Foundations and Trends® in Human-Computer
Interaction
Vol. 9, No. 1 (2015) 1–68
© 2015 P. Blikstein
DOI: 10.1561/1100000057

Computationally Enhanced Toolkits for Children: Historical Review and a Framework for Future Design[1]

Paulo Blikstein
Stanford University, USA
paulob@stanford.edu

[1]This work is supported by the Stanford's Lemann Center for Educational Entrepreneurship and Innovation in Brazil.

Contents

Abstract

Robotics toolkits and physical computing devices have been used in educational settings for many decades. Based on a techno-historical analysis of the development of 30 years of development of these devices, this monograph examines their design principles and presents a framework for the analysis and future design, based on the analytic construct of “selective exposure,” which examines what is foregrounded or backgrounded in hardware and software design. Selective exposure has two sub-dimensions: usability, which examines how the material communicates rules for its use, and power, which looks at how cognitive and physical operations are mapped to each other, and how the design can make these connections more explicit. I show how these dimensions crucially impact what children can achieve with these materials, and make the case for the design of toolkits in synchrony with the childŠs developmental trajectory.

P. Blikstein. *Computationally Enhanced Toolkits for Children: Historical Review and a Framework for Future Design.* Foundations and Trends® in Human-Computer Interaction, vol. 9, no. 1, pp. 1–68, 2015.
DOI: 10.1561/1100000057.

1 Introduction

Archeologists can reconstruct a dinosaur from fragments of a bone, and biologists can infer the Earth's temperature millions of years ago by examining fragments of fossil DNA. Technology historians have also looked at details of simple machines over the millennia as a proxy for the technological level of different civilizations. Semiotics, a "science of detectives," infers larger meaning by looking at details in language, gesture, or prosody Blikstein [1993]. When we do not have access to the entire object, but need to understand the beast, we create indirect ways to complete the puzzle. In this monograph, my goal is to historically and technologically analyze physical computing devices designed for children, derive categories of design decisions, and create theoretical and design frameworks to guide designers and researchers. This is timely, given the growing presence of these devices in formal and informal education.

The presence of several types of physical computing and robotics devices in educational settings is attributable to many research and design initiatives of the past 30 years. However, although the design of such devices has evolved significantly and their popularity has grown wildly, there is little research that examines this technology taking into account their history and the theoretical underpinning that guided their design.

But before delving into these frameworks and devices, it is crucial to understand a few of the educational ideas that guided the pioneers in this field. As we will see throughout this monograph, much of the inspiration and early work came from Seymour Papert's research group at the MIT Media Laboratory. Before coming to MIT, Papert had worked with Jean Piaget, who was the proponent of Constructivism, a very influential theory of human cognition and development. One of the important ideas in Piaget's model is that for a child to abandon a current theory about the world, it takes more than simply being exposed to a better one. The new theory has to emerge from students' complex experiences and actions in the world. Papert added to this theory the idea that this happens more robustly if the learner is engaged in building a public, shareable "object," such as a robot or a computer program [Papert, 1980] — and called his new variation "Constructionism." In other words, Papert was very concerned with not only how to promote sophisticated ways for children to interact with the world (for new knowledge to emerge) but also in making sure that they had at their disposal rich materials and toolkits to build those sharable objects. Therefore, much of Papert's group work was about theorizing about how to create toolkits, programming languages, and other materials for children.

Papert opens his most influential book, Mindstorms [Papert, 1980], with an essay about the "gears of his childhood," in which he talks about how his own experience playing with gears and differentials as a young child generated a deep affective connection with multiplication tables, equations, and mathematics in general: "By the time I had made a mental gear model of the relation between x and y, figuring how many teeth each gear needed, the equation had become a comfortable friend." Papert's computational toolkits ultimately intend to create these same connections in new domains such as engineering, robotics, and cybernetics:

> "A modern-day Montessori might propose, if convinced by my story, to create a gear set for children. But to hope for

> this would be to miss the essence of the story. I fell in love with the gears. This is something that cannot be reduced to purely "cognitive" terms. Something very personal happened. [...] My thesis could be summarized as: What the gears cannot do the computer might. The computer is the Proteus of machines. Its essence is its universality, its power to simulate. Because it can take on a thousand forms and can serve a thousand functions, it can appeal to a thousand tastes. This book is [my attempt] to turn computers into instruments flexible enough so that many children can each create for themselves something like what the gears were for me." [Papert, 1980]

One contribution of the current work is to identify whether and in what ways, and along what dimensions, designers of physical computing toolkits can use the principles of Constructionism in their own work.

I start the monograph by reviewing the history of microcontrollers and robotics in education, comment on their design principles and forms of interaction, and propose a set of analytic constructs to interpret design decisions. Using these constructs, I more deeply analyze representative examples to understand their affordances and usability, and finally I propose principles for theoretically-guided design. The theory I propose addresses three under-researched issues in the literature and design about the design of robotic toolkits:

(1) What are the levels of abstraction exposed to students and what interactions do they afford?

(2) What are the direct connections between specific design decisions and the learning goals intended for each toolkit?

(3) How should toolkits be considered as part of a larger developmental trajectory?

To address these questions, I first divide the history of these technologies into five generations, spanning 30 years of research and development. I then propose a categorization based on the design commitments and principles of these five generations of devices, employing the analytic construct of "*selective exposure*," which examines what is foregrounded or backgrounded in hardware and software design (this construct will be fully explained in Section 4). Selective exposure has two sub-dimensions. Selective exposure for *usability* examines how the material communicates rules for its use — in other words, how the design embeds error correction schemes (for example, self-correcting polarity errors). Selective exposure for *power* looks at how cognitive and physical operations are mapped to each other, and how the design can make these connections more explicit so that users can exploit the full potential of the toolkits (for example, parts can be designed to explicitly show a hierarchy). Thus, selective exposure for usability guarantees a low-threshold for users to quickly start building, and selective exposure for power assures a high-ceiling by indicating through the design the more complex possibilities offered by the toolkit. Finally, I propose the idea of *selective unveiling*, a design principle which advocates the progressive exposure of layers of abstraction in synchrony with the developmental trajectory of children.

I hope that this discussion might move forward a research agenda that exposes children to the powerful ideas in several disciplines through physical computing, leading to better, more theoretically-informed designs.

2

The Story of Microcontrollers in Education: Five Generations

Since often generations of physical computing platforms developed in parallel, I will have to go back and forth in time a few times to describe each of them. The reader will notice that there is a chronological overlap in the story of these platforms, but hopefully by the end of the narrative the “big picture” will be clear.

Tim McNerney was one of the first to narrate the early chapters of this story [McNerney, 2004], which as many in the field of technologies for education, starts with the Logo programming language. Since the late 1960s, researchers have shown that not all programming languages are created equal Papert [1980]. Logo has had a significant impact on K-12 education because it was the first expressive, student-centered educational technology to be widely adopted by the mainstream educational system, and it was created according to theoretically-inspired design principles. Based on these principles, Papert and collaborators [Papert et al., 2001] made a strong case for why the BASIC programming language — then the de facto standard in personal computers — was not an appropriate design, and why there was a need for languages designed specifically for children. They argued that *media matters*: in other words, the particular properties of the constructive building blocks offered to children limit or enhance what they can build, create, and ultimately learn. In particular, they made an important distinction between *understanding the inner workings of a technology* and the content that we want children to *learn through the use of that technology.* For example, Papert was interested in Logo as a way for children to learn the powerful ideas at work in mathematics and computer science (for example, differential geometry and recursion) and not how the computer's memory was being managed by the operating system, or how the transistors were wired inside the microprocessor. The history of Logo and, more recently, of the Scratch[1] and NetLogo[2] programming environments shows how such design principles are crucial for a powerful, sustained, and deep engagement by children with technological tools that continues beyond the novelty effect.

The development of programming languages for children soon inspired the creation of programmable tangibles that would bring coding to the physical world. These developments have occurred in five waves or generations:

(1) The first generation of tangible physical computing devices emerged in the 1980s and early 1990s with the development of the LEGO/Logo platform and the many generations of "programmable bricks" by researchers at the Massachusetts Institute of Technology's Media Lab (for example, Crickets, Lego/LOGO, Lego Mindstorms).

(2) The second generation of devices was developed in the late 1990s and extended the capabilities of these early platforms by including new types of sensors, actuators, and ways to interact with computers, as well as programmable boards targeted to hobbyists and interaction designers (for example, BASIC Stamp, Wiring, Arduino).

(3) The third generation of these devices was developed in the early years of the 21st Century and sought not simply to extend the capabilities of earlier platforms, but additionally placed a

[1]http://scratch.mit.edu
[2]http://ccl.northwestern.edu/netlogo/

particular emphasis on creating devices that would broaden participation in computing and allow users to access new domains of knowledge. Platforms developed during this period were specifically designed to target new classes of users, such as very young children, non-technical designers, and children in the developing world. Other platforms used a modular approach to design as a way to open up possibilities for exploring complex concepts in mathematics and science.

(4) The latter half of the 2000s saw a fourth generation of devices that brought new form factors, new architectures, and new industrial design, embedding computational capacity at the level of the components, enhancing the capabilities of older platforms and further broadening the reach of physical computing to new audiences (for example, Topobo, Cubelets).

(5) A fifth generation of devices started to gain traction with the launch of the Raspberry Pi board in 2012 — a full computer packed into a credit-card sized board for the same price as a regular microcontroller-based board, opening up new possibilities for physical computing in education (for example, Raspberry Pi, Beagle Board).

2.1 The first generation: Pioneers of physical computing (LEGO/Logo, Braitenberg Bricks, and Programmable Bricks)

One of the very early pioneers of physical computing for children was Radia Perlman, who went on to become a very well-known software engineer and inventor. While still in her early twenties, from 1974 to 1976, Radia worked with Seymour Papert at the MIT Artificial Intelligence Lab and created what is believed to be the first system that enabled children to program with physical blocks: TORTIS [Perlman, 1974] (Toddler's Own Recursive Turtle Interpreter System.) Radia was able to get children as young as 3 1/2 years old to program using her system, which had blocks to give robots direct commands, and a "slot

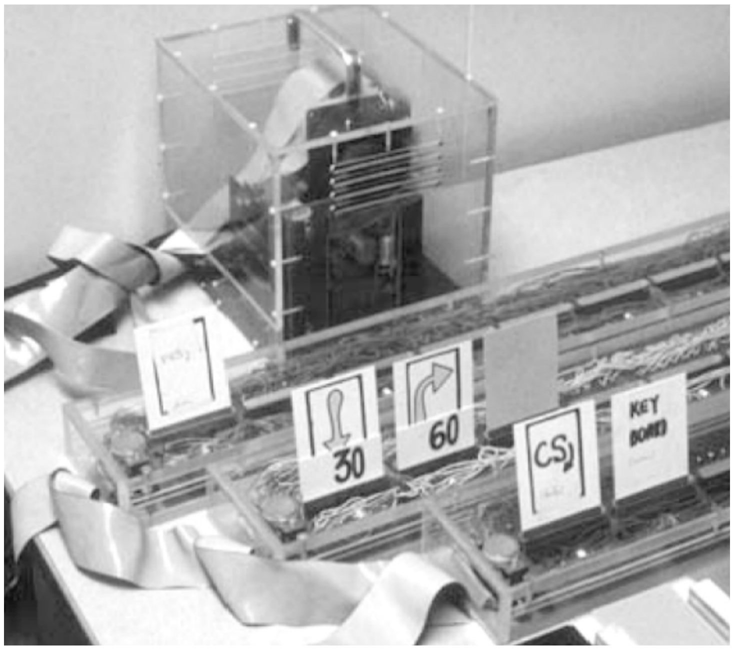


Figure 2.1: Radia Perlman's TORTIS "slot machine" — the part of her system which enable children to write procedures.

machine" (Figure 2.1) that allowed children to write, store and execute procedures. Around the same time, two other systems were being developed at MIT: FASTR [Goldenberg, 1974] and TEACH [Solomon and Papert, 1975], which used single keystrokes to program instead of making children type the full text — however, their tangible components were much less developed.

After a few years, Mitchel Resnick, Fred Martin, and Stephen Ocko began pioneering work on LEGO/Logo began in the 1980s, at the MIT Media Lab. LEGO/Logo (see Figure 2.2) was a computer-based learning platform that combined LEGO construction with the Logo programming language. Children built machines out of traditional LEGO pieces as well as new bricks designed specifically for the Logo platform, which included gears, motors, and sensors. They could then program their constructions. Much of this work was also inspired by Martin's

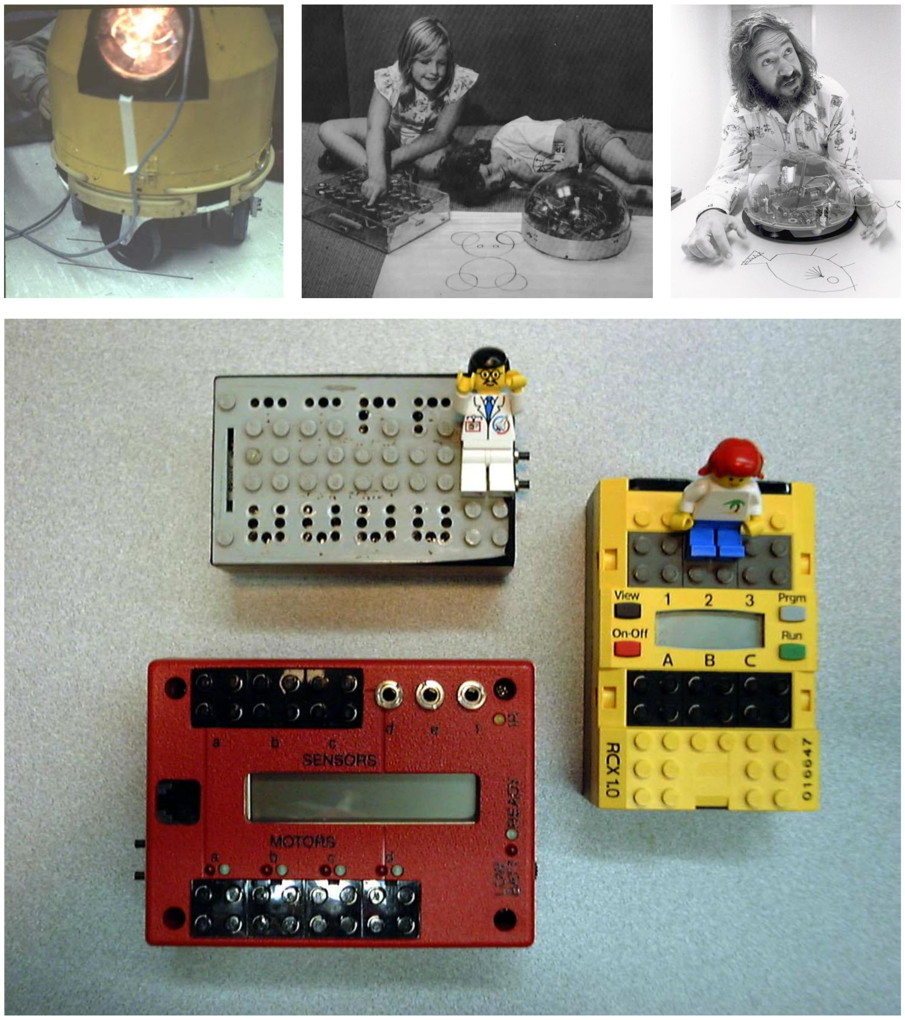


Figure 2.2: The first programmable turtles (top left and top center, circa early 1970s), Seymour Papert and one programmable turtle (top right, 1980s), and the first prototypes of the MIT Programmable Bricks (bottom): counterclockwise, MIT Logo Brick (1987), MIT Red Brick (1995), LEGO® RCX™ Brick (1998) [Martin, 1988].

work with robotics competitions at MIT, which pioneered these types of competitions in engineering schools [Martin, 1988]. At the time, LEGO/Logo represented a return to Logo's roots. Logo was used originally to program a robotic turtle, but the second generation of Logo environments took advantage of the advent of personal computing and shifted to screen turtles from the previous mechanical version.

