists. We pay special attention to make sure that our technologies are accessible and appealing to the softs (and dramatists), since we feel that math and science activities have traditionally been biased in favor of the hards (and patterners), and we want to work affirmatively to close the gap.

5. MAKE IT AS SIMPLE AS POSSIBLE – AND MAYBE EVEN SIMPLER

In some ways, this guideline seems obvious. Who wants needless complication? But there is no doubt that technology-based products have become more and more complex. One reason is “creeping featurism”: advances in technology make it possible to add new features, so each new generation of products has more and more features.

We yearn for a return to the clean simplicity of the Macintosh of the 1980s. We see a role for complexity: we make use of ever-more complex technologies, and we want to help users accomplish complex tasks. But we want the user experience to be simple. We try to develop systems that offer the simplest ways to do the most complex things.

We have found that reducing the number of features often improves the user experience. What initially seems like a constraint or limitation can, in fact, foster new forms of creativity.

In the mid-1990s, for example, we had developed a Programmable Brick that was roughly the size of a child’s juice box. It could control four motors and receive inputs from six sensors. For a sponsor event at the Media Lab, we wanted to create some interactive decorations for the tables. We didn’t need all of the capabilities of the Programmable Brick, so we quickly developed a smaller, scaled-down version, roughly the size of a matchbox car. It could control only two motors with inputs from only two sensors. We expected it to be a short-lived project; we “knew” that most users would want more motors and more sensors. But once we had developed the scaled-down version, which we called a Cricket, people kept finding more and more creative applications for it, in spite of (or perhaps because of?) its apparent limitations [9]. Over time, we shifted our research effort, making the Cricket the centerpiece of our new construction kits. Even though the original Programmable Brick was better suited for certain projects, the simplicity of the Cricket won out.

6. CHOOSE BLACK BOXES CAREFULLY

In designing a construction kit, one of the most important decisions is the choice of the basic building blocks of the kit. This choice determines, to a large extent, what ideas users can explore with the kit – and what ideas remain hidden from view.

When kids build robotic devices with our Programmable Bricks, for instance, they learn about mechanisms and gearing, and they learn about feedback and control. But they generally don’t learn about the inner workings of motors. The motor remains a black box. If you wanted to help kids learn how motors work, you should design a construction kit with lower-level building blocks, so that kids could build their own motors.

Similarly, the choice of the basic “building blocks” in a programming language determines what kids are likely learn as they use the language. When kids put together Logo commands like **forward** and **right** into instructions like **repeat 4 [forward 50 right 90]** (to make a square) or **repeat 360 [forward 1 right 1]** (to make a circle), they gain a better understanding of many important mathematical and geometric concepts. But the primitive command **forward** is still a black box. Each time the turtle moves, the computer must calculate new x and y positions from the original x and y positions using trigonometric calculations. These calculations are hidden from the user. If the goal of the construction kit were to help kids learn these types of trigonometric calculations, then the turtle would be a bad black box. But by hiding these calculations inside a black box, the turtle frees the user to experiment and explore other mathematical and geometric ideas.

We faced a similar choice when we were developing the programming language for our Cricket programmable brick. We needed a new command for controlling the color of the LEDs that plug into the Cricket. At a low-level, the Cricket needs to provide the LED with three inputs for the red, green, and blue components of the color. So, at first, we provided users with a **setcolor** command with three numeric inputs, to give them direct control over the color. But kids found it difficult to use this command for the types of activities that they wanted to do. For example, they wanted the color of LED to change based on the current reading from a temperature sensor. If the temperature sensor reported a low value, they wanted the LED to turn blue; as the temperature increased, they wanted the color of the LED to move through the spectrum, turning red at high temperatures. It is very difficult to program this behavior using a three-input **setcolor** command. So we created a simpler **setcolor** command with just a single input that ranges from 0 to 100 (the same range as the readings from the temperature sensor). Kids could use this new command to program the desired behavior with the simple instruction **forever [setcolor temperature]**.

In short, we found that the best way to deal with three dimensions was to throw away two of them. (Make it as simple as possible – and then even simpler!) If our primary goal were to help kids learn about red-green-blue composition of light, the single-input **setcolor** command would be a bad choice. But we have found that the single-input **setcolor** encourages and supports much great exploration of color effects than the three-input version.

7. A LITTLE BIT OF PROGRAMMING GOES A LONG WAY

Programming languages are the construction kits of the computational world. When kids learn to program, it extends the range of what they can design, create, and invent with the computer. Moreover, it provides them with experience in using and manipulating formal systems – experience that is important not only in computer science but also in many other domains (from mathematics to grammar to law).

