Tangible Programming in Early Childhood:

Revisiting Developmental Assumptions through New Technologies

Marina Umaschi Bers and Michael S. Horn

Tufts University

Abstract

This chapter explores the idea that when given appropriate tools, young children can actively engage in computer programming and robotics activities in a way that is consistent with developmentally appropriate practice. In particular, this project proposes the use of emerging *tangible user interface* (TUI) technology to fundamentally re-envision the way in which children program computers. In short, rather than using a mouse or keyboard to write programs to control robots, children instead construct physical computer programs by connecting interlocking, “smart” wooden blocks. In this chapter we will describe our development efforts both in terms of curriculum and technology. We will also share results from a two-year design-based research study conducted in three kindergarten classrooms.

Tangible Programming in Early Childhood:

Revisiting Developmental Assumptions through New Technologies

We are surrounded by technology. From pens and pencils to cell phones and digital cameras, technology permeates our existence. Yet, in the early grades, children learn very little about this. For decades early childhood curriculum has focused on literacy and numeracy, with some attention paid to science, in particular to the natural world—insects, volcanoes, plants, and the Arctic. And, while understanding the natural world is important, developing children's knowledge of the surrounding man-made world is also important (Bers, 2008). This is the realm of technology and engineering, which focus on the development and application of tools, machines, materials, and processes to help solve human problems.

Early childhood education hasn't ignored this; it is common to see young children using cardboard or recycled materials to build cities and bridges. However, what is unique to our man-made world today is the fusion of electronics with mechanical structures. We go to the bathroom to wash our hands, and the faucets "know" when to start dispensing water. The elevator "knows" when someone's little hands are in between the doors and shouldn't close. Our cell phones "know" how to take pictures, send emails, and behave as alarm clocks. Even our cars "know" where we want to go and can take us there without getting lost. We live in a world in which bits and atoms are increasingly integrated (Gershenfeld, 2000); however, we do not teach our young children about this. In the early schooling experiences, we teach children about polar bears and cacti, which are probably more remote from their everyday experience than smart faucets and cellular phones. There are several reasons for the lack of focus on technologies in early childhood, but two of the most common claims are that young children are not developmentally ready to understand such complex and abstract phenomena, and that there is a lack of technology with age-appropriate interfaces that allow children to develop their own technologically-rich projects. In this chapter we address both of these claims by presenting research to evaluate young children's ability to build, program, and understand their own robotic creations through the use of a novel programming interface designed specifically for young children

Background

Our work is rooted in notions of developmentally appropriate practice (DAP), a perspective within early childhood education concerned with creating learning environments sensitive to children's social, emotional, physical, and cognitive development. DAP is a framework produced by the National Association for the Education of Young Children (Copple & Bredekamp, 2009) that outlines practice that promotes young children's optimal learning and development. DAP is based on theories of child development, the strengths and weaknesses of individual children uncovered through authentic assessment, and individual children's cultural background as defined by community and family (Copple & Bredekamp, 2009).

DAP is built upon the theory of developmental stages introduced by Jean Piaget, which suggests that children enter the *concrete operational* stage at age six or seven. According to Piaget, at this age, a child gains the ability to perform mental operations in his/her head and also to reverse those operations. As a result, a concrete operational child has a more sophisticated understanding of number, can imagine the world from different perspectives, can systematically compare, sort, and classify objects, and can understand notions of time and causality (Richardson, 1998). Based on this developmental model, one might make the argument that a child's ability to program a computer might be predicted by his or her general developmental level, and, that by extension, a pre-operational kindergartener (typically five years old) may be too young to benefit from or understand computer programming. However, since its introduction, various problems and inconsistencies have been identified with Piaget's stage model. For example, studies have shown that when a task and its context are made clear to children, they exhibit logical thought and understanding well before the ages that Piaget suggested as a lower limit (Richardson, 1998).

In the early days of personal computing, there was lively debate over the developmental appropriateness of computer technology use in early elementary classrooms (Clements & Sarama, 2003). Today, however, the pressing question in no longer *whether* but *how* we should introduce computer technology in early elementary school (Clements & Sarama, 2002). For example, a 1992 study found that elementary school children exposed to exploratory software showed gains in self-esteem, non-verbal skills, long-term memory, manual dexterity, and structural knowledge. When combined with other non-computer activities, these students also showed improvements in verbal skills, abstraction, problem solving, and conceptual skills (Haugland, 1992). Other studies have demonstrated that computer use can serve as a catalyst for positive social interaction and collaboration (Clements & Sarama, 2002; Wang & Ching, 2003). Of course, the developmental appropriateness of the technology used by young children depends on the context. *What software is being used? And how is it integrated with and supported by the broader classroom curriculum?* Fortunately, we live in an advantageous time for introducing technology in early childhood. Given the increasing mandate to make early childhood programs more academically challenging[1], technology can provide a playful bridge to integrate academic demands with personally meaningful projects (Bers, 2008).

Robotics and Computer Programming in Early Childhood Education

"Computer technology" is a broad term that can mean many things, especially in the context of a classroom. We believe that robotics is one type of educational technology that holds special potential for early childhood classrooms where children engage in cognitive as well as motor and social skills development. Furthermore, in early childhood content areas tend not to be isolated, but integrated more broadly into classroom curriculum that encompasses different content and skills; learning can be project-driven and open-ended; and student work does not have to fit into an hour-long class period. Thus, robotics can be a good integrator of curricular content (Bers, Ponte, Juelich, Viera, & Schenker, 2002).

