Dancing the "Robot Hokey-Pokey": Cognitive Developmental Level as a Predictor of Programming Achievement

A thesis submitted by

Louise P. Flannery

In partial fulfillment of the requirements

for the degree of

Master of Arts in

Child Development

Tufts University

November, 2011

Advisor: Marina Bers

Committee: Marina Bers, Ph.D.; David Henry Feldman, Ph.D.; Bakhtiar Mikhak, Ph.D.

Acknowledgements

I would like to begin by expressing my gratitude to my thesis committee members, Professors Marina Bers, David Henry Feldman, and Bakhtiar Mikhak, for supporting and challenging me over the past two years and throughout the process of putting together this thesis. Their insights and honest critiques have been invaluable. I also want to acknowledge the foundational work without which this thesis would not exist: the dedicated and creative research done through the TangibleK Robotics Project over the past three years by Marina, fellow students Elizabeth Kazakoff, Jordan Crouser, and David Kiger, and my predecessors Michael Horn, and Rachel Fein. Finally, I extend my appreciation to Matt, who continually challenges me to expand my ideas and goals.

The TangibleK Robotics Project was supported by National Science Foundation Grant #DRL-0735657. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

© 2011 Louise Flannery

Table of Contents

List of Tables

List of Figures

Abstract

Distinctive within the extensive ecosystem of children's technologies available today are those for constructing and exploring digital objects – for instance by building robots and programming their behaviors. Such technologies have gained popularity as they can be both entertaining and enriching, especially when designed and used according to cognitive developmental and constructionist learning principles. The TangibleK Robotics Project has conducted three years of developmentally driven research on technology designs and learning expectations for CHERP, a robotics programming tool for kindergarteners. This thesis examines preschoolers' and kindergarteners' problem-solving and reasoning during a programming task as a function of their cognitive developmental level. Results show that while children in late pre-operations engage in meaningful programming explorations, their work differs qualitatively from that of older children transitioning into or already in early concrete operations. The findings inform discussion of developmentally differentiated learning expectations and issues to consider in future technology revisions.

Chapter 1: Introduction

"Technology is anything that wasn't around before you were born," Alan Kay noted in the late 1980's to point out the difference between how adults and youth perceive the same new tools (boyd, 2008). Many young children today interact with computers or "technologies" in myriad forms: electronic learning toys, robotic toys, handheld devices and video games, apps on smartphones and tablets, social networking sites, 3D virtual worlds, online media, and interactive or non-linear computer games (Bergen, 2001; Gutnick, Robb, Takeuchi & Kotler, 2011; NAEYC, 2011; Shuler, 2007). Within the dense ecosystem of technologies children can access, products that encourage children to construct and explore their own digital content, similarly to how they build and experiment with physical materials, stand out to parents and educators as both engaging and enriching (Ito, 2009; Shuler, 2007). As computer technologies become increasingly inexpensive and ubiquitous, young children are more likely to use them, whether in formal educational settings or simply because they are available at home. It is crucial that a developmental perspective inform parents, educators, and technology designers in their work so that educationally and developmentally productive genres and uses of technology are promoted across home, school, and industry contexts as well as throughout childhood.

The purpose of this thesis is to systematically incorporate a cognitive developmental perspective into research at the intersection of early childhood learning and new technologies. Developmentally-based understandings of young children's cognition are crucial to effectively shape learning expectations, curricula, and the design of appropriate learning materials. The analysis presented here makes use of cognitive developmental and other data based on a programming activity completed by each of 36 children as part of the TangibleK Robotics Project. A new procedure was constructed for assessing three Piaget-based sub-stages of cognitive development based on the programming activities. Findings from this analysis can inform further work on the role of cognitive development and technology use, the adoption of differentiated learning expectations and curricula for programming (of robots) throughout the early childhood years, and the design of new technologies for this complex and engaging activity.

The novel technological tools seen today are many and diverse, but the debates they inspire reflect continuities in the social and cultural context over the past several decades. Each new form of media since the radio broadcast has raised strong concerns by some as well as fervent optimism by others over its impact on children’s cognitive, social, physical, and moral development (Ito, 2009; Wartella & Jennings, 2000). Product developers, educators, parents, and policy-makers justifiably continue to debate whether and how new technologies can meaningfully impact learning and development as well as provide entertainment, just as they did two and three decades ago when children’s software was just entering the market (Ito, 2009). At the core of these discussions is the worry that new and imperfect technologies inherently determine how children will use them, a perspective that ignores the fact that societies, individuals, and technologies mutually shape the roles novel technologies end up filling (Ito, 2009; NAEYC, 2011). Despite decades of theory and research on effective uses of computers for learning (e.g. Papert, 1993), too often, adults design and use new technologies in limited ways compared to their potential uses for supporting learning (Bergen, 2001). With the wealth of technology at their fingertips, youth are often today’s leaders in using computers to their full advantage during childhood (New Media Consortium, 2005).

It is not enough, however, to trust that children will construct digital objects and knowledge on their own (Jenkins, Purushotma, Weigel, Clinton & Robison, 2009). This is where education plays a key role. In fact, some of the most pertinent skills for later success that children can get from their education may be supported by the use of technology and have more to do with adaptability, creativity, and self-driven learning than any particular domain of factual knowledge (Resnick, 2007). Technologies which encourage and enable children to design and create complex and dynamic objects both on- and off-screen fulfill a need in today's evolving educational and work requirements and fill a niche in entertainment as well.

Unfortunately, television dominates the hours most children spend engaged with media technologies (Gutnick et al., 2011). As children get older and more technologies are available to them, however (Gutnick et al., 2011), children's relationship to technology and media increasingly draws on the active, creative, and personal (Resnick, 2006) connotations of an artist's 'media' rather than the passive connotations of television and video 'media.' Youth today not only view but also generate, share, and remix digital content, explore interactive simulations, and participate in rich virtual worlds (Ito, 2009; Shuler, 2007). Children have always explored and created with the materials around them through activities like drawing, dramatization, and making models. The kinds of materials available and their affordances, though, have dramatically changed with the introduction and evolution of computers.

Advancements in both computer technology and in human-computer interactions (Horn, Solovey & Jacob, 2008) are beginning to provide technological toys and learning tools that engage children as early as preschool with the genre of creative and cognitively engaging technological activities that older children currently have available. Technologies for creating models, dynamic simulations, games, and interactive art by programming them translate traditional forms of hands-on exploration into the realm of modern materials (Bers, 2008; Resnick et al., 2009). The iterative process of imagining, programming, exploring, sharing, and reflecting on digital creations allows children to assert ownership over the pervasive digital components of their world (Resnick, 2006).