But the history of introducing computer programming to kids is a mixed success. When personal computers first moved into schools in the early 1980s, programming (often with Logo or Basic) was one of the primary activities – and, indeed, one of the main rationales for buying the computers in the first place. Over the past 20 years, however, the role of programming has steadily diminished in educational uses of computers, even as computers have proliferated in schools. Many people now view computer programming as a narrow, technical activity, too difficult for the masses, appropriate only for the small segment of the population who choose it as a career path.

We continue to believe in the value of everyone learning to program, but we are also well aware of the difficulties of learning to program. Many beginning programmers hit a plateau, able to write simple programs, but unable to go further. We have found that it is difficult to help kids get beyond this plateau. But, over the years, we have begun to realize that being "stuck" on the plateau is not such a big problem: kids can learn a great deal, and benefit a great deal, while they are on the plateau. We have shifted our efforts, trying to leverage what kids can do well, rather than focusing on what they can't. Kids generally have little difficulty learning to use imperative (action-oriented) commands (like **forward** and **on**), simple control structures (like **repeat**), basic conditionals, and simple procedural abstraction. So we have been developing programming languages and contexts that enable kids to do a lot with those basic elements.

We attribute the success of our Programmable Bricks to the fact that kids can accomplish a lot with a little. Kids can engage in interesting design projects (and important learning experiences) with very simple programs, controlling lights and motors, triggered by inputs from sensors. Our new Scratch programming language has similar qualities, enabling kids to manipulate rich media (sounds, music, animations) with simple combinations of commands.

8. GIVE PEOPLE WHAT THEY WANT – NOT WHAT THEY ASK FOR

All good designers want to understand their users, in order to design products well-matched to the needs and interests of their users. Many design teams invest considerable time interviewing users or talking with focus groups, asking users for feedback and suggestions on features and capabilities. But is asking questions directly to users really the best way to understand what they want?

We don't think so. We have found that user suggestions are usually not very helpful. In some cases, users ask for impractical or infeasible features. When we were designing the first Programmable Bricks, for instance, elementary-school students recommended that we design the Bricks so that they could fly. In other cases, users ask for only incremental changes, not aware of the possibilities of radical change. With early versions of Logo software in the 1980s, users often suggested new ways for the turtle to draw – but they never suggested the addition of paint tools.

Another problem is that users often ask for more flexibility than is really needed or desirable. When we showed an early version of our Scratch software to potential users, they suggested that all of the window panes in the interface should be movable and resizable. We implemented a new version with that type of flexibility, but users weren't happy with that either. What we needed to do, it turns out, was to fine-tune the parameters (i.e., adjust the sizes of the panes), not provide full flexibility. Often, designs with well-chosen parameters are more successful than designs with fully-adjustable parameters. We are all in favor of giving control to users – but only where control will really make a difference in their experiences.

Rather than asking users what they want, we have found it more productive to observe users interacting with our prototypes, and try to infer what they want (and don't want) from their actions. Often, their actions speak louder than their words. It is usually easy to see when users get frustrated, even if they don't articulate their frustration. When we observe users repeatedly making the same "mistake" with a prototype, we sometimes are able to revise the software so that it behaves in the way that users had expected. In early versions of our LogoBlocks graphical programming language, for example, users often tried to get rid of blocks by dragging them from the workspace back onto the palette. Initially, we did not want to allow this method for deleting blocks, since we worried that users would too often delete blocks by mistake. But after seeing users attempt this action repeatedly, we changed the software so that it behaved as users expected and wanted.

9. INVENT THINGS THAT YOU WOULD WANT TO USE YOURSELF

At first blush, this guideline might seem incredibly egocentric. And, indeed, there is a danger of over-generalizing from your own personal tastes and interests. But we have found that we do a much better job as designers when we really enjoy using the systems that we are building. And we have found that it is, in fact, possible to design systems that are interesting and enjoyable for both kids and ourselves.

We feel that this approach is, ultimately, more respectful to kids. Why should we impose on kids systems that we don't

enjoy using ourselves? For example, we are generally skeptical of educational software that, in an effort to encourage kids to reflect on their actions, requires that kids annotate each action that they take. We wouldn't want to do that with the software that we use, so why should we require it of kids?

There is an additional, perhaps less obvious, reason why we try to invent things that we enjoy using ourselves. The technologies that we develop can not succeed on their own. As kids use our technologies, they require support from teachers, parents, and mentors. We aim to build not only new technologies, but also communities of people who can help kids learn with those new technologies. And we have found that it is easiest to build those communities if everyone involved (adults as well as kids) enjoy using the technologies. In New York, for example, groups of MIT alumni have been volunteering their time to help kids at Computer Clubhouses [8] learn to use our Programmable Bricks. The MIT alumni are motivated, in part, by a desire to help youth in low-income communities. But there is no doubt that they are also motivated by their own desire to build robots.