The physical turtle inspired the development of Logo — the story came full circle with the development of programmable tangibles — LEGO/Logo brought the physical turtle back to tangible form [Resnick et al., 1996, Resnick and Ocko, 1991].

The MIT team ran several successful workshops in the early 1990s and attributed the platform's success to the fact that it put children in control, offered multiple paths to learning, and encouraged the building of a sense of community. However, there were still some important limitations, including the fact that LEGO/Logo creations had to be tethered to personal computers in order for them to work, a serious issue when children wanted to create mobile creatures — in other words, there was not yet an autonomous Lego robotics brick which could store and execute programs.

Limitations aside, the platform enabled children to learn powerful ideas through design and about design [Resnick et al., 1996]: In LEGO/Logo, children were "actively involved in creating and constructing meaningful products" [Resnick and Ocko, 1991], and learned about mathematical and scientific ideas as well as the design process itself [Shaffer and Resnick, 1999]. LEGO/Logo was later released as a commercial product through the LEGO Group and enjoyed tremendous success, even though it was only sold to schools. In the mid-1990s, the platform was being used in more than a dozen countries, including 15,000 elementary and middle schools in the United States [Resnick and Ocko, 1991]. This success inspired further work on the first generation of computationally enhanced construction kits. The earliest extensions of the LEGO/Logo platform addressed a major limitation: cables. A new version, the "Logo Brick," used a 6502 processor, the same that powered Apple II computers one decade earlier. Another innovation, the "Braitenberg Brick" system, developed primarily by

Fred Martin, represented a serious conceptual modification. Martin's system included LEGO bricks with embedded electronics, instead of relying on specially designed electronic bricks used in conjunction with traditional LEGO bricks. The resulting system consisted of a set of "low-level logic bricks" [Resnick et al., 1996] that children could wire together to create different behaviors. The limitation here was that each brick had a dedicated function, so many of them were needed for more complex projects.

Another design, the Programmable Brick, overcame this limitation by embedding a fully programmable computer into a LEGO brick [Resnick et al., 1996]. This design also took advantage of key technological advances; for example, it had faster processing power, more memory, and a variety of input–output possibilities. It also supported infrared communications, had an LCD display, and included buttons for basic operations such as selecting and running programs. It is significant that the Programmable Brick Project found itself at the convergence of two trends that were prominent during the early 1990s in research on computers and children: constructionism and ubiquitous computing. This convergence would have some key design implications. It is important to note that the dominant paradigm in computing up until that time was that "computing" takes places in front of a computer. However, the goal of ubiquitous computing was to spread computation throughout environments and to embed it in all types of objects and artifacts. The Programmable Brick extended the idea of distributing computational power but, for the first time, aimed the work explicitly at children. Studies with children at the time revealed three broad categories of application: children could use the Programmable Brick to make their environments come alive, to program autonomous creatures, and to conduct new types of scientific experiments [Resnick et al., 1996]. The Programmable Brick's proponents claimed that it was not just a "thing that thought" — it actually acted as a "thing to think with."

This historical account is important for few reasons. First, it shows the intellectual roots of the first generation of devices. These designers were deeply embedded in a constructivist/constructionist/educational research culture, so their main motivation was to give powerful expressive tools to children to see how they would use them. In the best tradition of constructivism and developmental psychology, these scholars were not interested in turning children into engineering prodigies or in increasing enrollment in engineering schools, or even in preparing students for careers in STEM. Rather, they were interested in seeing how these new tools would change how all children expressed their ideas [Granott, 1991], and not only the more technically inclined.

2.2 The second generation: Conquering the World (Crickets, Programmable Bricks, and BASIC Stamp)

2.2.1 The MIT Cricket and its first descendants

The work that took place in the 1980s and early 1990s set the stage for the flurry of "things to think with" developed in the later half of the 1990s. Many of these products were updates and extensions of earlier models. For example, LEGO Mindstorms (Figure 2.3) was introduced in 1998 with improvements that made it well suited for those who

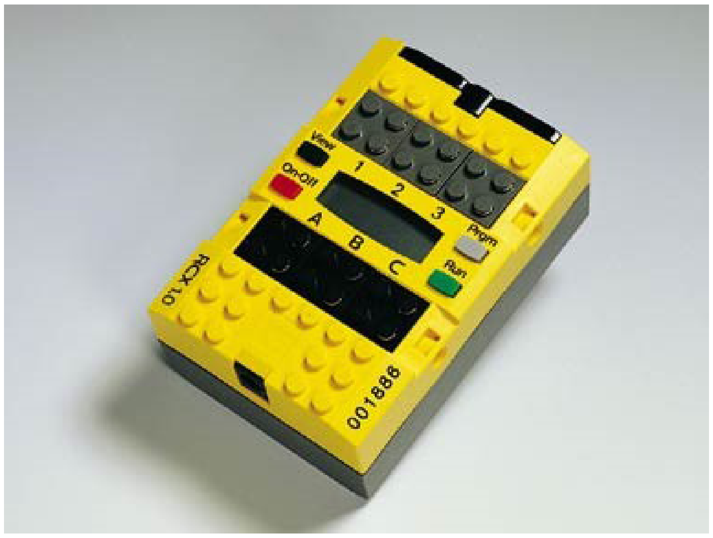


Figure 2.3: LEGO RCX Brick.

wanted to build mobile robots, although not, at that time, for those who wished to create artistic objects involving light, sound, and music.

The team also worked on other devices as well. More specifically, they designed a new class of toys that would "expand the range of concepts kids [could] explore through direct manipulation" [Resnick et al., 1998]. By embedding computation into traditional toys like blocks, beads, balls, and badges, they attempted to leverage children's familiarity with those toys to introduce a new capability that would expose them to new ideas. Unlike the earlier LEGO/Logo implementation, these toys were autonomous. At the time, digital manipulatives were not a novel idea, but this new class of toys did allow children to explore new concepts such as dynamic systems [Resnick et al., 1998].[3]

The Cricket platform (Figure 2.4), a variation of the Programmable Brick, emerged in the late 1990s. It was a general purpose, handheld,

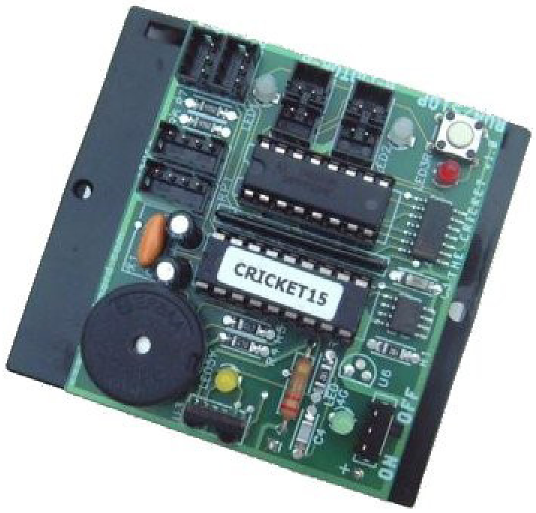


Figure 2.4: The MIT Cricket.

[3]Resnick et al. define digital manipulatives as "computationally-enhanced versions of traditional children's toys [...] these new manipulatives — with computational power embedded inside — are designed to expand the range of concepts that children can explore through direct manipulation, enabling children to learn concepts that were previously considered too advanced." [Resnick et al., 1998].

low-cost, fully programmable computer designed specifically for use in science and engineering education, and influenced by a number of related traditions, including research on home science, design education, microcomputer-based lab activities, and children's programming [Resnick et al., 2000]. It had two input and two output ports, and communicated with the computer through an infrared base. All connectors were polarized, so that children could not connect them in the wrong way. Also in the tradition of the LEGO Brick, the Cricket had on-board motor drivers and batteries, making it an all-in-one solution for sensing and robotics. The sensors and motors that came with the toolkit were also carefully chosen and designed for ease of use and compatibility, and no additional components were necessary to connect them. The Cricket used a special version of the Logo language for programming, called Cricket Logo.

The most important catalyst in the development of the Cricket was the MIT Research Group's continued belief that science instruction dominated by direct instruction and lab activities was ineffective, and that children needed the opportunity to engage in real-world science [Resnick et al., 2000]. However, logistical challenges existed for teachers who wanted to set up Cricket-based activities: in the late 1990s and early 2000s, computers in classrooms were still few, slow, and unstable, installing device drivers was obtuse, and configuring serial ports (when they were available at all) was challenging. The idea of children doing robotics and physical computing in schools was also very new, and there were no good formats to fit this new type of activity into the school day. Given all of these limitations, the MIT Research Group's initial calls for widespread Cricket integration into school settings eventually gave way to calls for "systemic change in the logistical and conceptual organization of schooling" [Resnick et al., 2000]. These setbacks, more than a failure of the platform, pointed towards the difficulties in introducing a very different type of technology in classrooms, which was open-ended and mobile, requiring very different types of classroom facilitation and infrastructure. Still, the platform was quite successful in afterschool environments — in the US, it was implemented in 20 science museums, and more internationally.

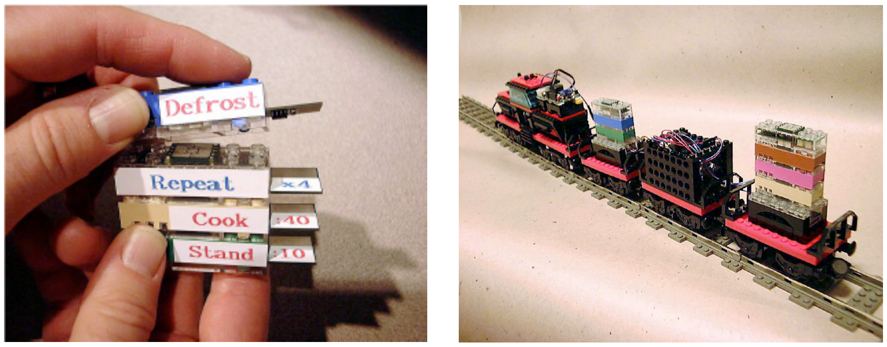


Figure 2.5: The Tangible Computation Bricks (left) and one of McNerney's conceptual prototypes: a "train" carrying a program of bricks (right). The idea was to have multiple "rules" or "methods" that controlled the train's behavior.

The Cricket set the standard for a whole new generation of devices during the following ten years. It spurred the development of other notable handheld microcontrollers, including the Handy Board [Martin et al., 2000] and the Tangible Computation Brick (Figure 2.5) [McNerney, 2000]. Inspired by the Braitenberg Blocks, McNerney took the Cricket design in a new direction by creating a tangible programming interface that allowed children to stack bricks together in different configurations to elicit specific behaviors.

This project was the second attempt at designing modular block systems as opposed to fully programmable ones, but soon the typical limitations of such modular systems became clear: the bricks only stacked in one direction. Some argued that the stacking constraint limited expression and asked for branching and two-dimensional structures [McNerney, 2000]. Another similar type of design, this time geared towards very young children, was proposed by Peta and Gordon Wyeth — the Electronic Blocks [Wyeth and Wyeth, 2001]. Given the limitations and the difficulties in manufacturing, the project did not reach a large number of users, and the group decided to go back to fully programmable systems [Martin, 2013]. These platforms, however, opened up a new realm of design, which would inspire many researchers several years later when microcontrollers became more capable, and sensors, actuators, and mechanical parts were much cheaper and more reliable.

2.2.2 A new dynasty is born: The BASIC Stamp, Stanford's board

We need now to go back in time to tell the story of a new category of boards that developed in parallel. In the early 1990s, a novel lineage of devices appeared: the BASIC Stamp board came out of a small electronics prototyping company named Parallax in 1992. However, the product was intended for quite a different clientele: it catered to hobbyists and engineers, and its designers had few connections to academic research or education. The BASIC Stamp was a microcontroller-based board with sensors and outputs, using a proprietary programming language based on BASIC. Many models were launched in the following years, and the platform was quite successful, selling millions of units. Until the wide popularization of Arduino-like platforms, Parallax was the key commercial vendor of programmable boards for hobbyists. The design choices, however, were quite different. The language was more powerful but much more complex. Furthermore, connecting devices to the board required external components and soldering. The BASIC Stamp did not follow the "Cricket" hardware model; it exposed one extra hardware layer that up to that point had been invisible to users: the pins of the microcontroller. This was a radically different design principle. The design of the boards, which was based on the popular PIC microcontrollers, was elegant and built to meet the needs of the hobbyist community at the time, mostly comprised of electronics aficionados and engineering students. However, the BASIC Stamp was not an all-in-one solution, and lacked built-in components to drive motors and receive sensor values. It was essentially a breakout board for the microcontroller itself, giving users full access to its pins and functionality. The history of the Basic STAMP, as it is told on the Parallax Website, confirms the target audience: "BASIC Stamp microcontroller was the tool they [the founders of Parallax] needed to do their own hobby projects. It let ordinary people program a microcontroller for the first time [...] the user base was tremendously diverse [and included] scientists and hobbyists to engineers and entrepreneurs."

The meaning of "tremendously diverse" here is quite different from the idea of diversity held by the creators of the Crickets. As we will discuss later in this monograph, these design differences, as well as the radical difference on the intended audience, had profound implications on how children would end up using these devices.