Robotics provides opportunities for young children to learn about sensors, motors, and the digital domain in a playful way by building their own projects, such as cars that follow a light, elevators that work with touch sensors, and puppets that can play music. Young children can become engineers by playing with gears, levers, motors, sensors, and programming loops, as well as storytellers by creating their own meaningful projects that move in response to their environment (Bers, 2008; Wang & Ching, 2003). Robotics can also be a gateway for children to learn about applied mathematical concepts, the scientific method of inquiry, and problem solving (Rogers & Portsmore, 2004). Moreover, robotic manipulatives invite children to participate in social interactions and negotiations while playing to learn and learning to play (Resnick, 2003).

[1] As of 2006, thirty-seven states have included engineering/technology standards in their educational frameworks.

Robotics, however, is about more than just creating physical artifacts. In order to bring robots to “life” children must also create computer programs—digital artifacts that allow robots to move, blink, sing, and respond to their environment. Previous research has shown that children as young as four years old can understand the basic concepts of computer programming and can build and program simple robotics projects (Bers, 2008; Cejka, Rogers, & Portsmore, 2006; Bers, Rogers, Beals, Portsmore, Staszowski, Cejka, Carberry, Gravel, Anderson, & Barnett, 2006). Furthermore, early studies with the text-based language, Logo, have shown that computer programming, when introduced in a structured way, can help young children with variety of cognitive skills, including basic number sense, language skills, and visual memory (Clements, 1999).

Nonetheless, computer programming is difficult for novices of any age. Kelleher and Pausch (2005) offer a taxonomy containing well over 50 novice programming systems, a great number of which aim to ease or eliminate the process of learning language syntax, perhaps the most often cited source of novice frustration. Beyond syntax, there are many specific conceptual hurdles faced by novice programmers as well as fundamental misconceptions about the nature of computers and computer programming. Ben-Ari (1998) points out that unlike the beginning physics student who at least has a naïve understanding of the physical world, beginning programmers have no effective model of a computer upon which to build new knowledge. Worse, rarely is an effort made to help students develop a working model. According to Norman (1986), the primary problem facing novice programmers is the gap between the representation the brain uses when thinking about a problem and the representation a computer will accept.

In addition to the above challenges faced by novice programmers, we must also consider the developmental needs and capabilities of young children. McKeithen, Reitman, Rueter, and Hirtle (1981) conducted a study that explored the differences in the ability of expert and novice computer programmers to recall details of computer programs. In their analysis, they theorize that because novice programmers lack adequate mental models for programming tasks, they rely on rich common language associations for these concepts. For example, computer words like LOOP, FOR, STRING, and CASE have very different common language meanings. Reflecting on this result, it seems reasonable to expect that young children will have an especially difficult time building conceptual models for programming concepts because they have fewer mental schemas on which to build.

Likewise Rader, Brand, and Lewis (1997) conducted a study with the Apple's KidSim programming system in which 2nd/3rd graders and 4th/5th graders used the system for one year with minimal structured instruction. At the end of the year, the younger children had significantly more difficulty with programming concepts such as individual actions, rule order, and subroutine. However, the authors suggest that with structured instruction of programming concepts, the young children would have developed a much better understanding of the system.

However, regardless of the way in which the content is presented, most current programming environments are not well-suited for very young children. One problem is that the syntax of text-based computer languages, such as Logo, can be unintuitive and frustrating for novice programmers. This is exacerbated for young children who are still learning how to read. Modern visual programming languages such as ROBOLAB[2] allow children to program by dragging and connecting icons on the computer screen. And, while this approach simplifies

[2] http://www.legoengineering.com/

language syntax, the interfaces require young children to use a mouse to navigate hierarchical menus, click on icons, and drag lines to very small target areas on a computer screen. All of this requires fine motor skills that make it difficult for young children to participate (Hourcade, Bederson, Druin, & Guimbretière, 2004). As a result, adults often have to sit with young children and give *click-by-click* instructions to make programming possible, which poses challenges for children's learning (Beals & Bers, 2006). It also makes it difficult to implement computer programming in average schools, where there are often only one or two adults per twenty-five children. Attempts have been made to create simpler versions of these languages. However, the resulting interfaces often obscure some of the most important aspects of programming, such as the notion of creating a sequence of commands to form a program's flow-of-control. In the next section we will present work on tangible computer programming that explores new interfaces to address some of the challenges presented here.

Tangible Computer Programming

With the emergence of tangible user interface technology (TUI), we have a new means to separate the intellectual act of computer programming from the confounding factor of modern graphical user interfaces. And, with this, we have an opportunity to build a much better understanding of developmental capabilities of young children with respect to computer programming. Just as young children can read books that are appropriate for their age-level, we propose that young children can write simple but interesting computer programs, provided they have access to a developmentally-appropriate programming language. Indeed, in our own extensive previous work with robotics and young children, we observed that when presented with new technologies that make use of well-established sensori-motor skills, young children are able to display complex mental operations.

We believe that we can overcome the inherent limitations of modern desktop and laptop computers by doing nothing short of removing them from children's learning experiences. Thus, rather than write computer programs with a keyboard or mouse, we have created a system that allows children to instead *construct* physical computer programs by connecting interlocking wooden blocks (see Figure 1). This technique is called *tangible programming*.