Programming tools for creating interactive art, games, and robots exist for children, typically over the age of 7, although some work has been done with even younger children (Bers, 2010). This work tends to be grounded in an iterative design methodology including substantial laboratory and classroom experience with the relevant age groups (Barab & Squire, 2004; Brown, 1992), but developmental theory stands to play a more significant role in informing this work. The more human-computer interaction research merges with developmentally appropriate practice and is informed by child development research, the more new technologies and effective curricula can lower the barrier for children's rich exploration of the digital world in ways that respect children's unique developmental characteristics and at the same time foster positive learning and personal outcomes (Cooper, 2005).

Discovering what young children are capable of learning and doing with new programming technologies is the work of the TangibleK Robotics Project, carried out by the DevTech Research Group at Tufts University's Eliot-Pearson Department of Child Development (Bers, 2010). The work presented in this thesis examines cognitive developmental differences in programming seen during one-on-one work with 36 preschool and kindergarten children in a laboratory-based TangibleK study. The study was designed to capture an in-depth picture of each child's thought processes as they reasoned through a given programming challenge. Wide variation existed in children's programming outcomes and was unaccounted for by measures assessed during the study. This analysis develops and applies a metric of revised Piagetian stages of cognitive development as well as a more detailed measure of programming achievement. The framework for assessing development is presented along with examination of how children's cognitive approaches to programming varied qualitatively by level of cognitive development. Correlations between developmental level and programming approaches and achievement are explored. Other possible cognitive, demographic, parental, and experiential factors outcomes are examined for comparison. Case studies detail patterns and findings. Results of the analysis are intended to inform future endeavors to design programming tools and curricula which purposefully and effectively scaffold children's learning of programming and robotics in cognitive developmentally appropriate ways.

Chapter 2: The TangibleK Robotics Project

The DevTech Research Group, which has carried out the TangibleK Robotics Project, explores the intersection of applied developmental theory, learning, and new computer technologies (Bers, 2010). Its work addresses the fields of technology and engineering, which have a warranted yet underrepresented place in K-12 education. The TangibleK Robotics Project in particular introduces kindergarteners to powerful yet age-appropriate technological tools for building and programming robots (Bers, 2010). Through these materials, young children explore basic concepts of computer science and computational thinking. They also gain access to a modern expressive media, which, much like pens or watercolor paints, can be used for a wide range of personally and academically meaningful endeavors.

More specifically, the TangibleK Robotics Project 1) explores what and how kindergarteners can learn about programming and robotics through building robotic vehicles and composing behaviors for them, 2) iteratively tests and refines a curriculum to introduce the core concepts of programming and robotics, and 3) examines how design features of the programming and robotics materials best support learning of these domains (Bers, 2010). The project builds on prior research showing that children can meaningfully explore technological domains given materials and pedagogies designed especially for their developmental characteristics (Bers, 2010).

Overview of the Technologies

To understand the research and findings of the TangibleK Robotics Project, it is useful to keep in mind the technological materials the study participants worked with. In each of the Project’s studies, children constructed a robotic vehicle with pre-selected parts from the LEGO® Mindstorms® robotics construction kit: the RCX™ brick (which contains the ‘brain’ of the robot and has attachment points for other parts), motors, sensors, wires, and wheels (Figure D1). Traditional LEGO® bricks were available for building non-robotic parts of the vehicle, adding sturdiness, and personalizing the robot. (An example of a completed vehicle can also be seen in Figure D1. See Appendix A for a more detailed description of how each robotic part works.)

A defining feature of robots is that they can be given behaviors to carry out automatically. Programming, or computer programming, is the selection and sequencing of instructions that a computer will carry out. Programming ‘languages,’ ‘interfaces,’ ‘tools,’ and ‘environments’ all describe overlapping aspects of the hardware and software used to program. In this paper, every effort is made to use ‘language’ to refer to the instruction set, ‘interface’ to the means by which a person constructs a program, and ‘tools’ and ‘environments’ to the total package of programming hardware and software.

To give the RCX™ robots behaviors, children used CHERP (the Creative Hybrid Environment for Robotics Programming). CHERP was developed in a joint effort between the Tufts University Computer Science and Child Development Departments (DevTech Research Group, 2010). It is designed for programming vehicle-like robots and expands earlier work on developmentally appropriate programming tools for children by combining graphical (on-screen) and tangible (physical, off-screen) interfaces into a hybrid interface (Bers & Horn; 2010). With CHERP, a program is made simply by connecting wooden blocks labeled with icons and text (Figure D2), or by clicking and snapping together the corresponding on-screen blocks (Figure D3) (Horn, Crouser & Bers, 2011).

When the child clicks one of the on-screen ‘upload’ buttons, CHERP translates the physical or graphical program into robot-interpretable code and communicates it to the robot via an infrared transmitter (Figures D3 and D4). A distinguishing feature of CHERP is that it creates an on-screen version of each uploaded tangible program (Horn et al., 2011). The child can then edit either the tangible or graphical program since both interfaces provide the same functions. This hybrid interface allows children to fluidly choose whichever programming interface is best suited to their skills, knowledge, interest, and current goal. In turn, this flexibility may improve children’s learning and enjoyment of programming (Horn et al., 2011).

CHERP’s high-level instruction set maps directly to actions by a robotic vehicle as a whole, rather than actions of an individual part, as is the case with many other programming languages. A high-level language reduces the significant cognitive burden inherent in more complex programming languages of decomposing and mapping multiple levels of representation of the goal (Repenning, Webb & Ioannidou, 2010). This simplification makes CHERP more appropriate for young children. CHERP’s language includes three different types of instructions, which are introduced lesson by lesson in the TangibleK curriculum: actions, control flow structures, and parameters. (CHERP also has ‘Begin’ and ‘End’ instructions which demarcate the intended program for image-processing.) Action instructions for movements, sounds, and lights correspond directly to a single behavior by the whole robot (Figure D5). Control flow structures are meta-instructions, which specify how or when to carry out actions, for instance, by looping a series of actions. Parameters provide information about how the robot should carry out the control flow instruction, for instance, how many times to loop. Each CHERP programming instruction is represented with both an icon and a word, to support understanding by early readers. Categories of actions are distinctly colored to further facilitate differentiation of the instructions; for instance, sounds are orange, lighting actions are green, and the ‘Begin’ and ‘End’ blocks follow a stop-light color-scheme analogy.

CHERP also has embedded supports for learning its syntax (Bers, 2010). The physical form of the blocks prevents many kinds of syntax errors, like attaching parameters to an action instead of a control flow structure. The graphical blocks behave similarly; they only snap together if the construction is syntactically logical. Furthermore, before CHERP uploads a program to a robot, it automatically detects any lingering syntax errors. In such a case, as in omitting the ‘End’ block, the software displays a concise icon-and-text message that supports children as they learn how CHERP works as a language and as software.