About a decade after the BASIC Stamp, researchers at Stanford working with interactive technologies and music at the Center for Computer Research in Music and Acoustics (CCRMA), such as Bill Verplank and Pascal Stang [Wilson et al., 2003], felt the need to offer a better hardware platform for their students who were building a variety of interactive music devices in undergraduate and graduate classes. Unhappy with the current platforms, they made a new design choice with important consequences: they switched to using AVR chips and built their own microcontroller board, which was the origin of the Arduino platform. In a publication in 2003, they justify their choice in this way:

> "... why not give students a black box that does 8 channels of A/D conversion [instead of using microcontrollers]? The short answer is pedagogical. Using a programmable microcontroller allows the students to learn about computer architecture, digital logic, programming, A/D conversion and serial and parallel communication protocols. [...] It gives students exposure to the technology used in actual commercial products, demystifying the world of embedded systems. In this respect, the switch to the AVR has been significant. It represents a shift from essentially a hobbyist's toolkit to professional, commercial-grade technology that is much closer to the technology used in a wide variety of existing commercial devices. [...] Admittedly, using a microcontroller, and specifically the AVR, creates more work for the students [...] The platform's infrastructure [...] gives the advanced student full 'under-the-hood' access to a set of well-established, industry standard tools." [Wilson et al., 2003]

Essentially, their realization was that their population of students had certain important learning goals (computer architecture, digital logic, programming), and that using technologies closer to the basic microcontroller level would be beneficial, even though it would require more specialized training. This type of goal is much different than the goals of the designers of the Cricket or Lego/LOGO platforms. However, this crucial difference in learning goals was lost in translation when these AVR-inspired devices (such as Arduino) became wildly popular in educational settings.[4]

2.3 The Third Generation: Broadening Participation and Accessing New Knowledge Domains (GoGo Board, Phidgets, Wiring, and Arduino)

Microcontroller kits designed in the early years of the new millennium continued to build on earlier designs. However, the literature suggests a growing focus by the research community on designing devices that would broaden participation in computing and allow access to new domains of knowledge. In other words, new designs emerged to better expose new groups to powerful experiences in computing, as well as to create kits to enable children to explore concepts previously considered too advanced. Curlybot, the MetaCricket, Phidgets, and the GoGo Board all spoke to the former concern, while the MIT Tower focused on the latter.

Curlybot was a digital manipulative developed at the MIT Media Lab for children ages four and older. Frei and his colleagues designed Curlybot in response to several issues. At the time, many digital manipulatives were being designed for middle and high-school children, but not for children as young as four. Furthermore, many of the computational environments that had been designed for children were limited to screen based interactions [Frei et al., 2000]. In order to address these issues, the team created a digital manipulative that was programmed

[4]The AVR platform had one decisive advantage over the PIC platform: a free, comprehensive, widely-supported compiler. The official PIC compiler was not free. Some researchers and engineers attribute the current dominance of the AVR platform to the existence of a free C compiler.

by example. Users would perform actions with CurlyBot, which would be recorded and later "played back." This innovation would later inspire the designers of other systems, such as Topobo (see Section 2.4).

Another Cricket offshoot, the MetaCricket was a hardware and software construction kit designed to make rapid prototyping of computationally-enhanced devices easier for non-engineers. In order to lower the barrier of entry for designers, Martin and his colleagues extended the Cricket design [Martin et al., 2000] by adding a collection of small bus devices that could communicate with the core Cricket device. Before the MetaCricket, users would need to create a plurality of crickets for different purposes (display, music, and so on). Now all of the circuitry was bundled onto the device itself and could be daisy-chained off the bus of the class cricket. This same design idea was later utilized for the Arduino "shields."

The University of Calgary's Phidgets project [Greenberg and Fitchett, 2001] (Figure 2.4) created "physical widgets" designed to make it easier for designers and programmers to develop physical interfaces. They were not designed to be used by children, but to be the tangible equivalent of screen widgets, the easily reusable building blocks that software designers use to build user interfaces. Greenberg and Fitchett were motivated by a similar problem that the BASIC Stamp sought to address. They wanted designers to spend more time on actual physical interface design and less on low-level electronics design so that their physical widgets could be inserted more readily into physical interfaces to make prototyping easier. Their design was effective and widely successful, and very soon it became commercially available. In terms of design, they followed the "Cricket" model, with an extra software layer that backgrounded much of the microcontroller's inner workings and exposed sensors and actuators in a much simpler way. The Phidgets were a step up in terms of hardware usability — no soldering was necessary, and users did not have to deal with off-the-shelf electronic components. The boards had easy to use, polarized connectors, and included in the kit were specially designed sensor boards, which were automatically detected by the system. Software plugins were developed for all the major programming languages then available. However, Phidgets had the same transparency flaws (both were not open source) as the BASIC Stamp and had no central processing power of their own; the hardware system had to be connected to a computer at all times.

In 2001, a new Cricket-inspired design was generated, this time intended for learners in developing countries. According to Sipitakiat, Blikstein, and Cavallo, microcontroller kits were simply not accessible to much of the developing world, and even less so in schools. Programmable bricks were expensive, hard to find, and only well resourced schools and organizations could afford them. The team wanted to develop a variant of the programmable brick that would be low-cost, open source, and easily assembled with simple tools. Their solution, the GoGo Board[5] (Figure 2.5), could be assembled on-site by the user at a very low-cost, and they made sure that all the components would be available in electronics stores and markets in major cities of developing countries, including Brazil, Mexico, and Thailand. The printed circuit board was single-sided and had large traces, making its "artisanal" production possible. The approach was innovative for a number of reasons. First, it made the kit more affordable because assembly (and repair) could be done locally, even by the children themselves. Second, it made its consumers into producers, and the authors observed a great sense of agency and ownership among the children and teachers who assembled their own robotics boards. Third, the board was the first such project to offer two operational modes, autonomous and tethered, extending the functionality of programmable bricks even further. The tethered mode supported several programming languages, including Microworlds Logo, Java, C/C++, and NetLogo, which gave the GoGo Board the ability to use a computer's superior processing power for experiments and interactive systems. For the first time, children could make screen elements move using physical sensors and make actuators turn on and off as a result of computer instruction. Another important contribution of the GoGo Board was that it allowed for the extensive use of found and broken materials: the hardware was designed to be tolerant of non-standard connectors, motors, and sensors [Sipitakiat and Blikstein, 2010, Sipitakiat et al., 2002, 2004]. Finally, a crucial

[5]http://www.gogoboard.org

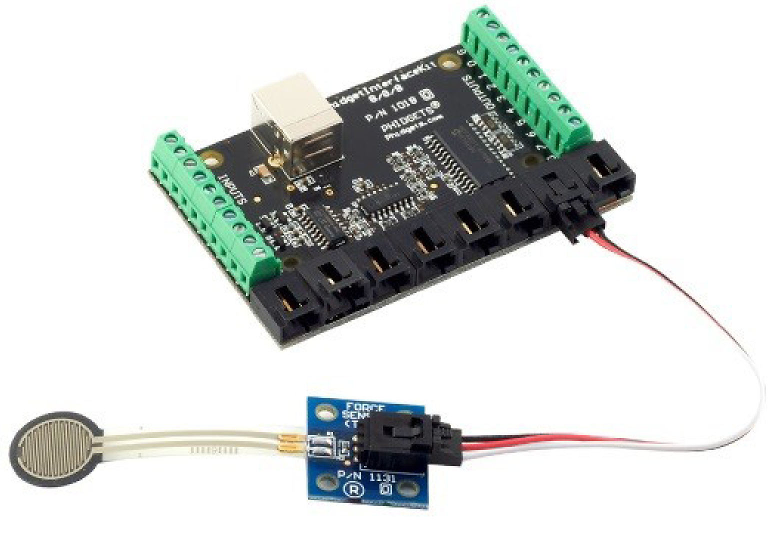


Figure 2.6: The Phidget interface, with a force sensor attached.

innovation was to make the hardware design open-source, so designers in different countries would be able to adapt the board to their own needs. There were custom boards developed in Brazil (BR-Gogo, see Figure 2.5, Ramos et al., 2007), Korea, and Mexico (see the board in Figure 2.5, which was entirely built with found and repurposed electronic components by Mexican schoolchildren).

The Curlybot, MetaCricket, Phidget, and GoGo Board were all responses to issues of accessibility and ease of use, but other projects in those years focused on issues related to extensibility (Figure 2.7). However, towards the middle of the decade, a different breed of programmable bricks started to develop in research labs, this time less concerned with issues of accessibility, but pushing the boundaries of what was possible with physical computing. The MIT Tower took a similar approach — modular design — in order to extend the capabilities of construction kits.

Lyons and Mitkhak, principal architects of the MIT Tower, wanted to create a more versatile construction kit that would enable anyone to design regardless of background and technical proficiency. There were

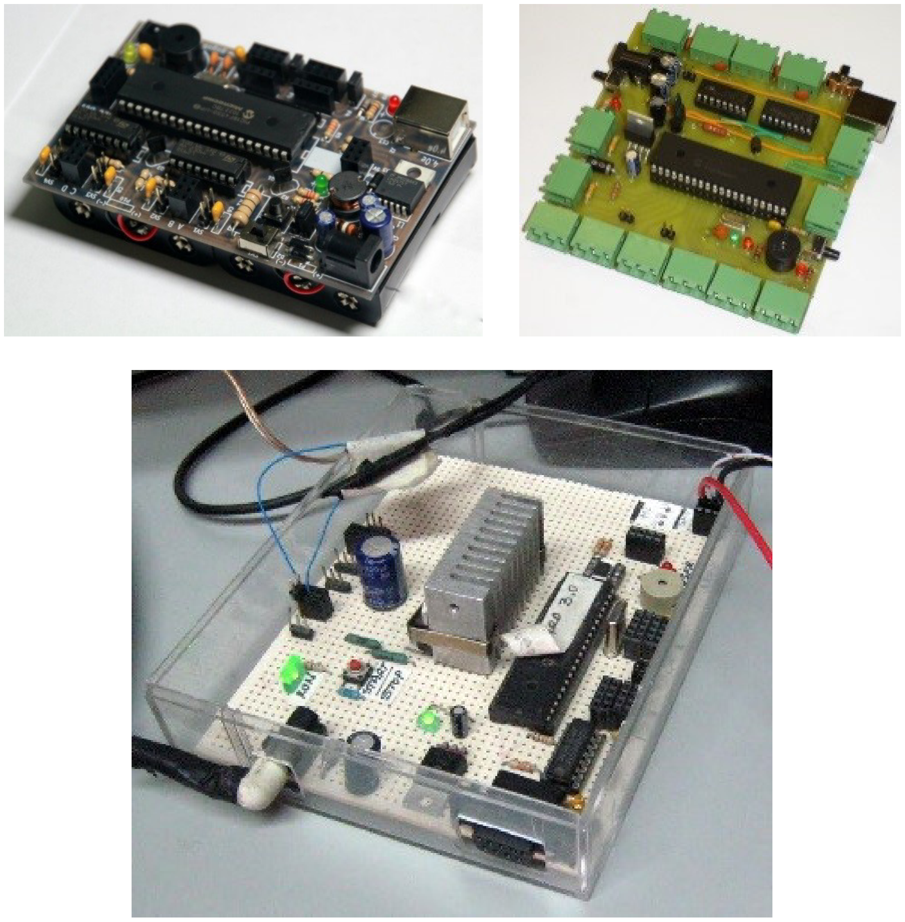


Figure 2.7: The GoGo Board (top), the Brazilian version of the board (bottom left), and a version made with repurposed electronic components by Mexican schoolchildren.

already a number of rapid prototyping kits in use that lowered the entry barrier for novices and experts alike. However, the existing technologies all had limitations in terms of processing power or openness of the software. The MIT Tower addressed many of these limitations with a fully modular computational construction kit that supported Logo and other languages and included standalone system components. The Tower system was the "Cadillac" of programmable bricks at the time, and had several add-on boards that greatly expanded the capabilities of the system, making its processing power comparable to a low-end computer. Additionally, users could attach standard peripherals such as keyboards and mice [Lyon, 2003]. The Tower was a visionary board that inaugurated a new type of design. Today, many computer-on-a-board designs such as the Raspberry Pi[6] or the BeagleBoard[7] still try to realize the MIT Tower vision.

A late entrant in the third generation of devices was the Wiring platform, inspired by the Stanford CCRMA board. Created at the IVREA Institute in Italy by Hernando Barragán [2004], it catered to artists and designers. It made use of the Processing programming environment, a stable and well-supported development platform created at MIT. Barragán's motivation is clear:

> "The programmable bricks, Crickets Logo Board and GoGo board share that are all targeted to children's education. They are toys that include mechanical parts, and the development environments are based on the concept of visual programming with the Logo programming language. They are very limited systems for activities involved in physical computing, but lack the flexibility required for general-purpose applications." [Barragán, 2004]

His goal was to cater to a different audience by creating a more flexible and general-purpose board, without all the design restrictions of designing for children. The Wiring platform was very powerful, but also expensive. Barragán's advisor, Massimo Banzi, was inspired by his work and decided to create a lower cost version, perceiving the need for inexpensive, easy-to-use hardware kits. He teamed up with other researchers and created the platform that would become the industry standard: the Arduino. The Arduino followed the breakout model of the BASIC Stamp: it exposed the microcontroller pins to the user directly and did not have extra electronics for connecting motors and LEDs. Circuits had to be built externally on breadboards, and additional components were needed for driving motors. The Arduino used

[6]https://www.raspberrypi.org/

[7]http://beagleboard.org/

a flexible architecture for expansions ("shields," similarly to the MIT Tower), a focus on open-source and distributed expertise, and a "bare-bones design," which made it low-cost compared to other platforms.

The advent of the BASIC Stamp, Wiring, and Arduino platforms, although not designed for children, brought some very positive wins for education (Figure 2.8). Except for the LEGO Mindstorms, which was expensive and used proprietary technologies, and the PICO Cricket, which was short-lived, the other Cricket-inspired platforms did not manage to reach a worldwide audience because the academics behind

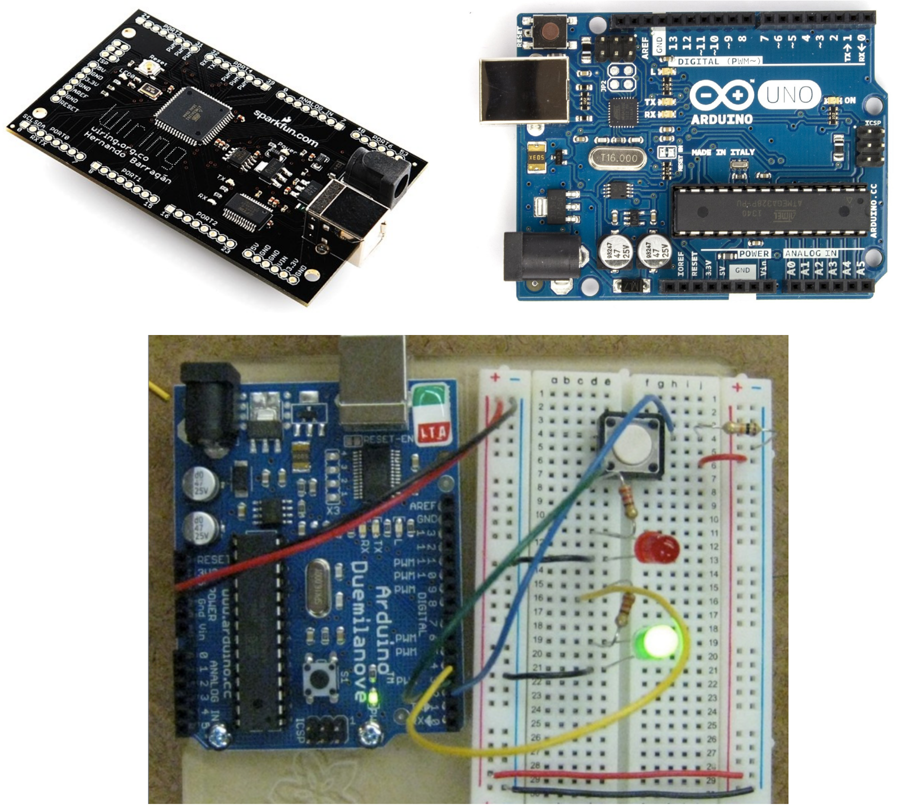


Figure 2.8: The Wiring (top left) and the Arduino Uno (top right) boards, and an Arduino connected to a breadboard (bottom) in a very common use case (making an LED blink).

them were not well-equipped to start companies and set up worldwide distribution channels.[8] In addition, the Internet was still incipient in the early 1990s, so the dynamics of knowledge sharing were quite different. Marvin Minsky famously said that a language needs more than a good grammar, it also needs a literature — a good corpus of things to see and learn from. A vast user community developed around BASIC Stamp and Arduino tools, which generated an immense body of educational materials, tutorials, and curricula. Arduino's success builds on this large literature that surrounded it. In addition, these devices became very robust, ran on all platforms, and were designed to be open source from the ground up. Not only did this generate an unprecedented amount of collective expertise, it also brought commercial vendors into the fold, ensuring the worldwide availability of these devices.