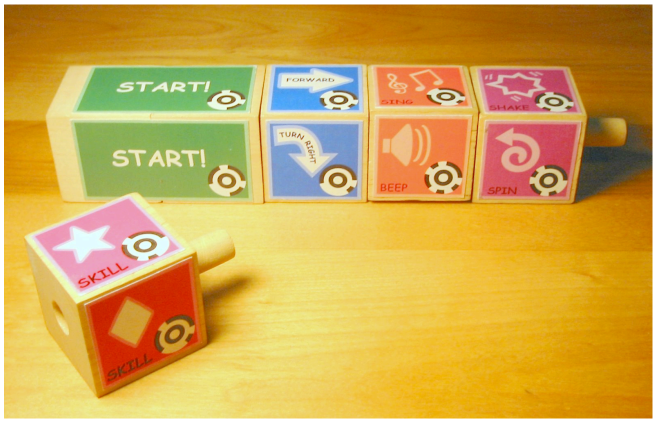

Figure 1. A tangible programming language for robotics developed at Tufts University. With this language, children construct programs using interlocking wooden blocks.

A tangible programming language, like any other type of computer language, is simply a tool for telling a computer what to do. With a text-based language, a programmer uses words such as BEGIN, IF, and REPEAT to instruct a computer. This code must be written according to strict, and often frustrating, syntactic rules. With a visual language, words are replaced by pictures, and programs are expressed by arranging and connecting icons on the computer screen with the mouse. There are still syntax rules to follow, but they can be conveyed to the programmer through a set of visual cues.

Instead of relying on pictures and words on a computer screen, tangible languages use physical objects to represent the various aspects of computer programming. Users arrange and connect these physical elements to construct programs. Rather than falling back on implied rules and conventions, tangible languages can exploit the physical properties of objects, such as size, shape, and material to express and enforce syntax. For example, the interlocking wooden blocks shown in Figure 1 describe the language syntax (i.e. a sequential connection of blocks). In fact, with this language, while it is possible to make mistakes in program logic, it is impossible to produce a syntax error.

In moving away from the mouse-based interface, our pilot studies, conducted in public schools in the Boston area, have suggested that tangible languages might have the added benefit of improving both the style and amount of collaboration occurring between students. And, since the process of constructing programs is now situated in the classroom at large—on children's desks or on the floor—children's programming work can be more open and visible and can become more a part of presentations and discussions of technology projects. Likewise, physical programming elements can be incorporated into whole-class instruction activities without the need for a shared computer display or an LCD projector.

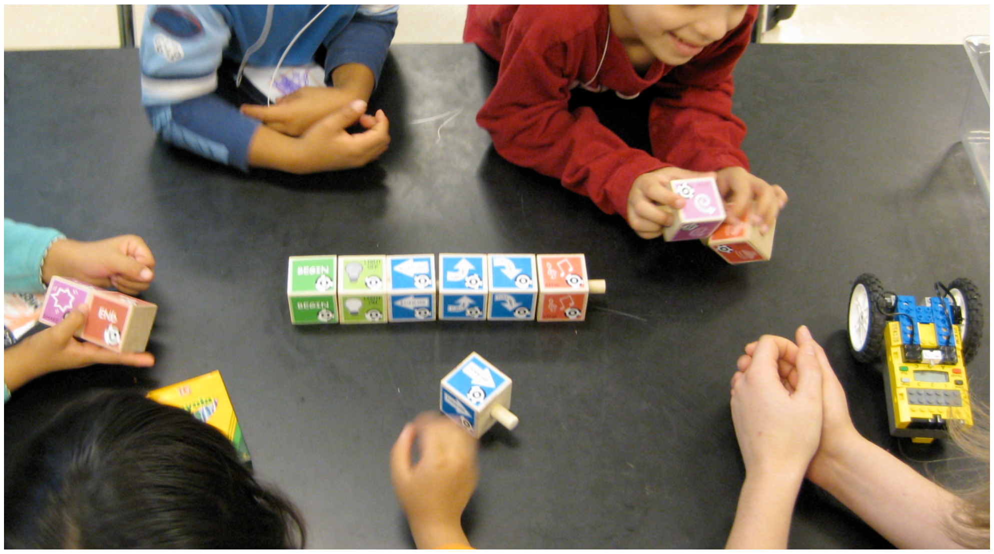

Figure 2. This picture taken from a pilot study show kindergarten students using the tangible programming language developed at Tufts to program a robot to act out a short story.

The idea of tangible programming was first introduced in the mid 1970's by Radia Perlman, then a researcher at the MIT Logo Lab. Perlman believed that the syntax rules of text-based computer languages represented a serious barrier to learning for young children. To address this issue she developed an interface called *Slot Machines* (Perlman, 1976) that allowed young children to insert cards representing various Logo commands into three colored racks. This idea of tangible programming was revived nearly two decades later (Suzuki & Kato, 1995), and since then a variety of tangible languages have been created in a number of different research labs around the world (e.g., McNerney, 2004; Wyeth, 2008; Smith, 2007).

In almost all cases, the blocks that make up tangible programming languages contain some form of electronic components. And, when connected, the blocks form structures that are more than just abstract representations of algorithms; they are also working, specialized computers that can execute algorithms through the sequential interaction of the blocks.