CHERP’s set of programming instructions, their representations, and the hybrid interface for manipulating them are cognitively and physically accessible to kindergarteners. Young children can playfully problem-solve and bring imaginative creations to life with CHERP and a robotics construction kit, all the while learning powerful ideas from technology-based domains generally – though unnecessarily – reserved for older children or even adults (Bers, 2008; Bers & Horn, 2010).

Overview of the TangibleK One-on-One Study

From January to May, 2010, the author worked individually with 36 pre-school and kindergarten children, teaching them how to program with CHERP and documenting the process of their learning and their use of the technologies. The goal of this lab-based study within the

TangibleK Robotics Project was to provide a highly detailed account of kindergartners' learning processes, an area that had been difficult to document in classroom settings earlier in the TangibleK Robotics Project. Children received instruction and completed three challenges with the CHERP programming tool and LEGO®'s RCX™ robotics systems. Substantial amounts of data were collected on children's thought processes and levels of understanding and accomplishment, as well as background information and experience.

Each child participated in four study sessions. During a small-group introductory session, children completed baseline assessments on key physical and cognitive abilities such as fine-motor skill, eye-hand-coordination, sequencing, making correspondences, and segmenting tasks into core components. The group of around three children was introduced to CHERP and a pre-built RCX™ robot, taught how these technologies work, and given time to program the robot as they wished. Each child then participated in three individual sessions on later dates. Each session included a review of familiar material, introduction of new concepts, building a robotic vehicle, completion of a programming challenge based on the new concepts, and reflection by children on core elements of robotics and programming. The first of the programming challenges was to program the robot to dance the last verse of the “Hokey-Pokey” (“You put your whole self in…”), an activity which used only action instructions, other than the requisite ‘Begin’ and ‘End’ instructions (Figure D5). The “Hokey-Pokey” activity is the subject of analysis in this thesis.

The second programming activity was to making the robot drive along an L-shaped ‘road’ using actions and a looping instruction. The third challenge was to use actions, a decision-making instruction, and a touch sensor to program the robot to drive along different routes depending on the state of the sensor. Post-intervention assessments were administered to measure the impact of the three programming and robotics activities on the physical and cognitive skills tested during the introductory session.

The TangibleK Robotics Project draws on decades of work on children's development and learning and the design and application of computational learning tools. The next two sections overview the theoretical and research background first on children's programming and construction-based learning environments and then the cognitive development of young children.

Chapter 3: Children's Computer Programming

Today, many creative and exploratory activities that first took form in the physical world have migrated into digital territory. The flexibility and power of computers to do many kinds of tasks quickly and automatically extend what people are capable of producing and exploring. Even children now widely achieve sophisticated levels of composition, expression, and communication with computers by creating two- and three-dimensional graphics and animations, making and playing with interactive art and games, integrating images, text, audio, and video to express information and ideas, and constructing programmable robotic objects. Programming environments are unique among tools for construction or creation in that children must specify a series of instructions to accomplish a given outcome. In this process, children can discover and apply a wide range of powerful cognitive strategies that may benefit them across domains. Programming also poses particular challenges to novices, including children, although many languages have been design specifically to alleviate these barriers.

Theory and Research on the Benefits of Programming for Children

Technologies for children's programming have existed since the 1970s, when Logo became widely available along with the first personal computers (Logo Foundation, 2000b), and have gained popularity as a form of media production as computers have become more commonplace and user-friendly. Throughout the past four decades, numerous theories and studies have supported the potential benefits of children's learning to program. These benefits depend heavily on the pedagogical frameworks and learning contexts within which they are used.

Skills for a creative 21st century society.

The introduction of the printing press increased the average person's access both to texts and to cheap publishing. This consequently shifted the predominant modes of documenting and communicating ideas and information from memory and speech to reading and writing. Similarly, the computer has transformed prevalent modes of gathering and presenting information from text-based to multi-modal and multi-media platforms by providing flexible tools for representing and disseminating ideas and information (New Media Consortium, 2005). Today, society relies heavily on an evolving set of technological tools, which has made adaptability and problem-solving at least as important for success in the work-force as particular domains of knowledge (Resnick, 2007). In fact, Laura Richardson describes 46 such "SuperPowers for the 21st century" which covering many forms of creative and proactive learning, many of which masquerade as play with rich materials such as computer-based media (SuperPowers of Play, 2011).

Many people have called for K-12 education to incorporate the diverse technologically-based modes of expression and exploration that computers have enabled, by broadening definitions of literacy and media education (e.g. Buckingham, 2007; New Media Consortium, 2005; Peppler & Kafai, 2007). Several frameworks advocate for calling the abilities to learn and use a range of technologies for expression 'literacies' in themselves – media literacies or technological literacies (International Technology Education Association (ITEA), 2007; Jenkins et al., 2009). These frameworks shed light on how children can use a changing palette of technologies to gather, critically examine, compose, and express ideas and content. In this context, programming graphics or robots can be seen not only as a mode of expression but also as a means of understanding and developing confidence in using technological media fluently. Developmentally appropriate tools for building and programming robots offer ways to actively engage with and understand the digital components of today's world (Bers, 2008). They provide a context for child-driven, expressive, generative, and exploratory experiences that increase technological fluency (Bers, 2008; Resnick, 2006).

While some frameworks for technological literacies focus on procedural or ‘how-to’ knowledge of using computers for word processing, internet research, etc. (e.g. Massachusetts Department of Education (MA DOE), 2006, 2008), many emphasize the importance of children’s taking an active role in generating, sharing, remixing, and responding to content (Jenkins et al., 2009). Procedural literacies are certainly relevant, but stopping there would dismiss the rich, varied, and highly user-driven modes of computer use available. The crucial skill for children to develop is technological fluency, or expressivity with a variety of computational tools (Papert, 1993) – just as the goal of traditional literacy education aims beyond decoding letters, words, and sentences toward oral and written fluency.

Whether connected to or separate from an academic context, computational tools for construction can provide a motivating context in which to learn the iterative creative thinking process (akin to the design process) and to “come up with creative solutions to solve unexpected problems,” (Resnick, 2007, p. 18) a fundamental skills for today’s dynamic world. The word *creative* in this line of thought is overloaded with three distinct connotations: the use of *artistic* media for expression, communication, and interaction; the development of a *divergent thinking* style characterized by the generation of multiple solutions or novel ideas; and the *construction* or creation of objects that exist outside the mind in three-dimensional or on-screen form.

Sophisticated yet accessible new technologies for programming animations and robots can support all three types of creativity given well-designed technologies and effective goals for their use (Resnick, 2007). The creative thinking process, applied to technological tools for construction, empowers children to have ownership over complex and fascinating aspects of their world – computers, in their many forms.