2.4 The Fourth Generation: New form factors, new architectures, and new industrial design (Pico Cricket, Lilypad, Topobo, Cubelets, LittleBits)

After 2005, the platforms that emerged were either key extensions of earlier iterations or radically new designs, many intended to broaden the participation of girls and younger learners. In 2006, LEGO launched the next generation of its robot development kit. The NXT's brick was a departure from the original RCX design and included a new breed of sensors and actuators as well as updated programming software. Similarly, the Cricket platform spawned a new generation of Cricket-based designs, including the Handy Cricket, Handy Board BlackFin, and Pico Cricket. Significantly, many of the designs in this period applied some of the design principles synthetized by Resnick [Resnick and Silverman, 2005]: designs with low floors and high ceilings, which could also support "many styles" and "many paths."

Fred Martin and Li Xu wanted to create an accessible and engaging way to teach compiler fundamentals to a more diverse audience of undergraduates, so they designed the Handy Crickets and the Chirp

[8]Still, at its peak, the Cricket platform was deployed to over 20 science museums in the United States.

language [Martin and Xu, 2006]. Later, they designed the HandyBoard BlackFin, an all-in-one solution for classroom use, a pioneering type of computing device that later came back as the Beagle Board and Raspberry Pi [Martin and Chanler, 2007].

Another evolution of the Cricket platform was Pico Cricket, designed to bring together art and technology in a robotics kit. The authors observed that although robotics had become increasingly popular, “the way robotics activities are introduced in these settings is unnecessarily narrow” [Rusk et al., 2008]. In most classrooms and workshops, the first activity involved building a car, which helped promote a significant gender imbalance. The team was interested in developing more ways to engage those students who were not interested in traditional approaches to robotics, but who would become “more interested when robotics activities are introduced as a way to tell a story or in connection with other disciplines, such as music and art” [Rusk et al., 2008]. The connectors and architecture with greatly modified to make the kit self-explanatory, and to make it look less gender-biased — the kit included materials to build projects other than the ones historically associated with boys (such as robotic cars). Children could easily connect output devices and sensors to the device and then program the device using a block-based graphical programming language.

However, many of the stars of this generation would follow a different tradition, that of the Braitenberg Blocks and modular systems. RoBlocks (later Cubelets, Figure 2.9) was a robotic blocks and software package that allowed children to build simple robots easily by snapping together active blocks. The RoBlocks platform consisted of nineteen blocks in four categories (sensors, actuators, logic, and utility). Computation was distributed throughout the kit’s pieces rather than restricted to a central computer that controlled the pieces’ functions [Schweikardt and Gross, 2006, 2007].

Topobo[9] (Figure 2.10) was another design that opted for a distributed modular system, taking design cues from earlier developments, such as the programming by example technique in Curlybots [Raffle et al., 2004] and the modular design of the MIT Tower system [Lyon,

[9]http://www.topobo.com/

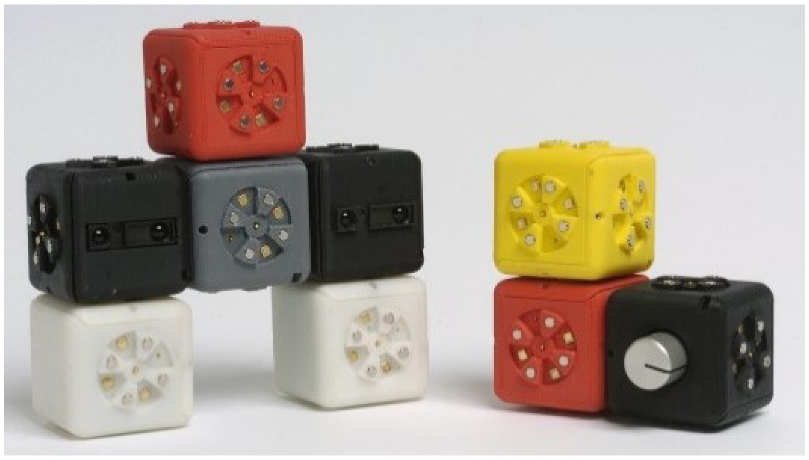


Figure 2.9: The RoBlocks system (later Cubelets).

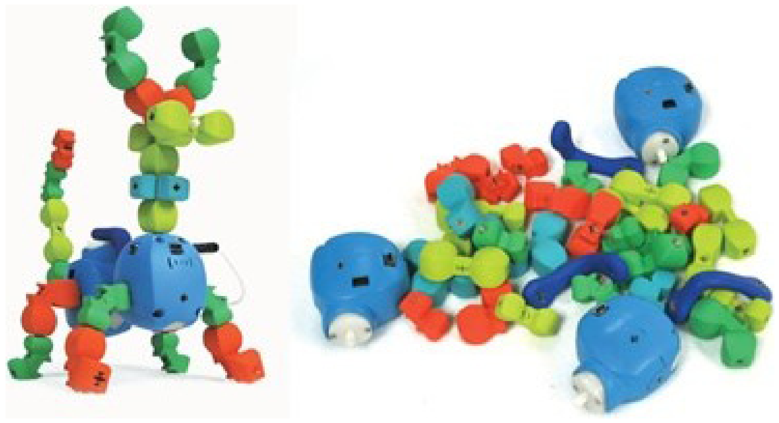


Figure 2.10: The Topobo platform: an assembled artifact (left) and the active and passive parts (right).

2003]. However, the key innovation in Topobo was the introduction of active components with embedded kinetic memory. Active and passive parts could be snapped together to form models of animals, regular geometries, and abstract shapes. Children would program their modular creations by example, and the system would record the program and play it back for them, in order to explore kinematic concepts, such as balance, center of mass, center of gravity, coordination, relative motion, and relationships between local and global interactions [Raffle et al., 2004].

Another kit that used a modular design, but made programming more explicit, was RoboBlocks [Sipitakiat and Nusen, 2012]. Sipitakiat and Nusen designed a robot that could be programmed with tangible blocks, following the Logo syntax, and targeted at elementary school learners. Differing from Topobo and RoBlocks, the robot and the programming blocks were separate, thus the system followed the architecture of the original Logo turtle.

One notable kit responded to calls for less gender-biased physical computing and robotics, allowing for new forms of expression. The LilyPad Arduino[10] (Figure 2.11) was a pioneering design that, for the first time, proposed a hardware platform focused on “soft materials” such as fabric, providing a new medium to engage a diverse range of students in engineering and computer science — especially girls. The open-source construction kit for e-textiles was rooted in Buechley’s earlier work

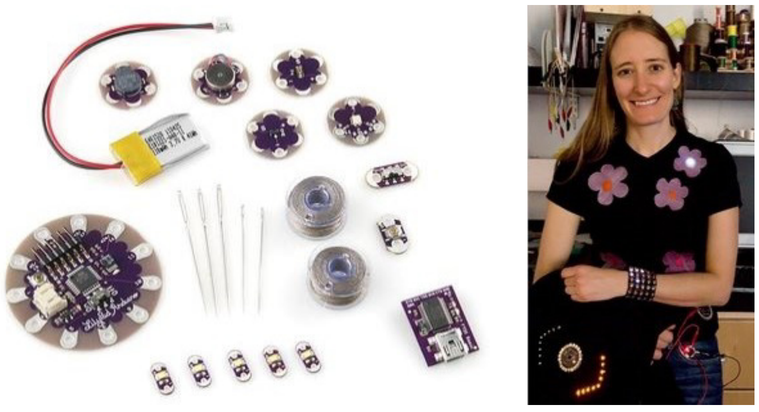


Figure 2.11: Lilypad Arduino kit, and Leah Buechley, showing some of the e-textiles built with the toolkit.

[10]http://lilypadarduino.org/

on craft-based electronics, which included the production of an electronic sewing kit, quilt snaps, programmable wearable displays, fabric printed circuit boards, electronic sequins, and socket buttons [Buechley and Eisenberg, 2007, Buechley et al., 2006]. In terms of design, the LilyPad borrows most of the Arduino's electronics and software, but with one fundamental difference. Buechley designed the kit in such a way that no external electronics were needed, and all the parts (LEDs, sensors, motors, and battery packs) were mounted on a small printed circuit board with all the extra components built in. This was a key usability innovation for the Arduino platform.

The LilyPad Arduino was released as a commercial product in 2007, and it inspired many extensions, including the TeeBoard, LilyPadadone, LilyPad XBee, DaisyPIC, and Bling Cricket [Buechley and Hill, 2010]. Buechley has been studying the efforts of the LilyPad Arduino community since the platform was released, and her research has highlighted the need to develop new strategies for broadening participation in computing. Buechley urged the design community to shift its focus. Instead of "unlocking the clubhouse," or trying to make traditional computing culture accessible to women, "it may be more constructive to try to spark new cultures, to build new clubhouses" [Buechley and Hill, 2010]. Buechley also explored other traditional materials. One of her projects aims to augment traditional materials like paper with computational capacity, so that children can engage in programming in more informal, approachable, and natural ways. Their flexible pieces (processor, battery, sensors, motors, and so on) can be attached to specially treated paper to create paper-based working programs. In another project, Buechley and colleagues challenged the construction kit paradigm entirely by proposing a new direction. They noted that while construction kits facilitate the making of technology, their modularity "constrains what we build and how we think" [Buechley et al., 2011]. They proposed a "kit-of-no-parts," or a handcrafting approach to learning about electronics and programming, as opposed to a construction kit approach. "Craft," they argued, "allows for rich design exploration that construction kits of pre-manufactured parts cannot offer" [Buechley et al., 2011]. In their recent designs, they propose that we should move from assembling electronics to crafting them; more recently, they advocated the idea of the “untoolkit” along those same lines.

Another example of a platform for broadening participation is the Hummingbird kit, developed as an offshoot of the Robot Diaries project at Carnegie Mellon University, with the overarching goal to “enable girls to engage with, change, customize, or otherwise become fluent with the technology in their lives” [Hamner et al., 2010]. One final entrant in the category is the Makey Makey toolkit, a modification of the Arduino platform that allows children to use everyday objects (including fruits or any mildly conductive object) as sensors, without breadboards or additional electronics. The Makey Makey became quite popular due to one clever innovation. It maps sensors to regular keyboard keystrokes or mouse clicks, so any existing software that is controlled through a keyboard or mouse can be used: virtual music instruments, animated characters, games, storytelling environments, and so on. This made the kit extremely easy to install and use.

The littleBits platform[11] (Figure 2.12) also appeared around the same time. Similarly to Cubelets and Topobo, it did not require a computer, and had blocks that would magnetically connect to assemble circuits. The blocks were color-coded, clearly identifying inputs, outputs, logical operators, and power. The physical design made sure that only functional circuits could be assembled, and a proprietary communications protocol between the blocks would take care of all data and power management.

Many other platforms were launched within this generation, but many were short lived or were restricted to relatively small niches. Examples of those are the Netduino (Figure 2.13, left) MAKE Controller (Figure 2.13, right), NET Gadgeteer, PCduino. More recently, a plethora of platforms and variations were launched. For space considerations, we will leave these new versions out of this review.

Finally, a different type of platform focused on broadening participation by catering to younger audiences. The Tern system,[12] designed by Michael Horn at Tufts University, was designed to bring

[11] www.littlebits.cc
[12] hci.cs.tufts.edu/tern

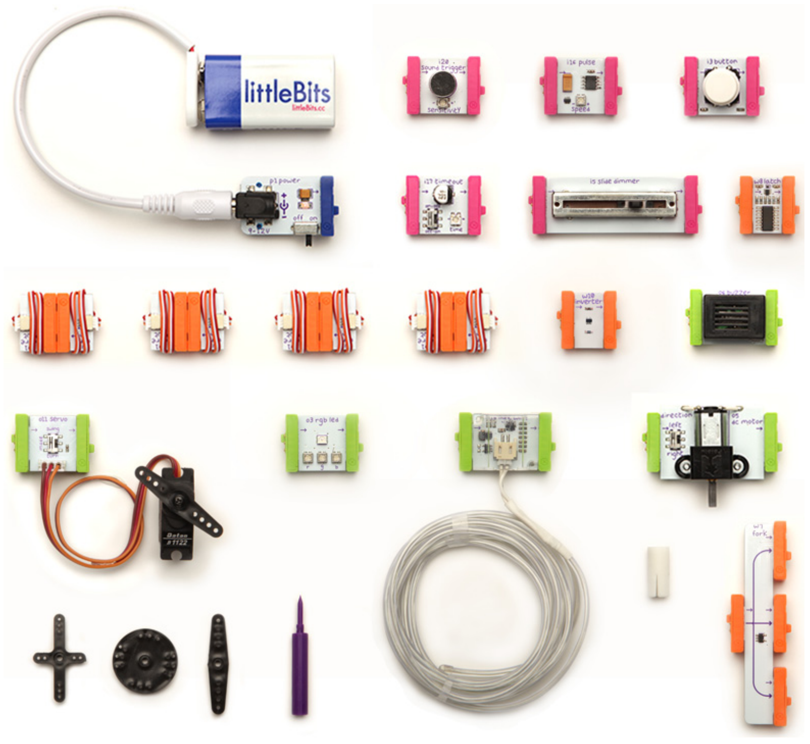


Figure 2.12: The littleBits platform.