Unfortunately, the blocks that make up these languages tend to be delicate and expensive. And, as a result, tangible programming languages have seldom been used outside of research lab settings. At Tufts we have taken a different approach (Horn & Jacob, 2007). Programs created with our language are purely symbolic representations of algorithms—much in the way that Java or C++ programs are only collections of text files. An additional piece of technology must be used to translate the abstract representations of a program into a machine language that will control a robot. This approach allows us to create inexpensive and durable parts, and provides greater freedom in the design of the physical components of the language. Our current prototype uses a collection of image processing techniques to convert physical programs into digital instructions. For the system to work, each block in the language is imprinted with a circular symbol called a TopCode (Horn, 2008). These codes allow the position, orientation, size and shape, and type of each statement to be quickly determined from a digital image. The image processing routines work under a variety of lighting conditions without the need for human calibration. Our prototype uses a standard consumer web camera connected to a laptop computer. Children initiate a compile by pressing the spacebar on the computer, and their program is downloaded onto a robot in a mater of seconds.

Tangible Programming Curriculum for Kindergarten

Research has shown that mere exposure to computer programming in an unstructured way has little demonstrable effect on student learning (Clements, 1999). For example, a 1997 study involving the visual programming language, KidSim*,* found that elementary school students failed to grasp many aspects of the language, leading the authors to suggest that more explicit instruction might have improved the situation (Rader et al., 1997). Therefore, an important aspect of our work is to develop curriculum that utilizes tangible programming to introduce a series of powerful ideas from computer science in a structured, age-appropriate way. The term *powerful idea* refers to a concept that is at once personally useful, interconnected with other disciplines, and rooted in intuitive knowledge that a child has internalized over a long period of time (Papert, 1991). We introduce these powerful ideas in a context in which their use allows very young children to solve compelling problems.

Table 1 lists example powerful ideas from computer programming that we have selected to emphasize in our curriculum using tangible programming. Based on these powerful ideas we developed a preliminary 12-hour curriculum module for use in Kindergarten classrooms that introduces a subset of these concepts through a combination of whole-class instruction, structured small-group challenge activities, and open-ended student projects. We piloted this curriculum in three Kindergarten classrooms in the Boston area, using the tangible programming blocks described above and LEGO Mindstorms RCX construction kits. Initially, we provided students with pre-assembled robot cars to teach preliminary computer programming concepts. As the unit progressed, students disassembled these cars to build diverse robotic creations that incorporated arts and craft materials, recycled goods such as cardboard tubes and boxes, and a limited number of LEGO parts. Table 2 provides a brief overview of the activities that we included in the curriculum.

Table 1

Powerful ideas from computer programming and robotics emphasized in our curriculum

Powerful Idea	**Description**
Computer Programming	This is the fundamental idea that robots are not living things that act of their own accord. Instead, robots act out computer programs written by human beings. Not only that, children can create their own computer programs to control a robot.
Command Sequences & Control Flow	The idea that simple commands can be combined into sequences of actions to be acted out by a robot in order.
Loops	The idea that sequences of instructions can be modified to repeat indefinitely or in a controlled way.
Sensors	The idea that a robot can sense its surrounding environment through a variety of modalities, and that a robot can be programmed to respond to changes in its environment.
Parameters	The idea that some instructions can be qualified with additional information.
Branches	The idea that you can ask a question in a program, and, depending on the answer, have a robot do one thing or another.
Subroutines	The idea that you can treat a set of instructions as a single unit that can be called from other parts of a program.

Table 2

Kindergarten Computer Programming and Robotics Curriculum Activities

Robot Dance: In this introductory activity, we introduce students to the concept of robots and programming. Children participate in a whole-class activity in which they use the tangible blocks to "teach" a robot to dance the *Hokey-Pokey.*
Experimenting with Programming: Students work in small groups to create and test several programs of their own design. At this point they combine simple command blocks (no sensors or control flow blocks yet) into sequences of actions that are acted out by the group's robot. Robots can perform simple movements, make sounds, blink a light, and perform dance moves (such as shake or spin around).
Simon Says: In this whole class activity, students pretend that they are robots and act out commands presented by the teacher. For example, the teacher holds up a card that says *SHAKE,* and the students all shake their bodies. The point of this activity is to help students, who may or may not be able to read, learn the various programming commands that are available to use. As this activity progresses, the teacher will begin to display two or more cards at the same time that the students will act out in order. This activity is repeated on several occasions throughout the unit.
Hungry, Hungry Robot: In this challenge activity, we introduce the idea of loops as students work in small groups to program their robot to move from a starting line (a strip of tape on the floor of the classroom) to a finish line (marked with a picture of cookies). Children are prompted with the challenge: *Your robot is very hungry. Can you program it to move from the start line to the cookies?* As this activity progresses, students quickly realize that the robot

needs to move forward several times in succession to reach its goal. Because they are only provided with a small number of blocks to move the robot forward, they learn to create programs that incorporate a simple counting loop.
If you’re happy and you know it: The teacher reads a story in which different animals show that they are happy in different ways. Students are then asked to work in small groups to program a robot to act like one of the animals in the story.
Bird in a Cage: In this whole class activity, the teacher tells a story about a bird that sings when it is released from a cage. The “bird” is a robot with a light sensor attached. When the bird is removed from a cardboard box (the cage), the increase in light intensity triggers it to sing. The teacher then introduces the concept of sensors and shows the class how to write the program that causes the bird to sing. This activity is repeated on two separate occasions, once as a whole-group activity, and once as a small group challenge activity.
Final Projects: Students work in teams of two to design and build a robot animal that includes one motor and one sensor. Students are encouraged to create a story that involves their robot animal. This activity is open-ended and student projects are completed over a period of several days.