More detailed frameworks for technology-based literacies also focus on children's proactive and creative consumption, construction, and sharing of ideas. Among eleven media literacies Jenkins et al. (2009) see as key in today's increasingly participatory media culture are three which outline a hands-on approach to technology: play (experimentation with one's surroundings), simulation (creation and/or use of dynamic representations of real systems), and distributed cognition – the use of tools that divide the cognitive load of a task among people and/or computers. (Other skills in Jenkins' framework of media literacies address the social aspects of using participatory and interconnected media as well as how youth interact with large and varied online resources.)

Another framework, "Standards for Technological Literacy: Content for the Study of Technology," proposes foundational knowledge that students should have about the technological world they engage with so much (International Technology Education Association (ITEA), 2007). ITEA proposes a definition of technological literacy that addresses the history and nature of many genres of technology to better inform our widespread use of them (ITEA, 2007). The group's frameworks, designed to support the integration of technology as a core content domain in K-12 education, define technological literacy as "the ability to use, manage, assess, and understand technology" (p. 7) as well as the ability to understand the role of technology in society. Among categories of standards regarding the nature of technology, how technologies and societies impact each other, and a wide range of technological fields and human-designed aspects of the world, three standards address general abilities that all people should have regarding the use of technology: application of the design process, comfort in learning how to use and maintain technological systems effectively and safely, and assessment of the impacts of a technology. These skills reflect the active role children – and adults – should have with regards to technology and the reflective mindset necessary for taking full advantage of what new technologies have to offer.

At their core, new literacies for the creative, digital, and web-connected world all promote a common vision: the proactive, reflective, and flexible learner who can draw from and contribute to diverse perspectives and modes of thinking. Programming tools are powerful within the ecology of digital media platforms in that they can be used for content production and expression and as a context for understanding something about how our ubiquitous computer technologies work.

Constructionist learning.

Digital literacies and technological knowledge and fluency can also transform learning and education. By creatively designing and personalizing computational objects and their behaviors, children work within a multi-disciplinary context for content-domain learning that has the potential to help children see learning and academic knowledge as personally meaningful (Martin, Mikhak, Resnick, Silverman & Berg, 2000; Papert, 1993). The *constructionist* perspective – learning through making and reflecting on the process and its results – aims to provide a model for learning that mirrors the active yet innate construction of knowledge during development (Papert, 1999; Papert & Harel, 1991). In the process of programmatically building and experimenting with digital objects, children may naturally and simultaneously need to learn *powerful ideas* from various domains, such as mathematics or science (Martin et al., 2000; Papert, 1993). The term ‘powerful ideas’ has been defined multiple ways, generally encompassing core concepts and skills of particular domains which are embedded in a personally meaningful context or problem, and which can be applied to solve or understand a wide range of authentic problems (Bers, 2008; Papert, 2000). Ideas, from the concept of proportion to the application of computers to solve problems, are powerful in their “contribution [. . .] to the growth of knowledge” (Papert, 2000, p. 725).

In Papert’s vision, school learning should align with the constructivist development model of knowledge structure formation and base learning on contexts and entry points that each child finds especially engaging. This could happen through interdisciplinary robotics and programming curricula and by structuring pedagogies around careful and curious reflection on the powerful content-domain ideas that are illustrated through programming. Such personally and academically relevant learning could, theoretically reverse societal trends of anxiety towards, for instance, math, and instead effectively promote a love of ideas and learning (Papert, 1993). At the very least, the constructionist framework for learning has provided a model of education that supports deep and motivating learning when properly implemented.

Computational thinking.

There are many ways of creating and exploring rich and engaging media with a computer, but programming tools specifically have the potential to engage users in *computational thinking* (CT). CT encompasses a broad and somewhat debated range of analytic and problem-solving skills, dispositions, habits, and approaches used in computer science (International Society for Technology Education & the Computer Science Teachers Association, 2011; Lee et al., 2011; Barr & Stephenson, 2011) to generate novel solutions to problems as algorithms for computers to carry out automatically (Papert, 1993; Wing, 2008). An “exploration of process” in general terms (Guzdial, 2008, p. 25), CT also shares characteristics with several kinds of analytical thinking – logico-mathematical reasoning, the engineering design process, and the scientific method (Lee, et al., 2011; Wing, 2008). However, CT uniquely focuses on problem representations which can be solved by information-processing agents (Cuny, Snyder, Wing, as cited in Center for Computational Thinking: Carnegie Mellon, 2011), whether human or machine (Wing, 2008).

Children’s programming of animations, graphical models, games, and robots with age-appropriate materials engages them with core elements of CT such as abstraction, automation, analysis, decomposition, modularization, and iterative design (e.g. Lee, et al., 2011; Mioduser, Levy & Talis, 2009; Mioduser & Levy 2010; Resnick, 2006; Resnick et al., 2009). Studies have shown that, with explicit instruction, programming can be a rich environment for acquiring transferable skills in problem representation and trouble-shooting (Klahr & Carver, 1988; Salomon & Perkins).

Programming tools let children imagine and build up complex actions from simpler units and grapple with sophisticated ideas. Given the rich variety of real-world problems solved through computer algorithms and computational models (Guzdial, 2008; Wing, 2008) and the generalizable nature of many core CT concepts, children may benefit from having computational thinking added to their repertoire of analytic perspectives. This is accomplished by providing age-appropriate computers and information-processing models as tools for children to creatively solve problems and accomplish goals through activities such as programming.

High-level cognition.

Programming can also engage children in high-level cognitive processes that do not fall directly under the category of computational thinking. Papert’s belief that programming could drastically change learning (Papert, 1993; 1999) rests on the richness of learning through the construction of real-world objects and the nature of programming to foster problem-solving and meta-cognition (Liao & Bright, 1991; Papert & Harel, 1991). Because creating instructions for computers requires much more exact and sequential steps than instructing a person, programming invites reflection on one's own thought processes as well as precise decomposition of complex processes (Papert, 1993). The learning context is vital to children's consistent use of particular cognitive strategies – projects must be complex enough to afford deep exploration, and the learning community must be open to acknowledging and analyzing instances in which something works in an unexpected way. For this reason, Papert simultaneously promoted Logo, a powerful tool for programming and learning, as well as the "Logo spirit," (Papert, 1999, p. vi), the kind of classroom values which support the rich thinking and learning he envisioned.