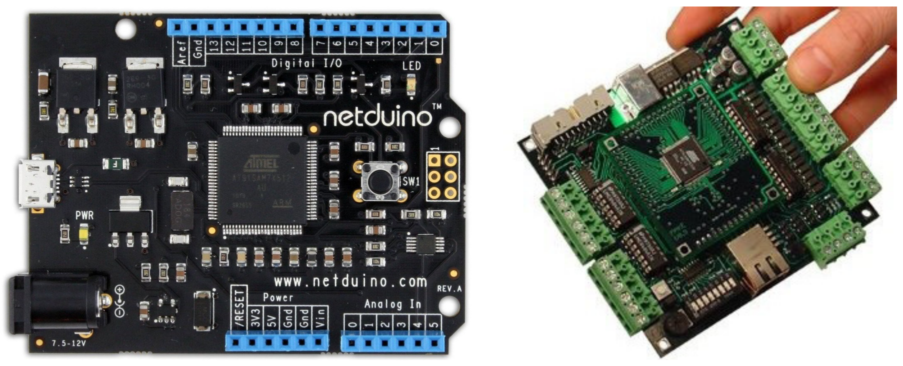


Figure 2.13: Netduino (left) and the MAKE Controller (right).

programming to younger audiences by moving it from the screen to the tangible realm, using wooden blocks with fiducial codes. Since the system was focused on programming, it did not include physical computing components such as sensors and actuators. However, a new iteration of the system — first named KIWI and then KIBO[13] — developed by Marina Bers at Tufts University took Horn's platform to the realm of physical computing by adding a physical robot to the system, sensors, and output devices (Figure 2.14). One top of Horn's original innovation

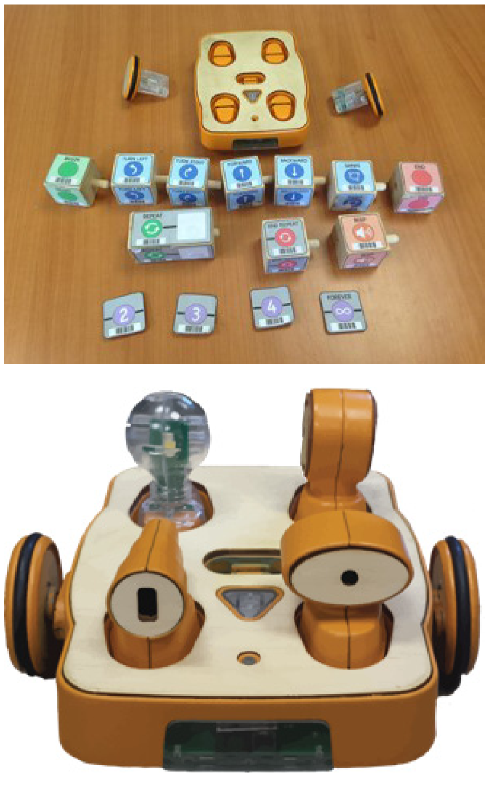


Figure 2.14: The KIWI/KIBO system, designed for children aged 4–7 years old. It uses wooden blocks without electronics (tagged with optical codes) as a programming platform, and a robot base which can read and execute the optical codes without a computer.

[13]kinderlabrobotics.com

of using low-cost wooden blocks with optional codes instead of electronics, KIWI/KIBO also introduced a new idea: the robot could read the optical codes from the wooden blocks directly, without a computer.

2.5 The Fifth Generation: Single board computers (Raspberry Pi, PCDuino, Beagle Board)

A new development in this field came in 2012 with the launch of the Raspberry Pi platform (Figure 2.15). Despite the fact that many other single-board computers have been in the market for a long time (for example, Beagle Board,[14] Figure 2.17), the Pi platform was the one that made them popular and affordable (4 million Pis have been sold up to February 2014). Other boards worth mentioning are the PCDuino[15] (Figure 2.16), which merged the idea of a full-blown computer with the Arduino architecture, offering output and input ports directly on

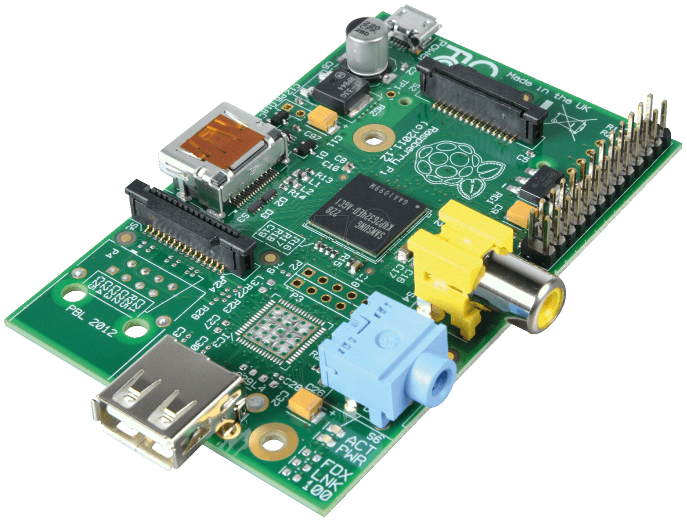


Figure 2.15: The first Raspberry Pi board.

[14]www.beagleboard.org
[15]www.pcduino.com

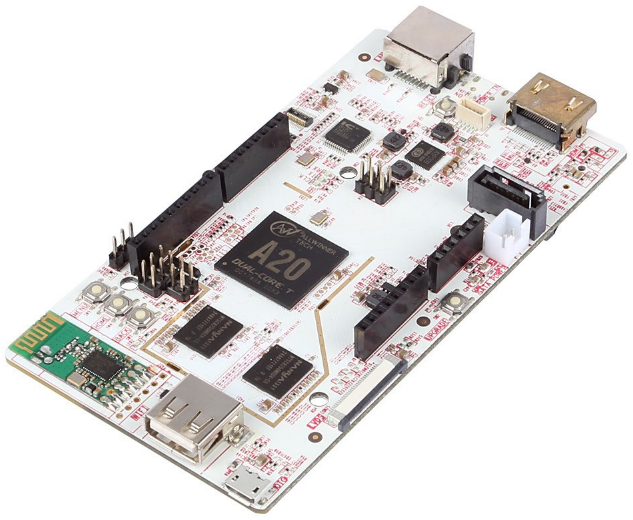


Figure 2.16: The PCDuino board.

the board; and the Intel Galileo,[16] a full-blown Windows-compatible computer also with the Arduino pinout.

The relatively low price point of these devices, which is quite close to many microcontroller-based devices, made them a viable alternative to the microcontroller in some educational scenarios. Pis and similar boards offer an entire new set of features for robotics and physical computing, but also some tradeoffs. For example, they allow computational-intensive applications such as real-time image processing; for example, students could use a webcam to implement a robot with real-time computer vision. Because these devices are full computers, they can be accessed via a remote terminal, enabling users to interact with the device at a distance using a tablet or phone. However, their physical computing device also changes the interaction paradigm in important ways. For example, users would not need a computer at all, since they can be programmed directly if connected to a monitor. This solution,

[16]http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-overview.html

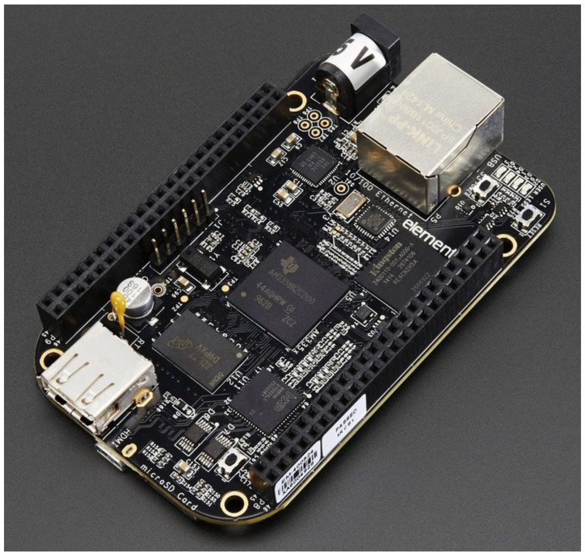


Figure 2.17: The Beagle Board.

although elegant, fundamentally changes how students do the programming and debugging of their devices. They could, for example, leave the Raspberry Pi board mounted on a robot, use remote desktop software to access it wirelessly, and then make changes to the program on-the-fly as the robot goes around the room, without the need to stop the activity, reprogram, and run it again. Another promising direction is the interaction with mobile devices via Bluetooth or wifi technologies, for environmental data collection, controlling everyday objects, and "Internet of Things" applications. These new ways of interacting with physical computing devices could open a host of new possibilities for designers, but further research and design are needed to ascertain how they will play out in education.

3

A New Framework for Generative Design, or a Call for Action Against the Arduino Monoculture

The proliferation of designs for digital manipulatives has prompted researchers to create taxonomies to better frame design principles, audiences, and learning goals. One such taxonomy, created by Eisenberg and collaborators [Eisenberg et al., 2002], has three categories. The first, "specificity of constructions," ranges from assembly kits that can generate just one product (for example, for creating anatomical models of animals) to very flexible platforms such as Lego, which affords infinite possibilities. The second, "domain specificity," gauges how much a given construction kit can be connected to a particular field of knowledge (for example, a kit for assembling molecules in chemistry), or if they are general kits. Finally, their last category refers to "means of connection and construction materials," and examines how the parts connect and what material is used. Zuckerman et al. [2005] came up with a more general taxonomy with two groups: "Froebel-inspired Manipulatives" (FiMs) and "Montessori-inspired Manipulatives" (MiMs). FiMs are analogous to wooden blocks: they allow children to build physical objects (for example, Froebel gifts, Legos), and their main learning goal is design and basic engineering. MiMs are tangible systems that enable children to explore particular domains of knowledge, such as mathematics or physics (for example, Cuisinaire rods, and so on), and concretize abstract concepts.

While these frameworks are useful in categorizing toolkits in various ways, especially regarding how connected they are to a particular discipline, there is still a missing level of analysis, because these frameworks do not address three important dimensions:

(1) *What are the levels of abstraction exposed to students and what interactions do they afford?* With digital manipulatives, children have to learn an abstraction layer before they can even think about the concrete projects they will produce [Shapiro, 2015]. In Figure 3.1, we have a representation of the different abstraction layers in a microcontroller-based device. Level 1 exposes all the most basic electronics components and requires complex skills; level 2 takes care of most of the electronics design and only exposes the microcontroller architecture, while level 3

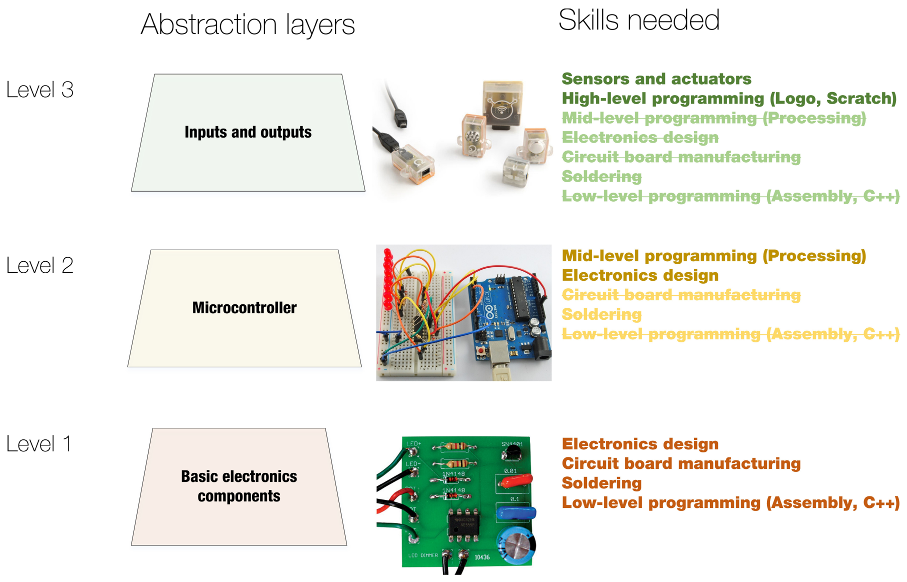

Abstraction layers
Skills needed
Level 3
Inputs and outputs
Sensors and actuators
High-level programming (Logo, Scratch)
~~Mid-level programming (Processing)~~
~~Electronics design~~
~~Circuit board manufacturing~~
~~Soldering~~
~~Low-level programming (Assembly, C++)~~
Level 2
Microcontroller
Mid-level programming (Processing)
Electronics design
~~Circuit board manufacturing~~
~~Soldering~~
~~Low-level programming (Assembly, C++)~~
Level 1
Basic electronics components
Electronics design
Circuit board manufacturing
Soldering
Low-level programming (Assembly, C++)

Figure 3.1: Representation of the idea of abstraction layers, from the most basic (level 1, basic electronics components) to level 3, where students are only exposed to input and outputs.

encapsulates even the microcontroller and exposes only inputs and outputs. Microcontroller pins, resistors, analog/digital signals, AC/DC current, and polarity might or might not be hidden behind an abstraction layer, so the design of the layer exposed to students determines how much of those concepts in electrical engineering they will have to learn prior to even beginning their projects.

(2) *What are the direct connections between specific design decisions and the learning goals intended for the toolkit?* Design decisions about these abstraction layers can radically affect the achievable learning goals. For example, a toolkit that is supposed to help students learn robotics, but makes them spend half of their time figuring out how a breadboard works, will not be able to live up to its goals.

(3) *Toolkits should be considered as part of a larger developmental trajectory.* If our goal is to turn children into experts, we need to design a developmental trajectory. This trajectory needs to have multiple milestones: which scaffolds or abstraction layers should be abandoned as children progress through the developmental trajectory, and which ones should stay? Over time, and considering the use of several toolkits, which parts of the construction tasks should children do by themselves, and which parts should be done by the materials?

My claim is that by better examining these issues, we will be able to make predictions of the match between different toolkits, their learning goals, and their developmental purpose. Historically, this concern was present in the design of many toolkits such as the Cricket and the LEGO Mindstorms kit (designed by education researchers), but was left behind when platforms such as Arduino (designed by interaction designers) became the *de facto* standard.

In this monograph, I propose a categorization based on design commitments and principles, employing three analytic constructs:

1. ***Selective exposure*** (*What is foregrounded or backgrounded in hardware and software design?*): I have identified specific

elements — based on designers' theoretical or pedagogical commitments — in several toolkits that can be exposed or hidden from users. Sometimes this happens specifically for pedagogical purposes, so that users can focus on one particular learning goal. For instance, if designers want users to learn about microcontroller architecture, they should "expose" the microcontroller pins to the user; otherwise, they can overlay a new hardware layer to make the pins invisible. Selective exposure can also have the effect of changing the design compromise between usability and power, as we will discuss in the next two items.

a. ***Selective Exposure for Usability*** (*or "embedded error correction" — How does the material communicate rules for its use?*): inspired by McNerney's idea of self-debugging [McNerney, 2000], I propose that, again, depending on the learning goals of the toolkit, some categories of errors should be corrected by the design of the toolkit itself, increasing usability. Designers make decisions that prevent errors, so that learners can get up to speed and experience success, in other words, students can start building with a "low threshold." For example, if we are not interested in electrical polarity as a learning goal, the connectors in a robotics construction kit should either "fix" (that is, reorient) themselves or not allow wrong connections to be made. Traditional LEGO bricks, for example, have a built-in error correction system that only allows parts to fit in ways that are structurally sound, and in mostly regular and symmetrical ways — the result is that children can build much more complex structures with the material than without. The error correction could also be exaggerated, in which case the materials would do "too much" for the students, or be nonexistent, which would arguably limit the complexity of the resulting projects since all error corrections would have to be done by the users themselves.

b. ***Selective Exposure for Power*** (*or "tangibility mapping" — How are cognitive and physical operations mapped to each other, and how can the design make them more explicit?*): In order to raise the ceiling of possibilities, designers want learners to know and have access to what is possible within a toolkit, beyond what is simply easy to construct. So this design dimension makes evident the new, innovative, and different possibilities and ways things can be connected to one another to reveal the real power of the toolkits. This construct, consequently, examines how the toolkit maps desirable cognitive operations to physical operations. For example, if a toolkit has magnetic pieces that snap together spontaneously, they map the fact that conceptually these parts "want" to connect as well. If certain parts "feel" or look like they should be inside other parts, they would be mapping some hierarchy about the system: for example, in tangible programming toolkits it is common to see blocks that would fit easily inside a "repeat" block. Conversely, the mapping could also be confusing, if the physical operations do not map well to the abstract ones. There could be several dimensions for tangibility mapping: connection between parts, their hierarchy, or indications of the flux of information between tangible blocks.