Research Questions

The research presented in this chapter is guided by the following questions: given access to appropriate technologies and curriculum, are young children capable of learning how to program their own robotics projects? How much direct adult assistance do they need? Can young children understand the underlying powerful ideas behind computer programming? Finally, should we revisit assumptions about what is developmentally appropriate for young children when designing curriculum—can we leverage new technologies to introduce more complex concepts?

These ideas are not entirely new. Researchers at MIT's Lifelong Kindergarten Group have suggested that with new *digital manipulatives* children are capable of exploring concepts that were previously considered too advanced, in part because of the ability of technology to bring abstract concepts to a concrete level (Resnick, Martin, Berg, Borovoy, Colella, Kramer, & Silverman, 1998). However, for the most part, this claim has not been empirically validated with children as young as five years old. And, as noted above, it can be difficult in this research to distinguish what young children truly understand and can do on their own, versus what was done by the supporting adults (Cejka et al., 2006). In the end, the functioning robotic projects tend to hide the challenges faced by children in the process of making them.

Methodological Approach: Design-Based Research

Design-based research (DBR) is an approach to studying novel tools and techniques for education in the context of real-life learning settings (Design Based Research Collective, 2003). As a methodology, DBR acknowledges the substantial limitations of conducting research in chaotic environments like classrooms; however, in exchange researchers hope that the resulting designs will be effective and practical for future use in classrooms. DBR is based on an iterative process of design, evaluation, and testing. By collecting both qualitative and quantitative data in non-laboratory settings, design-based research has the additional goal of developing theories of learning that will inform future research.

Study

We have conducted a two-year design-based research study exploring the use of tangible programming technology in Kindergarten classrooms. Both the curriculum and the technology have evolved substantially as a result of each iteration. This research was conducted at two public schools in the greater-Boston area, one urban K-8 school and one suburban K-5 school. In all, we have worked with 56 children and six teachers. Data were collected through observation notes, photographs, and video tape. We also collected student work (both the programming code as well as the robotic artifacts) and conducted one-on-one interviews with children and teachers in the classroom. Our first intervention consisted of two sessions (four hours total), with 20 students and two teachers at the urban school. Afterwards, we made substantial revisions to the technology (described in detail below) and expanded the curriculum. Our second intervention consisted of a eight-hour curriculum module in two classrooms at the suburban school with 36 children and four teachers.

Results

In this study, we had two primary hypotheses. First, given access to appropriate technology, young children are capable of programming their own robotics projects without direct adult assistance. In other words, through a combination of unstructured exploration, structured activities, and scaffolding, children will learn how to build and program indendently.

Second, children are able to understand the underlying powerful ideas of computer programming and robotics through the curriculum.

Based on observation notes and an analysis of video tape, we found that children were able to easily manipulate the tangible blocks to form their own programs. And, for the most part, students were also able to differentiate the blocks and discern their meanings. Not all of the children could read the text labels on the blocks, although we saw evidence that children who could not read the blocks were able to use the icons as a way to interpret meaning. For example, in the initial sessions, we asked children to look at the blocks and describe what they saw. Some children were simply able to read the text labels on the blocks. Other children said things like: “mine says arrow” or “mine says music.” In reality the text on the blocks reads “Forward” and “Sing.” We used the *Simon Says* activity to reinforce the meaning of each block in terms of the robot’s actions.

The children also seemed to understand that the blocks were to be connected in a sequence and interpreted from left to right. For example, one student in the introductory session pointed out, “they connect; some have these [pegs] and some have these [holes].” Another student added, “the End block doesn’t have one [peg],” with another classmate adding, “the Start block doesn’t have a hole.” The blocks were designed using physical form factor to enforce language syntax. The students noticed that blocks could not be inserted into a program before the Start block or after the End block. During the second session, researchers asked individual children to describe their programs. One girl explained, “my program is: Begin *then* Beep *then* Forward *then* Sing *then* End.” Here she was reading the text labels on the blocks in her program from left to right, resting her finger on each block in turn. The use of the word *then* suggests that she might have understood not only the implied left-to-right ordering of the blocks but also the temporal step-by-step sequence in which the blocks would be executed. Understanding the idea of sequencing is powerful, not only in the domain of computer programming but for most analytical activities that children would encounter in schooling as well as life.