Papert himself took a holistic approach to analyzing and synthesizing the outcomes of classroom experiences with Logo to support and refine his ideas (Papert, 1987). His work inspired dozens of studies looking for data-based evidence of Logo programming's impacts on cognition as well as many theoretical arguments for and against the basic tenets of his work. The wave of research on Logo from the mid 1980s to early 1990s showed mixed results (Clements & Meredith, 1992; Liao & Bright, 1991). One highly cited study found positive outcomes in reflectivity, divergent thinking, meta-cognition, and direction giving for Logo groups compared to control groups (Clements & Gullo, 1984). A meta-analysis of research on Logo and cognition showed that 89% of such studies had found positive correlations between Logo experience and cognitive outcomes in comparison to a control group (Liao & Bright, 1991). This pattern was upheld in the current TangibleK work, which showed that, on average, children's ability to sequence picture stories is higher after even a brief but focused exposure to programming robots than beforehand (Kazakoff & Bers, 2011).

Another comprehensive meta-analysis which covered domains from basic math to social cognition and language showed that Logo had positive impacts on children’s acquisition of certain concepts in particular contexts. Concepts that Logo addressed most directly – like distance units or angles – were more likely to be impacted than concepts that it indirectly addressed – such as variables, which have different connotations in algebra and programming (Clements & Meredith, 1992). This finding makes sense in light of the well-established difficulty of transferring knowledge or skills across domains without explicit instruction. Overall, though, the research has shown that factors influencing students’ programming outcomes are complexly linked and difficult to fully model, but that with intentional and structured use, Logo can be used as a context for exploring and teaching a wide range of concepts.

Since the research of the 1980s on cognitive outcomes of Logo programming, huge advances in computer power and human-computer interfaces have allowed the development of programming and robotics tools that are much more tailored to children's unique abilities at different stages of development than the tools available during the 1980's and 1990's were. These advances may allow research to return to the issue of developing a comprehensive theoretical model of what benefits can come of programming throughout childhood and how such benefits are attained.

The importance of context.

Learning goals, activity structure, and the design of the technology all contribute to the impact programming or other constructionist activities have on cognition and learning. By itself, the availability of certain programming instructions or possibility of using particular cognitive strategies with a given programming tools does not lead to all users independently discovering, fully exploring, and appropriating them without specific guidance to do so, as Pea and colleagues argued in their controversial warnings against Logo-as-educational-cure-all attitudes (Pea & Kurland, 1984; Pea, Kurland, & Hawkins, 1985). While some criticized the research as having taken too wide a definition of ‘programming’ and too narrow a definition of ‘cognitive impacts’ (Papert, 1987), the point raised is an important one. The debate over guaranteed impacts has always figured substantially in discussion of children’s technologies. Computer technologies are just tools, no more and no less; ideas about how they can be used and the manner in which people learn to use them determine whether and how they will prove beneficial in particular ways.

Explicit instruction is a crucial factor in student’s acquisition not only of Logo knowledge but also of meta-cognitive, self-monitoring, and debugging or trouble-shooting skills (Clements & Meredith, 1992; Lee & Thompson, 1997). In fact, explicit and ongoing teacher mediation during Logo activities seems to be necessary for acquisition of concepts from the details of geometry to the steps of successful problem-solving (Clements & Meredith, 1992). This has also been found to apply to computer activities more generally (Nir-Gal & Klein, 2004).

The design of powerful, engaging, and user-friendly technologies is the start to seeing diverse educational benefits from programming. However, the keys to successfully and meaningfully incorporating programming in childhood contexts are not inherent to technology but are instead socially constructed. These include: instruction and activities that introduce tools and concepts in an explicit and structured way, pedagogies that focus on project-based learning and that foster reflection, iterative design, and problem-solving, the attitude that success comes after many ‘failed’ and revised attempts, and that unexpected results and the intermediate steps of an iterative problem-solving process are not cause for judgment on the learner but integral and informative parts of the learning process.

Positive Technological Development (PTD) theory highlights the importance of the personal, social, and cultural context of learning with technology (Bers, 2010). The PTD model provides a framework for intentionally structuring technology-focused programs and curricula so as to promote advantageous cognitive, social, and moral developmental outcomes in addition to content-domain learning and technological literacy. Educational experiences can be structured to encourage content generation, creative design and problem-solving, collaboration, communication, choices of conduct, and community-building in ways that may in turn foster the development of beneficial core cognitive and social traits: a sense of competence and confidence, the ability to connect with and care about others, contribution to entities outside the self, and moral character (Bers, 2010). While many situations and learning tools may make these gains possible, it is the responsibility of educators to carefully structure learning programs to purposely and systematically foster positive outcomes.

Computers and devices with embedded computational power are everywhere, and learning to program them has many benefits in addition to being highly relevant in today's world. Playing with a *re*-programmable rather than *pre*-programmed or minimally interactive toy (Bergen, 2001) engages a child in imagining, creating, and playfully exploring the outcomes of his or her efforts (Resnick, 2007). The computational thinking, observation, analysis, refinement, and iteration involved in creating a successful program can foster skills and attitudes in problem-solving and persistence that are helpful in all domains. Continued exploration into child-appropriate programming tools and curricula can support the successful use of these materials in classrooms and give more children developmentally appropriate experiences in creating and learning through computational materials.

Novice Programming

Programming can have many benefits, but it is also a complex activity which can pose challenges for novices, and it is important to understand these challenges to design better programming environments and curricula. Programming can be broken down into different sets of conceptually distinct components, strategies, and skills to understand how different people attain different levels of programming achievement. Bishop-Clark (1995) decomposes programming into steps resembling the engineering design process, each of which requires distinct skill sets: representation of the program, design of the solution, coding, and debugging.

Other frameworks examine programming through genres of knowledge and skills that cut across the steps of designing and revising a program. The specific categories vary by author, but most draw from: declarative knowledge (the instructions and syntax used to write programming code), procedural knowledge (how each instruction works), conditional knowledge (how instructions interact in an algorithm and knowledge of general solution patterns or outlines), and strategic knowledge (decomposition of the goal and construction and revision of a program that addresses it) (Lau & Yuen, 2009; McGill & Volet, 1997; Robins et al., 2003). Mental models or analogies also play a role with regards to understanding the meaning of programming instructions and what the computer does to carry them out (Robins et al., 2003).

There is a steep learning curve in mapping high-level goals to successively lower-level abstractions and finally to specific instructions to create a solution (Repenning et al., 2010). Novice programmers can have trouble with any of the above components and skills of programming compared to more experienced or expert programmers (Robins et al., 2003). Many novice-expert differences in programming parallel those seen in many other fields (Winslow, 1996) and are seen in programmers of all ages (Pea, 1986). The literature on designing and assessing programming languages for novices notes the complexity and rigidity of many languages' instruction sets and syntaxes as major sources of frustration and difficulty in the learning process (Horn et al., 2011; Kelleher & Pausch, 2005). On the other hand, literature on types of novice difficulties with programming finds that challenges with learning languages' basic components are overshadowed by higher-level skills, like planning a solution structure; this body of work suggests that it is the application of strategy and problem-solving skills to specific programming languages and goals that pose the most substantial barriers to programming success (Robins et al., 2003; Winslow, 1996).