4

Applying the Framework to Existing Platforms

In what follows, I will examine most of the platforms described earlier in this monograph to illustrate these three constructs, draw conclusions about their potential and limitations, as well as provide directions for future designs. In particular, I will focus on a detailed analysis of the first construct, and also present examples of the remaining two.

4.1 Selective exposure

4.1.1 Exposure of the microcontroller

For physical computing toolkits, a first criterion of selective exposure refers to the level of exposure of the microcontroller, the "brain" of most toolkits. A microcontroller is a programmable component that interfaces with external devices such as sensors and motors. As we can see in Figure 4.1, they have several terminals to which these output/input components are connected. Each of those terminals has a number or a code, and can be set to "high" (roughly equivalent to "on") or "low" ("off"). The use of "high" and "low" is due to the way semiconductor components work and the voltage level that will be applied to the pins. This terminology makes perfect sense to engineers who have to deal

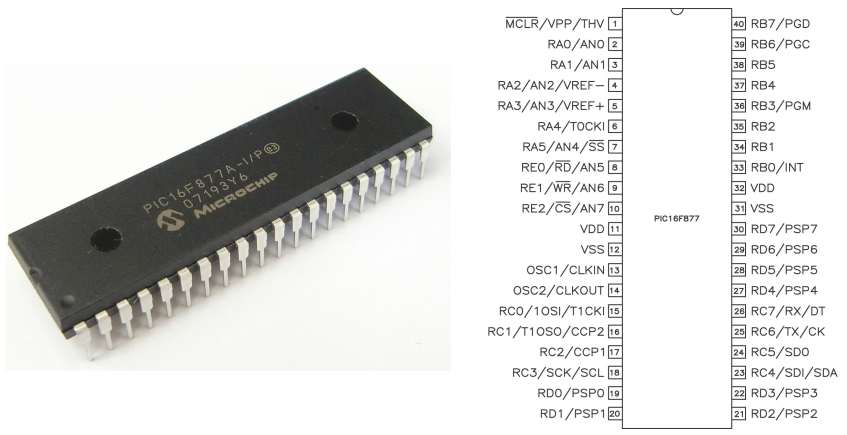
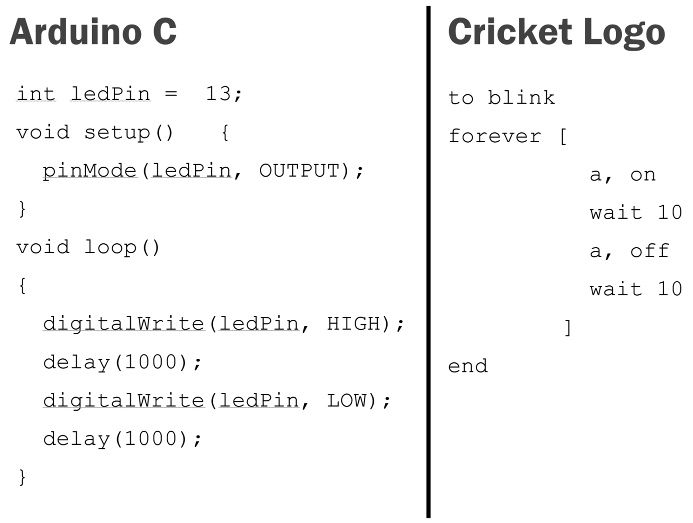


Figure 4.1: A very popular microcontroller (PIC 16F877A), and its pinout diagram.

with the intricacies of circuit design and logic, but they are obscure symbols to novices who would instead instantly understand the meaning of "on" and "off."

The Lego Mindstorms kit, for example, adopts a very careful level of exposure of the microcontrollers, heavily inspired by the design of the MIT Cricket. These devices kept most of the inner working of microcontrollers hidden from users and exposed only their high-level functionality (that is, receive sensor values, process those values in a program, and control output devices such as motors or lights). Users could not activate individual pins of the microcontrollers or any of its hardware-level features; instead, there was an extra abstraction layer that enabled students to control three intuitively-named output ports (A, B, and C) and receive values from three sensors (1, 2, and 3) — much more learnable than memorizing that the input ports are, for example, pins 16, 19, and 21. In this case, selective exposure increases usability by employing embedded error correction.

On the other end of the spectrum, other devices expose most of the specificities of the underlying electronics and are based on the idea of a barebones "breakout board" for microcontrollers. The BASIC Stamp, Wiring, and the Arduino board are popular representatives of this category. With these devices, users cannot simple address "motor A, on"


Arduino C

```
int ledPin =  13;
void setup()   {
  pinMode(ledPin, OUTPUT);
}
void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Cricket Logo