In the second intervention round, we moved beyond programs consisting of simple sequences of actions and introduced more advanced constructs including loops, sensors, and numeric parameter values. Through these activities we found evidence that children could engage these concepts, reasoning about possible outcomes of different blocks and forming and testing solutions to challenge activities. For example, in our *Hungry, hungry robot* activity, we prompted teams of four students with this challenge: *Your robot is hungry. Can you program it to drive to the cookies?* We had placed the robot on a starting line, represented with a strip of tape on the floor. The *cookies* were represented with a picture taped on the wall approximately two meters from the starting line. Groups were provided with a small collection of blocks, including two Forwards, a Repeat, an End Repeat, and a number parameter block. The number parameter block was labeled with the numbers two through five, with one number on each face of the cube. This number block could be included in a program after a Repeat block to specify how many times a robot should repeat a sequence of actions. Each Forward block would cause the robot to drive forward approximately one half meter. So, for example, one solution to the challenge would be the following program:

Begin -> Repeat (4) -> Forward -> End Repeat -> End

Many other possible solutions exist, and students could add creative elements to their programs. For example, one group included a Sing block to indicate that the robot was eating its cookies after it arrived at the wall. Many groups started this challenge by creating programs that consisted of all the blocks provided to them arranged in a seemingly random order. However, after testing these programs, the groups quickly decided that the Forward block was necessary to drive the robot towards its goal. After one or two additional trials, groups discovered that they didn't have enough Forward blocks to drive the robot all the way from the starting line to the wall. Here, often with prompting in the form of leading questions from a teacher, the groups began to experiment with the Repeat blocks, creating programs that incorporated the Forward blocks inside of a loop.

Later in the unit, we introduced students to the idea of sensors through the *Bird in the cage* activity. Here the cage was a cardboard box, and the bird was a robot with a light sensor attached. The lead teacher told a story of a bird who liked to sing when released from her cage. We used this program to have the robot act out the story.

Begin -> Repeat(forever) -> Wait for Light -> Sing -> End Repeat -> End

The students were curious how the robot was able to sing when it was removed from the cage. The teacher used this curiosity to prompt students to explore the idea of sensors and to develop hypotheses about how the robot is able to sing when it emerges from the box. Finally, the teacher demonstrated how she had created the program that controls the robot:

Teacher:	*Do you recognize all of the blocks?*
Student 1:	*No. The moon one. [Wait for Dark]*
Student 2:	*The sun and moon [Wait for Light]*
Teacher:	*Can one of you read this?*

Student 2: *Wait for light*

Student 1: *It means you're going to wait for the dark to come.*

Teacher: *What are these?*

Students together: *Repeat*

Teacher: *What do they do?*

Student 3: *Start all over again.*

Teacher: *The bird is outside.*

Student 2: *The witch catches the bird*

Student 1: *If we turn this block over we could make him sing when it's dark outside. It might be shy so he sings in the dark.*

Here the child was pointing out that it would be possible to use a *Wait for Dark* block instead of a *Wait for Light* to achieve the opposite effect, a bird that sings only when it is inside the cage.

For the remainder of the curriculum unit, the children worked in pairs to create robotic creatures that ideally incorporated one moving part and one sensor. The children struggled with many aspects of this activity and required substantial help from the teachers. The lead teacher described her interactions with one group:

I worked with M. and N. for the majority of the time. They struggled with understanding that the RCX [LEGO computer] was what would power the robot. M. spent most of the time searching for materials, and I sat with N. trying to understand her plan for their robot (a parrot that snaps it's beak). She had an egg carton, and was decorating it with stickers. When I asked *her where the beak was, she pointed to the egg carton. When I asked her how it would move, she picked up two large pompoms and said that the motors (the pompoms) would make it move. N. had little interest in the RCX and how it worked. When pressed, she said that the buttons and the wires would make the beak move…*

We believe that these problems were due to our curriculum, which did not include enough time for children to explore the RCX brick and understand how a computer (although small) can be connected to a physical construction and how that construction needs to have moving parts that could be controlled by the computer.

As part of the final project presentations we asked students not only to demonstrate their robots, but also to show the class the blocks they used to program it. In many cases, students selected blocks that had little to do with their actual programs. We believe that in the current version of the curriculum, students were not given enough time to experiment with programming on their own, either individually or in small groups. In many cases, the programs were loaded quickly by the teacher in order to finish projects in time for the final presentations. This might explain some of the difficulty students had recreating their programs during the final project presentations.

In other cases, however, students were able to recreate programs more accurately. For example, in this transcription, two girls described their *toucan* robot during the final project presentation:

Teacher: *So what does it do?*

Student 1: *We have to make the program first*

Student 1: *[Starts connecting blocks] Begin. Wait for Dark…*

Teacher: *Can you tell us what we're doing while you're doing it?*

Student 2: *Yes. I think we should use a repeat block*

Student 1: *Great idea*

Teacher: *[Student 2], why don't you show us your robot while [Student 1] builds the program.*

Student 2: *This is our robot… [she starts to demonstrate how it works]*

Student 1: *The program isn't done yet!*

Student 2: *[Looking at the blocks] It's supposed to repeat the shake*

Student 1: *Yes. It's going to repeat the shake over and over again.*

[The program student 1 has created is: Begin -> Wait for Dark -> Beep -> Turn Left -> Repeat -> Shake -> End]

Student 2: *[Runs the program on the robot for the class to see. The actual program waits for dark, turns, beeps, and then shakes once.]*

Student 1: *These are the three blocks we used [She puts these blocks in the middle of the rug: Begin, Wait For Dark, Beep]*

Here it is clear that there is some confusion on the part of the students about the concept of a program being stored on the RCX robot. However, the blocks the students chose to explain their robot are consistent with the program that they had actually loaded on the robot earlier in the day. Moreover, they seemed to understand the notion of repeating an action.