10. ITERATE, ITERATE – THEN ITERATE AGAIN

In designing our construction kits, we put a high priority on "tinkerability" – we want to encourage kids to mess with the materials, to try out multiple alternatives, to shift directions in the middle of the process, to take things apart and create new versions. Kids learn new lessons with each iteration.

Just as we want kids to iterate their designs, we apply the same principle to ourselves. In developing new technologies, we have found that we never get things quite right on the first try. We are constantly critiquing, adjusting, modifying, revising. The ability to develop rapid prototypes is critically important in this process. We find that storyboards are not enough; we want functioning prototypes. Initial prototypes don't need to work perfectly, just well enough for us (and our users) to play with, to experiment with, to talk about.

In his book *Serious Play*[11], Michael Schrage argues that prototypes are especially helpful as conversation starters, to catalyze discussions among designers and potential users. We agree. We find that our best conversations (and our best ideas) happen when we start to play with new prototypes – and observe users playing with the prototypes. Almost as soon as we start to play with (and talk about) one prototype, we start to think about building the next.

This process requires both the right tools (to support rapid development of new prototypes) and the right mindset (to be willing to throw out a prototype soon after creating it). Too often, the software-development community seems to follow a paradigm of: plan ahead, design carefully, then implement once. We much prefer the paradigm proposed by our colleague John Maeda [1]: imagine, realize, critique, reflect, iterate.

Of course, design principles should be subject to this same process. The ten principles discussed in this paper have already gone through multiple iterations – and we expect that we will continue to iterate them in the future.

ACKNOWLEDGMENTS

We want to give special thanks to Seymour Papert and Alan Kay, who inspired us to begin developing new technologies for kids, and who continue to provoke us with their ideas. We also want to thank the many people who have collaborated with us on the design of new construction kits (and who, implicitly, contributed to the guiding principles discussed in this paper), particularly Andy Begel, Robbie Berg, Paula Bonta, Rick Borovoy, John Maloney, Fred Martin, Bakhtiar Mikhak, Steve Ocko, and Natalie Rusk. Members of the Lifelong Kindergarten group at the MIT Media Lab provided very useful feedback on an earlier draft of this paper. We are grateful to the LEGO Company (particularly Kjeld Kirk Kristiansen and Erik Hansen) for wonderful collaboration and financial support for more than 20 years. The National Science Foundation (grants CDA-9616444, ESI-0087813, and ITR-0325828), the Media Lab's Things That Think and Digital Life consortia, and the Center for Bits and Atoms (NSF CCR-0122419) all provided financial support for the research described in this paper.

REFERENCES

1. Maeda, J. (2000). *Maeda@Media*. Rizzoli Publications. New York.
2. Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books. New York.
3. Papert, S. (1993). *The Children's Machine: Rethinking School in the Age of the Computer*. Basic Books. New York.
4. Papert, S. (2000). What's the big idea: Towards a pedagogy of idea power. *IBM Systems Journal*, vol. 39, no. 3-4.
5. Resnick, M. (1993). Behavior Construction Kits. *Communications of the ACM*, vol. 36, no. 7, pp. 64-71 (July 1993).
6. Resnick, M. (1994). *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. MIT Press. Cambridge, MA.
7. Resnick, M., Martin, F., Sargent, R., and Silverman, B. (1996). Programmable Bricks: Toys to Think With. *IBM Systems Journal*, vol. 35, no. 3-4, pp. 443-452.
8. Resnick, M., Rusk, N., and Cooke, S. (1998). The Computer Clubhouse: Technological Fluency in the Inner City. In Schon, D., Sanyal, B., and Mitchell, W. (eds.), *High Technology and Low-Income Communities*, pp. 266-286. MIT Press. Cambridge, MA.

9. Resnick, M., Berg, R., and Eisenberg, M. (2000). Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation. *Journal of the Learning Sciences*, vol. 9, no. 1, pp. 7-30.
10. Resnick, M., Kafai, Y., Maeda, J., et al. (2003). A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers in Economically-Disadvantaged Communities. Proposal to the National Science Foundation (funded 2003-2007). http://www.media.mit.edu/~mres/papers/scratch.pdf
11. Schrage, M. (1999). *Serious Play*. Harvard Business School Press. Cambridge, MA.
12. Shotwell, J., Wolf, D., and Gardner, H. (1979). Exploring Early Symbolization. In B. Sutton-Smith (ed.), *Play and Learning*.
13. Turkle, S., & Papert, S. (1990). Epistemological Pluralism. *Signs*, vol. 16, no. 1
14. Turkle, S. (1995). *Life on the Screen*. Simon & Schuster. New York.