```
to blink
forever [
          a, on
          wait 10
          a, off
          wait 10
        ]
end
```

Figure 4.2: A comparison of two programs that make an LED blink, in Arduino C and Cricket Logo.

but they have to use a much more technical language to, for example, "set pin 14 high."

A clear example of the consequences of different levels of selective exposure of the microcontroller — and how it decreases usability — is how students end up having to program such devices. In Figure 4.2, both pieces of code are meant to make an LED blink. In the Arduino programming language (Figure 4.2, left), we can observe how the complexities of the hardware design, such as exposing microcontroller pins directly to users, have important usability consequences. Not only do users have to pre-assign particular pins to their functions (outputs or inputs), but also pins are set to "high" and "low" instead of the more intuitive "on" and "off." More significantly, the technical terms such as "void" and "digitalWrite" make parsing the code much harder for novices. In comparison, Cricket Logo (Figure 4.2, right), which was designed for the MIT Cricket and also the LEGO Mindstorms kit, carries over the design decisions at the hardware level to the programming language: students can simply turn things "on" and "off," do not have to master unnecessarily complex programming concepts and terms, and the code is much more readable and understandable.

One can imagine that the writing of “digitalWrite (13, HIGH)” might trigger extraneous and hard questions that would get in the way of children building their projects: What is “digital write”? Why not only “write”? Are there other types of “write”? What am I writing to? Why the number 13 and not any other number? Why does HIGH mean “on”? These questions make sense for students digging deeper into electronics, robotics, and physical computing. They don’t make sense for students who are just getting started. Even if we were to introduce children to these concepts before actually using the toolkits, they would still sound quite extraneous to the job at hand, which raises the question of transfer of knowledge from the “kid” platforms to the “adult” platforms.

4.1.2 Exposure of the embedded electronics

A second criterion is the selective exposure of the embedded electronics. When connecting a sensor to a microcontroller, one might need to add resistors, amplification circuits, filters, or a power supply to the sensors. For motors, often there needs to be a driver circuit and overcurrent protection. Motors and sensors for the LEGO or Pico Cricket kits had all the circuitry enclosed and hidden in a plastic block; users did not need extra components or soldering, thus increasing usability. On the Arduino/BASIC Stamp side, however, none of those circuits are present; all the electronics are exposed to users who need to calculate the values for the resistors, find out the right components for the circuits, and build them. Despite the fact that the platform is, as a result, more flexible and powerful, this has important consequences for learnability: to simply make an LED blink (using the code from Figure 4.2), the hardware assembly would be radically different, as we can see in Figure 4.3. The Arduino circuit needs a breadboard, three jumper wires, one resistor, and the LED itself. The Pico Cricket circuit only needs the Pico Cricket board, a cable, and the light module — no extra components.

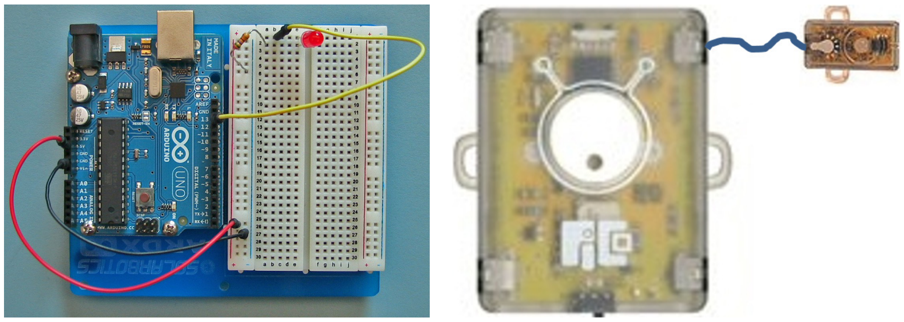


Figure 4.3: A comparison of two light-blinking circuits, using an Arduino (left) and Pico Cricket (right).

4.1.3 Other levels of selective exposure

A third category, beyond the "Arduino" and "Cricket" models, is composed of devices that took the idea of selective exposure even further — they enabled interaction at a different level of abstraction that broke the traditional paradigm of programmable bricks. Topobo, Braitenberg Blocks, and Cubelets — all programmable without computers — are among the main devices that fall into this category. They can be programmed autonomously either by example or by putting together their own "smart" tangible blocks; the parts are at the same time the programming medium and the physical objects sensing or being controlled. These platforms added new levels of selective exposure, since their designs further hide technical aspects of robotics and engineering. In Figure 4.4, we have an illustration of the different levels of exposure of these types of platforms: the Arduino-like devices expose the entire microcontroller but are harder to use. On the other end of the spectrum, Topobo and Cubelets expose only their smart modules, so they are easier to use but less flexible.

Now that I have explored in depth the selective exposure for two major items — the microcontroller and the embedded electronics — I will go into more detail concerning the connection between learning goals and selective exposure, using the various toolkits already mentioned. Typically, when children use physical computing and robotics

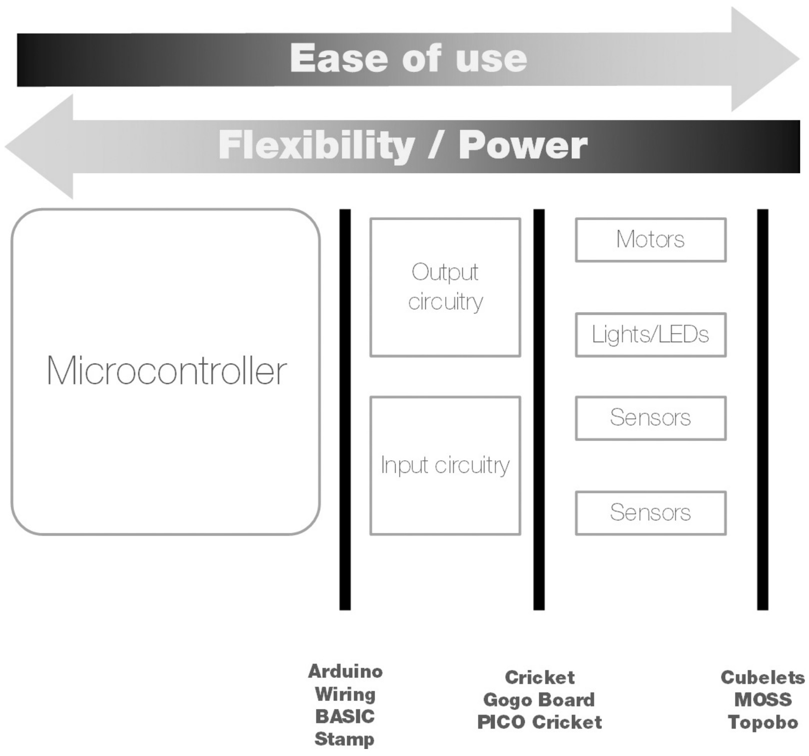


Figure 4.4: Selective exposure makes platforms easy to use, but alters the compromise between power and usability.

kits in school, there are eight very common learning goals (some of them present in the New Generation Science Standards), involving the understanding of:

(1) Resistors.

(2) Ohm's law (that is, the need to calculate current, resistance, voltage)

(3) Functionality of a breadboard.

(4) Polarity.

(5) Microcontroller pins.

(6) Electrical connections.

(7) Programming.

(8) Inputs and outputs.

If educators have limited time in classrooms, how do we choose platforms for each of these goals? The Arduino and BASIC Stamps platforms, for example, "expose" all eight elements to the user. Even very basic projects, such as making an LED blink, require a breadboard, resistors, special components, and some knowledge of electronics, which might overburden students and teachers with work that is not related to the original learning goal. Table 4.1 lists platforms and summarizes which elements are exposed or hidden. Four categories emerge from Table 4.1. Region A is populated by Arduino/Basic Stamp architectures that expose every one of the eight items. Region B abandons some aspects of this architecture: the Lilypad kit is provided with sensor and output boards with motor drivers, resistors, and all necessary electronic components, so there is no need to understand how to use breadboards or resistors. Region C has the classic Cricket-inspired devices and the Phidgets kit, which add an extra hardware abstraction layer to make the interaction easier: those platforms do not require students to think about polarity, microcontroller pins, or Ohm's law. Finally, region D adds an extra abstraction layer by allowing the components of the kits to be at the same time input-output devices and elements in their own programming — essentially, users only need to operate at the level of inputs and outputs.

4.2 Selective exposure for usability: Embedded error correction

One of the sub-constructs proposed in this monograph concerns how materials communicate the rules and best practices for construction. In other words, how does the toolkit "correct itself" and avoid certain types of user error, either by restricting actions or by adding cues to the design? Here I exemplify this idea by examining some common considerations a user must make when using physical computing platforms:

Table 4.1: Selective exposure of common platforms.

		Resistors	Ohm's law	Breadboard	Polarity	MC pins	Elec. connections	Programming	Inputs/ outputs	#
A	Arduino	Exposed	Exposed	Exposed	Exposed	Exposed	Exposed	Exposed	Exposed	8
	BStamp	Exposed	Exposed	Exposed	Exposed	Exposed	Exposed	Exposed	Exposed	8
B	LilyPad				Exposed	Exposed	Exposed	Exposed	Exposed	5
C	Phidgets						Exposed	Exposed	Exposed	3
	Cricket						Exposed	Exposed	Exposed	3
	GoGo						Exposed	Exposed	Exposed	3
	LEGO							Exposed	Exposed	2
	PicoCricket							Exposed	Exposed	2
D	Brai. Blocks								Exposed	1
	Topobo								Exposed	1
	Cubelets								Exposed	1
	LittleBits								Exposed	1

1. **How do the parts connect:** is it through wires, magnets, soldering, and so on?
2. **Valid connections:** which are the allowed connections between the parts?
3. **Categories of parts:** what are the types of parts and what do they do (sensors, motors, logic blocks, and so on)?
4. **Indications of functionality:** Can the user visually or physically infer functionality of the parts?

Examining the platforms discussed in this monograph, we again see a very clear progression from Arduino-like platforms to Cubelets. Thinking as a novice, it is hard to infer how the Arduino components have to be connected: with solder, wires, or jumper wires? The kit itself does not communicate how the connections should be made. It also does not enforce valid connections, so even connections that can damage the board are valid.

The LilyPad kit proposes a change by offering a clear indication of how the parts connect: with conductive thread. However, it does not clearly enforce valid connections, nor there is indication of the categories of parts except for the label in them. The Phidgets, the Cricket, and the GoGo Board have a different type of polarized connector that guides the user in making the correct connections. But more advanced design is found only in the LEGO and in the Pico Crickets kits, in which parts have different shapes and colors depending on their function (motor, sensor, light). The Lego connectors were designed to be connected in any position (Figure 4.5), and automatically correct any polarity mistake. Finally, kits such as Topobo, littleBits, and Cubelets are able to inform the user when a particular block is active, thus helping with debugging and understanding the program's flux. This feature is implemented by using connectors with extra terminals for an internal communication protocol. The littleBits connectors, shown in Figure 4.6, also have color-coded connectors with specially designed magnets that enforce the correct positions.

Table 4.2 summarizes the characteristics of all the platforms according to the self-error correction construct. We can observe the

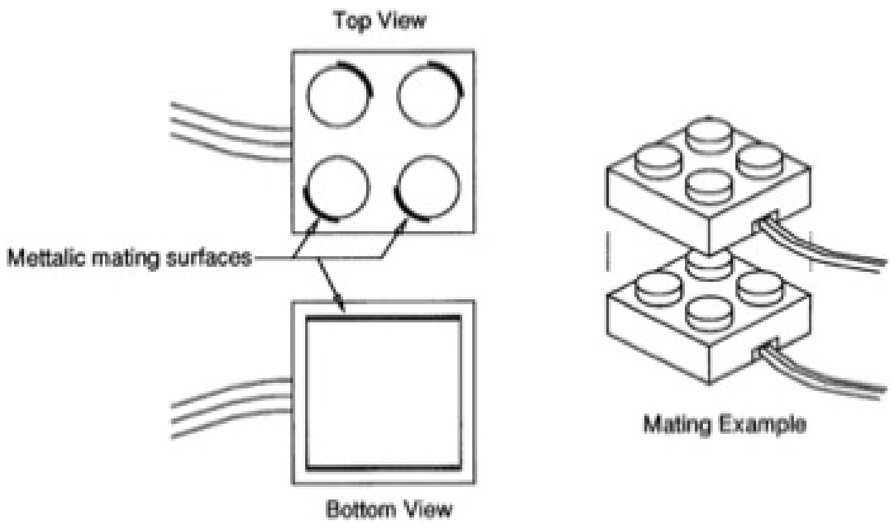


Figure 4.5: Sketches of the Lego connectors.

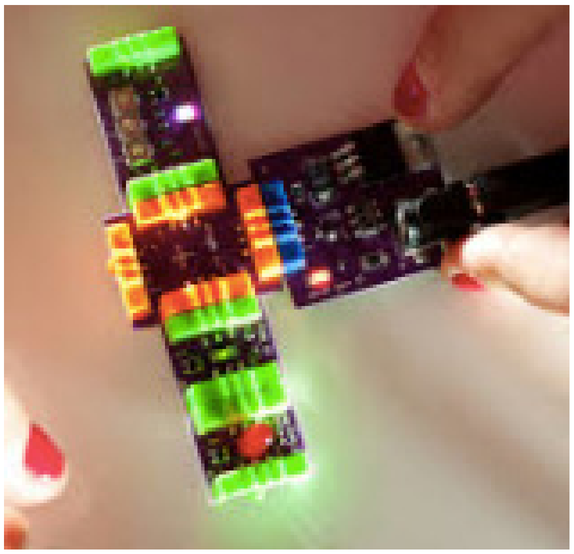


Figure 4.6: The littleBits platform, with magnetic “snap-on” connectors.

Table 4.2: Different levels of embedded error correction for physical computing.

	How to connect	Valid connections	Categories of parts	Indications of functionality	Count
Arduino					0
BStamp					0
LilyPad	Yes				1
Phidgets	Yes	Yes			2
Cricket	Yes	Yes			2
GoGo	Yes	Yes			2
LEGO	Yes	Yes	Yes		3
PicoCricket	Yes	Yes	Yes		3
Topobo	Yes	Yes	Yes	Yes	4
Cubelets	Yes	Yes	Yes	Yes	4
Brai Bl	Yes	Yes	Yes	Yes	4
LittleBits	Yes	Yes	Yes	Yes	4

progression and trend in recent platforms to adopt more embedded error correction and designs that try to better communicate to the users how its parts should be connected and used.

4.3 Selective exposure for power: Tangibility mapping

The last sub-construct that I propose is the *power* dimension of selective exposure. This looks at how the toolkit maps cognitive to physical operations in order to allow learners access to all the possibilities of the toolkit beyond the basics, thus "raising the ceiling" by restricting or foregrounding particular design elements. For example, a kit can have magnetic pieces to increase usability by avoiding polarity errors, as described in Section 4.2. However, if designers want to implement features to explicitly increase the power of the toolkit, they could map structure or hierarchy through the tangible pieces; instead of simply indicating polarity, the magnets could do more sophisticated tasks. For example, certain parts could "feel" or look like they should be inside other parts — for example, in tangible programming toolkits, blocks could fit easily inside a “repeat” block. Another example of indicating structure through the tangible pieces comes from Topobo, where the “brain” of the system (called “active”) is larger than the other pieces and has multiple connectors (Figure 4.7).

A second example of how tangibility mapping can increase the ceiling for students’ projects comes from the MOSS toolkit. MOSS is a toolkit created by Mark Gross and Eric Schweikardt at Modular Robotics around 2013, with a novel, physical-block-based design. The blocks have assigned functions and connect magnetically. The toolkit has several features to enforce error correction, but it also embeds several design elements that indicate to users more powerful kinds of possibilities. MOSS’s architecture has several types of blocks: power (green), data in (brown), data out (red), and pass-through (blue) (Figure 4.8).

The MOSS architecture enables users to quickly build robots that respond to simple sensors, such as proximity or light sensors. In Figure 4.9 we can see how two blocks “talk” to each other — users just connect the red face (data out) of the proximity sensor block to the brown face (data in) of the motor block. This feature successfully maps a desirable cognitive operation (connecting sensors and the actuator) to a physical activity. MOSS, however, has new tangibility mapping features. The “pass-through” block enables children to quickly move beyond the basics by connecting two sensors to a motor or two motors to a sensor (Figure 4.10). The fact that the “pass-through” block (the

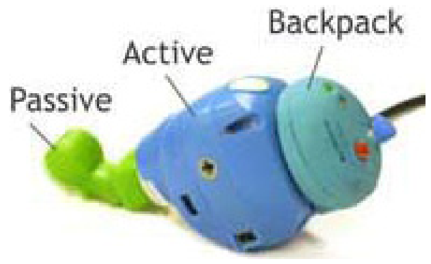
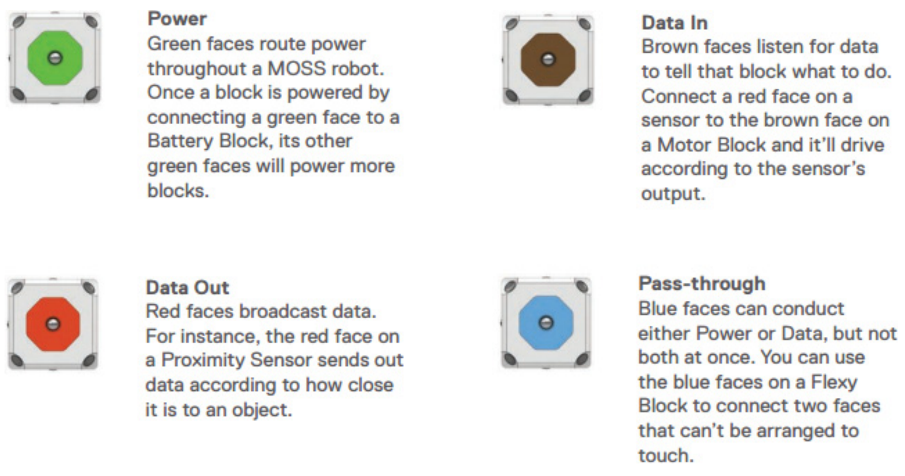


Figure 4.7: The “brain” of Topobo (blue), and other peripheral blocks — backpacks and passives.



Power
Green faces route power throughout a MOSS robot. Once a block is powered by connecting a green face to a Battery Block, its other green faces will power more blocks.

Data In
Brown faces listen for data to tell that block what to do. Connect a red face on a sensor to the brown face on a Motor Block and it'll drive according to the sensor's output.

Data Out
Red faces broadcast data. For instance, the red face on a Proximity Sensor sends out data according to how close it is to an object.

Pass-through
Blue faces can conduct either Power or Data, but not both at once. You can use the blue faces on a Flexy Block to connect two faces that can't be arranged to touch.

Figure 4.8: Types of blocks in the MOSS architecture.

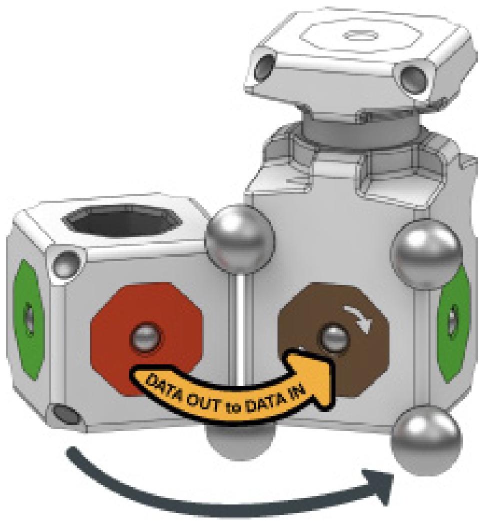


Figure 4.9: A simple MOSS connection.

blue one) has six faces automatically suggests that it can be connected to more than two blocks. Thus, the possibilities would be to connect two (or more) motors to a sensor (Figure 4.10, left) or two or more sensors to a single motor (Figure 4.10, right). If two motors are connected

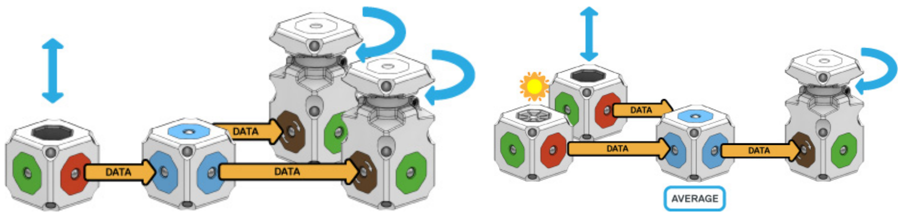


Figure 4.10: More sophisticated arrangements of MOSS blocks.

to the pass-through block, it "understands" that the signal from the sensor should be used to control both. If two sensors are connected, the pass-through block averages the two values and communicates it to the motor. Here, again, a desirable cognitive operation (combining inputs and outputs) is successfully mapped to tangible, physical actions, enabling users to build more complex projects without great technical complications. Imagine, for example, combining two sensors in an Arduino platform. Users would have to make the correct connections, normalize the data from the sensors, average the values, and rescale the resulting value to control the full rotation range of the motor. It would be virtually impossible for a user to "stumble" upon the possibility of averaging sensor values using Arduino-like platforms, but the opposite is be true for MOSS.

4.4 Other factors influencing selective exposure: Cost, audience, and production

Since the main focus of this monograph was education, I had to leave out some additional factors that also influence the design of physical computing platforms. Those factors are not inconsequential for education but they are less directly connected to issues of learning and cognition. One of those is cost. Differently from software, hardware needs to be manufactured and physically distributed. The way manufacturing works is determinant in what designers can and cannot produce for a reasonable cost. For example, adding a plastic enclosure to a circuit board can multiply the price of the final product several times, due to do the high cost of tooling for injection molding. Adding a manual task to the production workflow also increases costs significantly: if a connector has to be placed onto the circuit board by a person, rather than by a machine, the entire cost equation changes dramatically. Along the same lines, the production of circuit boards is optimized for miniaturization, so manufacturers would like to cramp as many components as possible in the smallest possible board. This might not be necessarily the best design for children, but the economic imperative might be, at times, more important.

Another factor that is fundamental is the intended reach of each toolkit. Toolkits that were born in research labs, initially intended for small audiences and for pilot studies, had the luxury of trying out more sophisticated designs and small-scale manufacturing techniques. Systems that were born already with a wide reach in mind had an extra constraint that needs to be taken into consideration. The cost equation of a commercial product such as a Lego kit is completely different from a platform being developed in the lab, so when making comparisons we need to be aware of those types of limitations.

Mentioning this additional design decisions is important since my goal is not to glorify or criticize the different platforms. Conversely, the point is that, for manufacturing a toolkit and deciding on the intended audience, designers and companies have very precise procedures. The financial impact of each design decision can be calculated with great accuracy. However, the learning impact of those design decisions is much harder to evaluate, in part because we do not have good frameworks for that analysis — and starting this discussion is the main goal of the remainder of this.

5

Discussion

This monograph focused on a review of computationally-enhanced toolkits for children and on the creation of a framework to better understand the useful dimensions for their design. This is particularly important today given the increased interest in programming and robotics in school environments, as well as the recent explosion of design creativity in the field fueled by lower production costs and crowdfunding.