Technology Evolution

The technology used in this project, including both the software and the tangible programming blocks, has evolved substantially as a result of our iterative testing with children.

For our first intervention, we used wooden blocks shaped like interlocking jigsaw puzzles (see Figure 3). These blocks had several drawbacks that became obvious to us when we observed children using them in classrooms. First, the blocks are only big enough to allow space for a computer vision fiducial (the circular black and white symbols) and a small amount of text. This is problematic for children who are still learning to read. As we discovered with our current prototype, children rely heavily on icons as well as text to interpret a particular block's meaning.

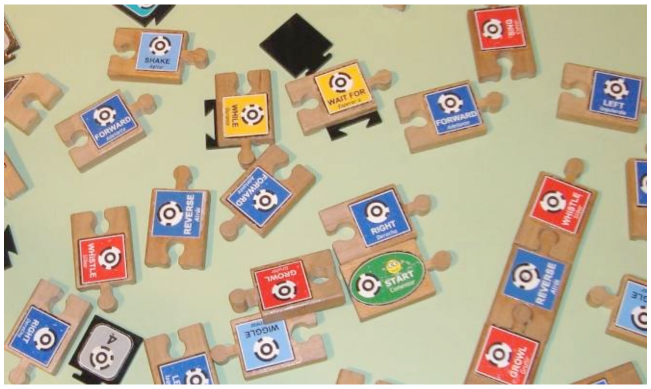

Figure 3. In our first round of evaluation in classrooms we used wooden blocks shaped like jigsaw puzzle pieces.

Second, although children have little difficultly chaining these jigsaw puzzle blocks together to form programs, the blocks tend to fall apart when children try to carry their programs around the classroom. Unfortunately, children need to be able to carry their programs to a specific location in the classroom (the computer with the web camera attached) in order to download their programs to the robot. Finally, these blocks are designed to lie flat on a table surface. As a result, in order for the computer vision system to function properly, we had to suspend the web camera above the table pointing down. This relatively elaborate setup limits the system's portability, increases the teacher's preparation time, and limits the number of stations that can be set up in the classroom for children to use.

To address these problems, we redesigned the blocks using wooden cubes with interlocking pegs and holes instead of jigsaw puzzle tiles. These cubes provide a slightly larger surface area on which to include both text and an icon in addition to the computer vision fiducial (see Figure 1). Furthermore, because the cubes interlock, they are easier for children to carry around the classroom. Finally, because we place a computer vision fiducial on every available face of the cube, we are able to take pictures of programs from the side rather than from above. This allows us to simply place the web camera on a table rather than suspending it above.

As expected, the prototype based on wooden cubes was much easier for students to use. However, for our next round of evaluation, we plan to make several additional improvements, including creating slightly smaller and lighter cubes and exploring better ways to represent the syntax of flow-of-control structures such as loops, parameters, and branches. Children also struggle with the mechanics of downloading their program to a robot. This process involves several discrete steps: 1) turning the robot on; 2) placing the robot in front of an infrared transmitter; 3) placing their program in front of the web camera; 4) pressing the space bar on the computer; and 5) waiting several seconds for their code to download to the robot. From our experience, this process is still too complicated to be practical for young children. It's easy to forget a step, and there are multiple points of possible failure. For example, a child might do everything right but have the robot pointing in the wrong direction. Or, a child might accidentally cover up one of the computer vision fiducials while the computer is taking a picture of the program. In these cases, it is difficult for children to diagnose and fix the problem without the help of an adult. We are investigating a number of potential improvements to this system to simplify the process and reduce the amount of adult assistance necessary.

Conclusion

We face a challenging time in early childhood education. On the one hand, there are stepped up federally-mandated academic demands and a growing concern to respect children's developmental stages. On the other hand, there has not been yet a profound re-thinking of what young children can learn in terms of technology, nor do we have research-based evidence to evaluate children's developmental capabilities with innovative technologies. The emphasis on developmentally appropriate practice and the lack of adequate technologies have severely limited our approach to early childhood education. Our work brings together contributions to the fields of child development, early childhood education, and human-computer interaction. The overarching goal is to provide an empirical foundation for the development of future curriculum and technology for use in early childhood education. The project also proposes to take an unprecedented look at what young children can accomplish with technology when given tools that are truly age-appropriate, and what kind of powerful ideas they are able to learn when programming robots. Furthermore, by conducting the evaluation in the context of a curriculum that introduces increasingly sophisticated concepts of computer programming, our work hopes to build a broad understanding for the scope of young children's abilities to engage in computer programming and robotics activities that can inform educational reform in early childhood.

Acknowledgments

We thank our collaborators at the Tufts University Developmental Technologies Research Group and the Human-Computer Interaction Laboratory including Laura Beals, Kathryn Cantrell, Clement Chau, Jordan Crouser, Rachael Fein, Robert Jacob, Nauman Khan, Emily Lin, Keiko Satoh, and Erin Solovey. Mary Murray from the Massachusetts College of Art and Design provided refreshing insights and designs for our tangible programming language. Finally, we thank the National Science Foundation for support of this research (NSF Grants IIS-0414389 and DRL-0735657). Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Beals, L., & Bers, M. (2006). Robotic technologies: when parents put their learning ahead of their child's. *Journal of Interactive Learning Research*, 17(4), 341-366.