There is consensus that strategically combining and structuring programming statements to solve a particular problem are skills that novices but not experts find difficult (Robins et al., 2003; Winslow, 1996). Although novices can usually solve a given problem with familiar materials or in their own words, they have difficulty translating the solution into code (Winslow, 1996). Novices are also less likely to strategically plan out a program, to draw on diverse problem models to do so, to test their work in progress, to trace through their code successfully and find errors, and to apply all the relevant knowledge they have towards creating a programmatic solution (Robins, et al., 2003). They have trouble with the fact that computers carry out programs entirely sequentially (Pea, 1986), which is a very different structure from the parallel nature of human thinking and many other complex systems. One major shortcoming of programming education may lie in the tendency for courses to focus on language syntax rather than on specific strategies for analyzing problems' structures or creating advantageous program plans with both general and specific problem solving strategies (Robins et al., 2003; Winslow, 1996).

In terms of specific programming structures, novices find it particularly challenging to trace through the logic of control flow structures like loops and conditional statements and keep track of how variable values change accordingly (Robins et al., 2003). Examination of non-programmers' natural patterns of describing rules and their success in applying different conditional statement forms has suggested that novices would have fewer control flow difficulties if the programming instructions and syntax more closely reflected natural human language and ways of describing conditional rules (Guzdial, 2008). To make sense of programming instructions and the computer's unseen actions, novices often do rely on natural language connections and anthropomorphism, but many such analogies are incorrect or misleading (Pea, 1986; Robins et al., 2003). Natural language and technical programming terms often have related but distinct meanings, which must be distinguished for the programming context. Thinking of the computer with the analogy of a sentient being also leads to erroneous assumptions that the computer will interpret code the way the programmer intends rather than by the strict syntactical rules of the language (Pea, 1986).

Surprising individual variation in adults' abilities to learn programming has been consistently observed (Mancy & Reid, 2004; Robins et al., 2003). Despite many studies examining differences between genders and among learning styles, cognitive styles, general aptitudes, and conceptual frameworks underlying specific languages, no overarching theory of novice achievement in programming has emerged (Lau & Yuen, 2009; Mancy & Reid, 2004). Nonetheless, research has provided insights into the impact of a number of factors.

Perkins, Hancock, Hobb, Martin & Simmons (as cited in Robins et al., 2003) found that novice programmers tend to have one of three patterns of behavior when uncertain as to the cause of an unexpectedly functioning program. 'Stoppers' simply gave up; 'tinkerers' modified their code unsystematically and were unlikely to reach a solution; 'movers' looked for sources of error systematically, reflected on the expected results of their actions, and were able to resolve impasses. This pattern illustrates the importance for computer science education to take into account the attitudes and emotions that accompany programming as well as the content of the domain.

Several studies (e.g. Engle, Tuholski, Laughlin & Conway, 1999; Heinz-Martin, Oberauer, Wittman, Wilhelm & Schulze, 2002) have shown that working memory capacity predicts intelligence, particularly reasoning. This finding may have implications for understanding differences among novices, particularly between younger and older children, as well as different levels of success according to the cognitive demands of a language. Shute (1991) found that working memory, problem identification, sequencing of elements, asking for hints from a knowledgeable source, and testing programs regularly by running them all contribute to success in learning to program. The field dependent/independent cognitive style, the ability to recognize an object or idea distinctly from its perceptual or superficial context, and spatial ability, which is linked, may also play significant roles in learning to program (Jones & Burnett, 2007; Mancy & Reid, 2004).

A number of programming languages have been created to address the difficulties experienced by adult novice programmers as well as to foster the recreational interest many people have in learning programming. Such languages include Alice (Carnegie Mellon University, 2011), ROBOLab (Rogers & Portsmore, 2004), Arduino (Arduino, 2011), and Processing (Fry & Reas, n.d.). Languages such as these reduce declarative and procedural knowledge requirements since they have smaller instruction vocabularies and less syntactically complicated coding. Some also use graphical interfaces for choosing and sequencing instructions, the advantages of which are discussed in the following section. These languages also facilitate learning to create and debug algorithms because the output is directly observable: graphics, animations, or robotics actions.

Other languages address many known challenges that young novice programmers in particular encounter: reading and typing, fine motor and eye-hand control for using a mouse, large volumes of programming instructions, and understanding and visualizing the effects of control flow structures (Bers, 2010; Horn et al., 2011). The language used in the TangibleK studies, CHERP, was designed to meet the cognitive and physical developmental needs of kindergarteners. Details can be found in the Chapter 2 overview of technologies used in this study. The next section provides other examples of programming languages for children.

Examples of Children's Programming Technologies

Programming environments for children have evolved along with advances in computer technology. Early programming languages for children, like Logo, were text-based, largely because of the state of computer technology at the time. Children used Logo’s simplified vocabulary and syntax to control the motion of either a robotic or on-screen ‘turtle’ and to explore mathematical relationships in the process (Logo Foundation, 2000b). Due to the bold claims by Seymour Papert regarding programming’s potential to revolutionize learning and schools, Logo became the subject of two decades of extensive research on the cognitive and social impacts of programming throughout childhood, discussed earlier. Since then descendants of the original Logo have incorporated rich new instructions for parallelism, a hybrid text/graphical interface for usability, and connections to LEGO® robotics parts to bring programs off the screen. These changes have resulted in tools with which children can construct diverse multimedia objects, simulations, and robots (Ito, 2009; Logo Foundation, 2000; Martin et al., 2000).

A growing variety of other programming environments for children is currently available freely online or commercially and can be used by children of varying ages to create animations and control robots. These languages have moved away from a text-based format, instead favoring graphical or physical blocks labeled with words and/or icons to represent instructions. Graphical and tangible interfaces address physical and cognitive challenges that text-based programming languages pose for novices, particularly young children: the need to type, the need to map actions to words rather than a more concrete representation, and the need to remember a complex grammar (Bers, 2010; Perlman, 1976; Repenning et al., 2010).