Nevertheless, another important dimension in this discussion is less about design and technologies and is more philosophical. For example, several contemporary scholars and practitioners are debating if we should focus our energies on developing technologies for children, or on having children learn how to use adult technologies. In what follows, I discuss some of these questions.

5.1 Children's technologies for children, or adults' technologies for children?

From the early 1980s up to the present time, in the realm of computationally-enhanced toolkits for children, there has been a clear shift from *child-driven* to *professional-driven* design. The early programmable bricks were born out of a limited group of researchers and developmental psychologists who were interested in how children would utilize this new technology as an expressive medium. This mindset was connected to the research on computer programming and Logo, and since the actual first turtles were essentially simplified robotic devices, "in the 1980s, when microcontrollers were available, it was natural for Seymour Papert to dream of smart bricks" [Tinker, 2013]. In fact, there were three main lines of research around smart bricks: (1) children's engagement in design and engineering; (2) examining how students would build and program cybernetic, creature-like systems; and (3) the sense-making processes through which children would move forward during their construction of such systems [Martin, 2013].

These early stages of the research and development were heavy on usability and cognitive/developmental research (see, for example, Nira Grannot's Ph.D. dissertation, advised by Edith Ackermann, an impressive treatise on how children make sense of computational manipulatives [Granott, 1991]). One consequence of these foci was the understanding of the importance of selective exposure. All exposed and hidden elements of the design were intentional, despite the higher cost and greater complexity of manufacturing. The design heuristic was first to consider what should be foregrounded for children at a given developmental stage, and how to maximize the complexity of what they could build — and only then design the technology around it.

In 1992, when the first BASIC Stamp came out of Parallax, the inspiration was quite different. Parallax catered to hobbyists, and K-12 education was an afterthought. In 2001, Phidgets appeared on the market aimed at designers, engineers, and college students, and the Wiring platform, which came out in 2003, was yet another attempt to make designers' lives easier by making rapid prototyping modular and more approachable. The Arduino board, an offshoot of Wiring, shared those same design goals. This second lineage of products catered to hobbyists, artists, college students, and interaction designers. Consequently, they differed from the earlier lineage in their design commitments and compromises. Reflecting the spirit of the open source software movement, these designs were intended to make electronics more accessible and to bring the benefits of programming to the physical world — but there was no connection with interaction design for children or K-12 education.

The consequences of the ensuing design decisions were that most of these platforms used programming languages based on Java, C, or BASIC. Likewise, they required soldering, resistors, and breadboards, even for simple projects, and they were not easily made into autonomous devices (most did not have built-in batteries and were not robust enough to be mounted on top of robots or cars). With these new platforms, much less attention was given to selective exposure, embedded error correction, or tangibility mapping, since the main consideration was low-cost and flexibility — exposing all possible features at the lowest cost.

5.2 Physical computing for children: The rules of engagement

Another important point is about why non-child-centered tools ended up becoming popular in schools. After "professional" technologies such as Arduino boards became widely popular in schools, educators sought a justification for encumbering children with all the extra work that they required. The justification was that exposing students to professional engineering tools could both prepare students for jobs in engineering and boost interest in the career path; after all, how you can do robotics if you do not know resistors? Arguably, for engineers and hobbyists to do robotics properly, it is crucial to understand what resistors and capacitors are and how to calculate current, resistance, and voltage. From this perspective, it is unproblematic to expose children to this content and level of detail since this is what they will encounter when they decide to do "real" robotics.

However, there is a flaw in this argument. I argue that this shift in focus impeded the goal of exposing students to powerful ideas in robotics, computing, and cybernetics [Papert, 1980] because much more time had to be spent on the technicalities of making basic things work — connecting together the breadboard, motor drivers, jumper wires, and resistors, as well as understanding the syntax of Arduino

C code. Since these were the first set of problems that children would encounter, they consequently became the focus of the pedagogy and the curriculum. These technicalities were exactly what the previous generations of designers attempted to hide from students, because they ended up being considerable barriers for novices — with very little gain. This setback was unfortunately obfuscated by the huge popularity of Arduino-like devices. Thus, because the justification for learning physical computing has deep consequences for design choices, those choices need to be made clear from the beginning [Berland, 2015]. The design of a mere motor connector will be quite different if we are designing for vocational training in engineering or for personal expression through robotics. Since time is limited in schools for activities that are not in the core curriculum, these design decisions will ultimately decide if these tools will democratize engineering or further privilege a small group of students.

5.3 A developmental trajectory, or selective unveiling

This approach, which advocates exposing children to "professional" tools, also ignores the fact that rarely do we adopt such an approach in other areas of education. We do not introduce children to reading by giving them James Joyce's *Ulysses* or to music by asking them to play the harpsichord; we use specially designed materials and tools. When we need to transit children to professional tools, we do so carefully and also respect children's developmental stages. In classical music, it is accepted that the transition between a child's flute and the clarinet will require several hundreds of hours of coaching, training, and a carefully curated sequence of materials.

Thus, in other fields, we understand how to construct developmental trajectories to lead children to different levels of expertise. Instead of exposing children to inadequate technologies, a more productive approach would be to identify the many toolkits that children should use throughout their school years, understand what each can accomplish, and task designers with the creation of better *bridges between toolkits*. Better yet would be to design kits that "grow" with the child, unveiling layers of abstraction as they get mastered by students, and allowing for higher customization as children are ready for it. This idea of *selective unveiling* could be particularly powerful for physical computing kits because the design space is quite ample: there are infinite design layers between the act of soldering a transistor and plugging in a Lego sensor, as well as ways to transition between them. For example, a toolkit could start as a completely enclosed device with inputs and outputs, à la Lego Mindstorms. Once children exhaust the possibilities, a special button could open the plastic case and unveil the circuit board, which could then be removed, and used just like a Cricket or GoGo Board. Finally, once students need more flexibility, they could be instructed to remove the microcontroller and plug it into a breakout board similar to Arduino.

Unfortunately, even given the existence of many toolkits in the market, we do not make available the time, coaching, or tools to make these developmental trajectories happen in schools. The consequence is that we either alienate students who are not identified with STEM careers (as they become frustrated with inadequate tools), or we relegate students to strictly scripted 30-minute workshops that do not lead to any meaningful learning.

It would be significant to develop more toolkits at the beginning of the trajectory, and figure out the specifics of how these tools can scale within the constraints of the educational ecosystem: fitting well into curricula, school budgets, learners' time, attention, and interest.

6

Conclusion

The main construct proposed in this monograph (selective exposure) and its two subcategories (embedded error correction and tangibility mapping) could help understand the use of current products and give designers a framework to imagine new ones. Today's designs for children are again beginning to be informed by research developments stretching back to the early 1980s. In surveying the more recent literature, one notices a deep commitment to the constructionist ideas articulated by Papert and his colleagues. The interlude of the Arduino popularity surge, while problematic from a design standpoint, was perhaps a necessary step for physical computing for children to grow out of its roots and its several design experiments and reach out to the world. It appears that designers are now realizing that the work is far from done, and there are multiple opportunities to remix and reconceive the Cricket, the Braitenberg Blocks, and the Arduino technologies to create brand new ways to engage children in robotics, interactive arts, and cybernetics. Fortunately, there still appears to be a deep commitment to broadening participation in the field of computing, supporting many paths and learning trajectories, designing devices that can be integrated easily into schools, and exploring new materials and media. The ethos of physical computing seems to have shifted back from catering to a minority of hacker kids to offering opportunities for all children to make these devices, hopefully, the Papertian gears of their childhoods.

Acknowledgments

Thanks to Ben Shapiro, Matthew Berland, Richard Davis, Joel Sadler, and Lisa McIntire for their comments, ideas, and corrections. A special thanks to Arnan Sipitakiat–with whom I have created many microcontroller-based platforms over the years, including our lovechild, the Gogo Board–for his insightful suggestions and ideas.

This work is dedicated to the memory of my uncle, Moriz Blikstein, who gifted me my first copy of “Mindstorms” when I was a teenager.

References

H. Barragán. Wiring: Prototyping physical interaction design. *Interaction Design Institute,* Ivrea, Italy, 2004.

M. Berland, 2015. Personal communication.

I. Blikstein. Semiótica: uma ciência de… detetives. Revista USP, 16. pp. 161-166, 1993. http://dx.doi.org/10.11606/issn.2316-9036.v0i16p161-166.

L. Buechley and M. Eisenberg. Fabric PCBs, electronic sequins, and socket buttons: Techniques for e-textile craft. *Journal of Personal and Ubiquitous Computing*, 2007.

L. Buechley and B. Hill. Lilypad in the wild: How hardware's long tail is supporting new engineering and design communities. In *Proceedings of DIS '10*, pages 199–20. Aarhus, Denmark, 2010.

L. Buechley, N. Elumeze, and M. Eisenberg. Electronic/computational textiles and children's crafts. In *Proceedings Interactive Design and Children '06*, pages 49–56. Tampere, Finland, 2006.

L. Buechley, H. Perner-Wilson, and M. Satomi. Handcrafting textile interfaces from a kit-of-no-parts. In *Proceedings of TEI '11*. Funchal, Portugal, 2011.

M. Eisenberg, A. Eisenberg, M. Gross, K. Kaowthumrong, N. Lee, and W. Lovett. Computationally-enhanced construction kits for children: Prototypes and principles. In *Proceedings of the Fifth International Conference of the Learning Sciences*, pages 23–26, 2002.

P. Frei, V. Su, B. Mikhak, and H. Ishii. Curlybot: Designing a new class of computational toys. In *Proceedings of CHI2000*. ACM Press, 2000.

P. Goldenberg. Fastr — a simple turtle runner, in: Logo working paper no. 30 (mit ai lab, massachussets institute of technology, cambridge, ma, usa), 1974.

N. Granott. Microdevelopment of co-construction of knowledge during problem solving: Puzzled minds, weird creatures, and wuggles. PhD. Dissertation, Massachusetts Institute of Technology. Cambridge. http://dspace.mit.edu/handle/1721.1/29069, 1991.

S. Greenberg and C. Fitchett. Phidgets: Easy development of physical interfaces through physical widgets. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*. Orlando, Florida, 2001.

E. Hamner, T. Lauwers, and D. Bernstein. The debugging task: Evaluating a robotics design workshop. *Association for the Advancement of Artificial Intelligence*, pages 20–25, 2010.

C. Lyon. Encouraging innovation by engineering the learning curve. Master's Thesis, Massachusetts Institute of Technology. Cambridge, 2003.

F. Martin. Children, cybernetics, and programmable turtles. Master's thesis. Cambridge: Massachusetts Institute of Technology, 1988.

F. Martin, 2013. Personal communication.

F. Martin and A. Chanler. Introducing the blackfin handy board. *American Association for Artificial Intelligence*, 2007.

F. Martin and L. Xu. Chirp on crickets: Teaching compilers using an embedded robot controller. In *Proceedings of SIGCSE '06*, pages 82–86. Houston, Texas, USA, 2006.

F. Martin, B. Mikhak, and B. Silverman. Metacricket: A designer's kit for making computational devices. *IBM Systems Journal*, 39(3–4):795–815, 2000.

T. McNerney. Tangible computation bricks: Building-blocks for physical microworlds. *Proceedings of CHI 2011*, 2000.

T. McNerney. From turtles to tangible programming bricks: Explorations in physical language design. *Personal and Ubiquitous Computing*, 8(5):326–337, 2004. (2004/09/01) DOI = http://dx.doi.org/10.1007/s00779-004-0295-6.

S. Papert. *Mindstorms: Children, computers, and powerful ideas.* New York: Basic Books, 1980.

R. Perlman. TORTIS — Toddler's Own Recursive Turtle Interpreter System, in: MIT AI Memo No. 311/Logo Memo No. 9 (MIT AI Lab, Massachusetts Institute of Technology, Cambridge, MA, USA), 1974.

H. Raffle, A. Parkes, and H. Ishii. Topobo: A constructive assembly system with kinetic memory. In *Human Factors in Computing (CHI) '04*. ACM, 2004.

J. J. Ramos, H. Azevedo, V. A. J. Vilhete, O. Noves, D. Figueiredo, L. Tanure, and F. Holanda. Iniciativa para robótica pedagógica aberta e de baixo custo para inclusão social e digital no brasil. In *Anais do VIII Simpósio Brasileiro de Automação Inteligente (SBAI 2007)*. Florianópolis, SC, 2007.

M. Resnick and S. Ocko. LEGO/Logo: Learning through and about design. In I. Harel and S. Papert, editors, *Constructionism*, Norwood, NJ: Ablex Publishing, 1991.

M. Resnick and B. Silverman. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 Conference on Interaction Design and Children (IDC '05)*, pages 117–122. ACM, New York, NY, USA, 2005.

M. Resnick, F. Martin, R. Sargent, B. Silverman, R. Berg, M. Eisenberg, S. Turkle, and F. Martin. Programmable bricks: Toys to think with. *IBM Systems Journal*, 35(3–4):443–452, 1996.

M. Resnick, F. Martin, R. Berg, R. Borovoy, V. Colella, K. Kramer, and B. Silverman. Digital manipulatives: New toys to think with. In *Proceedings of CHI '98, (Los Angeles, April 1998)*, pages 281–287. ACM Press, 1998.

M. Resnick, R. Berg, and M. Eisenberg. Beyond black boxes: Bringing transparency and aesthetics back to scientific investigation. *Journal of the Learning Sciences*, 9(1):7–30, 2000.

N. Rusk, M. Resnick, R. Berg, and M. Pezalla-Granlund. New pathways into robotics: Strategies for broadening participation. *Journal of Science Education and Technology*, 17(1):59–69, 2008.

E. Schweikardt and M. Gross. roBlocks: A robotic construction kit for mathematics and science. In *Proceedings of ICMI '06*. Banff, Alberta, Canada, 2006.

E. Schweikardt and M. Gross. A brief survey of distributed computational toys. *The First IEEE Intl Workshop on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL '07)*, pages 57–64, 2007.

D. W. Shaffer and M. Resnick. "Thick" authenticity: New media and authentic learning. *Journal of Interactive Learning Research*, 10(2):195–215, 1999.

B. Shapiro, 2015. Personal communication.

A. Sipitakiat and P. Blikstein. Think globally, build locally: A technological platform for low-cost, open-source, locally-assembled programmable bricks for education. In *Proceedings of TEI 2010*, pages 231–232. Boston, Massachusetts, USA, 2010.

A. Sipitakiat and N. Nusen. Robo-blocks: Designing debugging abilities in a tangible programming system for early primary school children. In *Proceedings of IDC '12*, pages 98–105, 2012.

A. Sipitakiat, P. Blikstein, and D. Cavallo. The GoGo Board: Moving towards highly available computational tools in learning environments. *Interactive Computer Aided Learning International Conference,* Villach, Austria, 2002.

A. Sipitakiat, P. Blikstein, and D. Cavallo. The GoGo Board: Augmenting programmable bricks for economically challenged audiences. In *Proceedings of ICLS 2004*, pages 481–488. USA, 2004.

C. Solomon and S. Papert. Teach: A step toward more interactive programming, in: Logo working paper no. 43 (mit ai lab, massachussets institute of technology, cambridge, ma, usa), 1975.

B. Tinker, 2013. Personal communication.

S. Wilson, M. Gurevich, B. Verplank, and P. Stang. Microcontrollers in music HCI instruction: Reflections on our switch to the atmel AVR platform. Paper presented at the Proceedings of the 2003 conference on New interfaces for musical expression, Montreal, Quebec, Canada, 2003.

P. Wyeth and G. F. Wyeth. Electronic blocks: Tangible programming elements for preschoolers. In *IFIP TC. 13 International Conference on Human-Computer Interaction*, pages 496–503. IOC Press, 2001.

O. Zuckerman, S. Arida, and M. Resnick. Extending tangible interfaces for education: Digital montessori-inspired manipulatives. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*, pages 859–868. ACM, New York, NY, USA, 2005. DOI: http://dx.doi.org/10.1145/1054972.1055093.