Ben-Ari, M. (1998). Constructivism in computer science education. In *Proceedings of the Technical Symposium on Computer Science Education SIGCSE'98* (pp. 257-261). Atlanta, GA: ACM Press.

Bers, M.U. (2008). *Blocks, robots and computers: learning about technology in early childhood.* New York: Teacher's College Press.

Bers, M.U., Ponte, I., Juelich, K., Viera, A., & Schenker, J. (2002). Teachers as designers: integrating robotics into early childhood education. *Information Technology in Childhood Education*, 123-145.

Bers, M.U., Rogers, C., Beals, L., Portsmore, M., Staszowski, K., Cejka, E., Carberry, A., Gravel, B., Anderson, J., & Barnett, M. (2006). Innovative session: early childhood robotics for learning. In *Proceedings International Conference on Learning Sciences* ICLS'06 (pp. 1036-1042). Bloomington, IN: International Society of the Learning Sciences.

Cejka, E., Rogers, C., & Portsmore, M. (2006). Kindergarten robotics: using robotics to motivate math, science, and engineering literacy in elementary school. *International Journal of Engineering Education*, 22(4), 711-722.

Clements, D. (1999). The future of educational computing research: the case of computer programming. *Information Technology in Childhood Education Annual*, 147-179.

Clements, D., & Sarama, J. (2002). The role of technology in early childhood learning. *Teaching Children Mathematics*, 8(6), 340-343.

Clements, D. & Sarama, J. (2003). Strip mining for gold: research and policy in educational technology: a response to "Fool's Gold. *Association for the Advancement of Computing in Education Journal*, 11(1), 7-69.

Copple, C. & Bredekamp, S. (2009). *Developmentally appropriate practice in early childhood programs: serving children from birth through age 8* (3rd ed.). National Association for the Education of Young Children.

Design-Based Research Collective. (2003). Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher, 32*(1), 5-8.

Gershenfeld, N. (2000). *When things start to think*. New York: Henry Hold and Co.

Haugland, S.W. (1992). The effect of computer software on preschool children's developmental gains. *Journal of Computing in Childhood Education*, 3(1), 15-30.

Horn, M.S., & Jacob, R.J.K. (2007). Designing tangible programming languages for classroom use. In *Proceedings TEI'07 First International Conference on Tangible and Embedded Interaction*. Baton Rouge, LA: ACM Press.

Horn, M. (n.d.). *TopCodes*. Retrieved May 1, 2009, from http://hci.cs.tufts/edu/topcodes

Hourcade, J.P., Bederson, B.B., Druin, A., & Guimbretière, F. (2004). Differences in pointing task performance between preschool children and adults using mice. *ACM Transactions on Computer-Human Interaction*, 11(4), 357-386.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A survey of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37(2), 83-137.

McKeithen, K.B., Reitman, J.S., Rueter, H.H., & Hirtle, S.C. (1981). Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13, 307-325.

McNerney, T. S. (2004). From turtles to tangible programming bricks: explorations in physical language design. *Personal and Ubiquitous Computing*, 8(5), 326-337.

Norman, D. (1986). Cognitive engineering. In D.A. Norman & S.W. Draper (Eds.) *User centered system design: new perspectives on human-computer interaction* (pp. 31-61). Hillsdale, NJ: Lawrence Erlbaum Associates.

Papert, S. (1991). What's the big idea: towards a pedagogy of idea power. *IBM Systems Journal*, 39(3-4), 720-729.

Perlman, R. (1976). Using computer technology to provide a creative learning environment for preschool children. *Logo memo no 24*, Cambridge, MA: MIT Artificial Intelligence Laboratory Publications 260.

Rader, C., Brand, C., &Lewis, C. (1997). Degrees of comprehension: children's understanding of a visual programming environment. In *Proceedings ACM Conference on Human Factors in Computing Systems CHI'99* (pp. 351-358). Atlanta, GA: ACM Press.

Resnick, M. (2003), Playful learning and creative societies. *Education Update*, 8(6), Retrieved May 1, 2009 from http://web.media.mit.edu/~mres/papers/education-update.pdf.

Resnick, M., Martin, F., Berg, R., Borovoy, R., Colella, V., Kramer, K., & Silverman, B. (1998). Digital manipulatives: new toys to think with. In Proceedings ACM *Conference on Human Factors in Computing Systems CHI '98* (pp. 281-287), Los Angeles, CA: ACM Press.

Richardson, K. (1998). *Models of cognitive development*. East Sussex, UK: Psychology Press.

Rogers, C., & Portsmore, M. (2004). Bringing engineering to elementary school. *Journal of STEM Education*, 5(3,4), 17-28.

Smith, A. (2007). Using magnets in physical blocks that behave as programming objects. In *Proceedings First International Conference on Tangible and Embedded Interaction, TEI'07* (pp. 147-150). Baton Rouge, LA: ACM Press.

Suzuki, H., & Kato, H. (1995). Interaction-level support for collaborative learning: algoblock–an open programming language. In *Proceedings Computer Support for Collaborative Learning CSCL'95* (pp. 349-355), Hillsdale, NJ: Lawrence Erlbaum Associates.

Wang, X.C., & Ching, C.C. (2003). Social construction of computer experience in a first-grade classroom: social processes and mediating artifacts. *Early Education and Development*, 14(3), 335-361.

Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *International Journal of the Learning Sciences*, 17(4), 517-550.