Graphical programs are constructed on-screen with a mouse or touch-screen by dragging and dropping instructions into a series; instructions ‘snap together’ to form the program. Such languages reduce or eliminate the need to learn the programming language’s grammar, which is inherently suggested or maintained through the shapes, sizes, colors, and behaviors of different types of on-screen instruction blocks. For example, Scratch, a piece of software for programming graphics and animations, uses differently colored and shaped blocks with text labels as instructions (Resnick et al., 2009). Similarly, ROBOLAB (Rogers & Portsmore, 2004; Vernier, n.d.) and LEGO® Education’s WeDo™ Robotics Software use blocks with icon and text labels to create instructions for robots. Figures D7 and D8 show “Hokey-Pokey” programs made with Scratch and WeDo™ and illustrate how complex these programs can become, even in an introductory, graphical language. These languages tend to work for older children, as they require precise eye-hand coordination and fine motor movements with a mouse and offer a large set of low-level instructions, making them less suitable for most kindergarteners.

Tangible interfaces bridge the physical and digital world and have been explored for their potential to make interacting with a computer more intuitive and less demanding of fine-motor and eye-hand control. They are concrete systems of objects that a user physically moves to manipulate digital data, images, etc. (Ishii, 2008; Patten et al., 2000). Examples tend to come from the realm of research rather than commercial products and include the Slot Machine (Perlman, 1976), AlgoBlock (Suzuki & Kato, 1995), GameBlocks (Smith, 2007), TurTan (Gallardo, Julià & Jordà, 2008) and the physical blocks of CHERP (Horn et al., 2011). Tangible interfaces reduce the physical requirements of programming by eliminating the need to type or use a mouse to click on small objects and drag them to equally small targets. Like graphical interfaces, they can provide a more direct mapping between the digital information and its manipulation to the representations of these in the programming environment. Because tangible languages tend to have small instruction sets to avoid being cumbersome, ease of use is gained at the cost of the complexity of programming possible. Therefore, tangible interfaces work well for introductory or specific-purpose programming languages which do not need as many instructions as more general languages (Perlman, 1976; Sharf, Winkler & Herczeg, 2008).

It is widely assumed – although without quantitative evidence – that tangible interfaces appeal to young children's concrete ways of thinking and reduce the cognitive load of programming, allowing for more effective or efficient learning than with graphical or text-based interfaces (Marshall, 2007; Horn et al., 2011). In fact, when a tangible interface differs only in its three-dimensionality from its corresponding graphical interface, studies show learning to be equal with either interface (Horn et al, 2011); more substantial differences may exist, but current research methodologies have not yet provided a thorough understanding of them. Nonetheless, tangibles do offer an appealing and engaging interface, especially for preschoolers and kindergarteners, (Horn, et al., 2008), as well as a tool that is easier to use in terms of fine motor skills and eye-hand coordination.

CHERP takes advantage of the benefits of both graphical and tangible interfaces by combining them in a hybrid interface. Its graphical and tangible programming blocks, labeled with icons and text, can be used interchangeably (Horn, et al., 2011). See the Chapter 2 section on TangibleK technologies for more details on the CHERP interface.

Another genre of programming tools embeds programming instructions in the objects to be programmed. Examples include Topobo, a construction kit of connectible plastic pieces and specialized parts that memorize and repeat movements (Raffle, Parkes & Ishii, 2004), Electronic Blocks, a small toy car that behaves differently when physical logic and sensor programming blocks are attached to them (Wyeth & Wyeth, 2001; Wyeth 2008), and the BeeBot, Pixie, and Roamer floor robots, which are all programmed to move around by pushing a simple set of buttons on its back. The input style of Topobo is also similar to ‘programming by demonstration,’ whereby the programmer uses familiar ways of interacting with a computer to give it instructions, which the computer then generalizes and can apply in other, specified settings (Smith, Cypher & Tesler, 2000). ToonTalk, a graphical programming language, is similar to these physical languages; it introduces programming through a 3D world in which each type of object behaves according to a modifiable instruction or rule (Kahn, 1996).

Whether based on text, graphics, or physical objects, each type of interface has benefits and drawbacks. Each of the different types of programming tools described here require different levels of sophistication in a child’s computational thinking and are therefore suited to children at different levels of cognitive development and programming experience as well as to different goals for the programming activity. While removing abstraction generally is a simpler way of interacting with a computational device, interfaces that rely on completely embedded interactions (i.e. that not at all abstracted) are not true programming languages. They reduce or eliminate many of the core elements of computational thinking and programming, such as a persisting representation of an algorithm and abstraction between programming instructions and the action or outcome they represent. Programming languages for young children should retain the core cognitive aspects of programming that are accessible to each age group while striving to do so in as straightforward a manner as possible.

Research on children's programming languages and interfaces has addressed a wide range of topics including effective interfaces (Horn et al., in 2011; Marshall, 2007), collaborative programming tools (e.g. Farr, Yuill & Raffle, 2010; Fernaeus & Tholander, 2006), and integration of programming activities with other content domains in schools (Bers, 2008). Advancements in human-computer interface (HCI) designs have lowered the barriers for children's entry into sophisticated computer programming with increasingly developmentally appropriate interfaces (Horn et al., 2011). While HCI research seems to be shifting towards collaborative interfaces made possible by new developments in computer hardware (e.g. Fernaeus & Tholander, 2006), essential questions about effective programming interfaces, languages, and interaction styles remain to be thoroughly answered.

In addition to features of programming languages, interfaces, and interaction styles, both developmental and individual characteristics impact how a person, especially a young child, approaches programming and succeeds with different programming tools. The following chapter reviews pertinent cognitive factors from developmental and other perspectives.

Chapter 4: The Cognitive Development of Four- to Six-Year-Olds

Contemporary Perspectives on Piaget

Decades of work by Jean Piaget on the nature and development of children's thinking fundamentally transformed developmental psychology. The current field of cognitive development continues to draw on core tenets of his theory, for instance, the active and internally motivated construction of knowledge and the presence of coherent cognitive patterns at different ages (Flavell, 1996). His work has also been extensively re-evaluated and extended to incorporate new ideas about cognitive developmental processes (Flavell, 1996). The focus of much of this work has been to specify the mechanisms by which stage transitions occur or how new cognitive structures arise, and to address empirical evidence of individual differences in these transitions that conflict with the original theoretical model.

Piaget theorized that changes in thinking, including cognitive stage transitions, occur due to a process he called equilibration (Feldman, 2004; Lewis, 2000; van Geert, 1998). As children try to fit their experiences into the model of the world they currently hold with the rules of reasoning characteristic of their development (assimilation), they realize that some things about the world do not make sense within their current framework, and that this framework must change (accommodation) (van Geert, 1998). This dissonance drives efforts to make better sense of experience through more logical and sophisticated variations on existing thought processes. Eventually, according to this theory, children reach a break-through point, a complete and qualitatively different set of cognitive structures. This explanation has left many unsatisfied in that it does not account in a compelling way for why ongoing conflict between perceptions and internal models of the world should suddenly result in the massive changes that are seen as children move from one stage to the next (Feldman, 2004; Lewis, 2000). Also warranting further explanation is the reality that individual differences exist along the cognitive developmental trajectory more than Piaget's theory addresses (Feldman, 2004). Several lines of theory and methodology have been applied to address this question.

Neo-Piagetian theories represent attempts to rework and build on Piaget's original theory, often describing and explaining the mechanisms of stage transitions by integrating roles for maturation and learning. Two pre-eminent neo-Piagetian theorists are Robbie Case and Kurt Fischer. Case's work (1984) maintains the four-stage structure of Piaget's model but draws on information processing rather than symbolic systems to describe each stage's cognitive structures. Each stage in his theory consists of a cycle of four sub-stages (Feldman, 2004). Transitions between the large stages themselves, he claims, arise from hierarchical integration of multiple, previously existing problem solving or executive control structures, with the support of relevant experience and increasing working memory (Case, 1984). As children mature and as they streamline well-practiced strategies, they have more cognitive 'space' available to carry out mental operations or store information in working memory. Because of this, children become able to execute multi-faceted lines of reasoning or problem solutions.

Fischer's skill theory (1980) includes three levels, or stages, of cognitive development, rather than Piaget's typical four. These levels are distinguished by children's mental models being sensorimotor, representational, or abstract in nature. Within each level, children's cognition relies on a set of skills that progress through four hierarchical structures of relationships. Fischer defines five types of transformations in these relationships to describe development within and between the broader levels (Fischer, 1980). By specifying in great detail how cognitive transitions from relatively simpler to more complex structures and abilities occur, and by postulating a cycle of cognitive structures that is repeated within each stage, neo-Piagetian theories like these attempt to clarify and broaden the empirical validity of Piaget's theory.

The microdevelopment perspective also stems from the question raised in Piaget's theory about the nature of transitions. It specifically delves into the mechanisms driving developmental transitions through a methodology of intense observation over short timeframes (Granott &

Parziale, 2002; Kuhn, 1995). Since extensive practice facilitates the generation of new strategies, and the accumulation of these new strategies ultimately build up to developmental shifts over time (Case, 1984; Kuhn, 1995), frequent observation of a cognitive activity increases the likelihood of researchers observing the process of change (Kuhn, 1995; Siegler & Crowley, 1991). Possibly, a fully explicated theory of macrodevelopment – across childhood and into adulthood – could be obtained by integrating sufficient quantities of microdevelopmental data. Developing an empirically tested theory of the relationship between microdevelopment and macrodevelopment is a prominent trend in microdevelopment research (Granott & Parziale, 2002).

Another relatively recent approach to understanding developmental shifts in cognitive structures is the dynamic systems perspective, which does not derive from Piagetian theory but is borrowed from the physical sciences. It explains the growth of novel and complex structures – whether molecules or cognitive structures – out of existing, simpler ones through the principle of emergent phenomena (Lewis, 2000), which a number of recent cognitive development theories have incorporated. Dynamic systems theories do not cast the cognitive system as containing specialized structures which inherently pre-determine developmental outcomes, or as limited in potential growth based on learning opportunities (Lewis, 2000). Rather, emergence-based theories view cognitive structures as self-organizing systems: systems which develop through the selective use of advantageous new structures that random interactions among existing structures have produced (Lewis, 2000). This model provides a scientific – and sometimes highly mathematical – explanation for the appearance of completely new structures within stages as well as the building up of complexity to the point of stage transitions (Lewis, 2000; van Geert, 1998). It also explains how cognitive development across individuals can at once follow broad patterns and yet be influenced by unique circumstances (Lewis, 2000). As neo-Piagetian, microdevelopmental, and dynamics systems models delve into the inner workings of developmental transitions, another model attempts to improve upon Piaget's original theory without deviating from its essential tenets.

Feldman-Revised Piagetian Stages

Feldman (2004) offers a revision of Piaget's stages and transitions that adds coherence to the theoretical description of stage transitions while retaining as much of the core theory as possible. The theory maintains Piaget's basic assumption that the human mind naturally strives towards more accurate and versatile mental structures. It also incorporates the principle of emergence common among other neo-Piagetian theories and systematically takes into account the uneven and gradual progress that a child makes in moving from one stage to another (Feldman, 2004). About halfway into each stage, children move from a phase of actively constructing new systems of thought to a phase of energetic application of those systems that begins to bridge the gap between the current and upcoming cognitive structures (Feldman, 2004). The pre-operational, or intuitive, stage of cognition covers ages two to six as a whole, so a four-year-old child is beginning to actively extend the symbol systems s/he has developed from the age of two, applying them to interactions in the physical and social world and through them constructing theories about how the world works (Feldman, 2004). This process is grounded in intuitive reasoning and forms a precursor to the increasingly adult-like logic of the next stage, concrete operations, which is characterized by empirical observation, organization of objects and their qualities, and mental modeling of actions and perspectives not taken in physical reality (Feldman, 2004; Gardner et al., 1996).

Children in the TangibleK study ranged from about four-and-a-half to six-and-a-half years old, a timespan during which children experience “major” cognitive growth (Case, 1984, p. 25). Characteristics of both Feldman-revised Piagetian stage phases that cover this period are presented in the following section. Their descriptions highlight how salient many key thought processes are to reasoning and problem solving and therefore to computer programming and understanding how digital objects work.

Pre-operational thought and symbol systems.

The defining characteristics of the pre-operational stage of cognitive development, which roughly covers ages two to six, are the acquisition and application of culturally-learned symbol systems as well as patterns of reasoning which are immensely compelling to the child and yet which seem illogical compared to formal, adult-like logic (Feldman, 2004; Gardner et al., 1996). During these years, a child learns an impressive amount of language, as well as other systems for representing objects and concepts like quantity symbolically (counting, time-keeping, etc). Piaget used the term *figurative* for such interactions: the child internalizes some aspect of the world as it is (Feldman, 2004). The child’s relationship to symbol systems shifts halfway through pre-operations, when s/he begins to emphasize more *operative* thought processes: the appropriation and application of the developing symbol systems to the child’s explorations and analysis of the world (Feldman, 2004). The point of change, sometimes called a ‘seizing of consciousness,’ represents a relatively sudden shift in the child’s mind as s/he has already built up sufficient cognitive structures and knowledge of symbol systems, and s/he now sees how to put these tools to use for a great number of purposes (Feldman, 2004).

Children in the second phase of pre-operations devote considerable energy to asking about and formulating theories on how different parts of the world work and interact, although these theories tend to defy adult logic (Feldman, 2004). The child at this point of cognitive development is most notably unable to *conserve* quantities of different types – s/he would say that a tall, slim glass of water contains more water than a short wide glass that is filled with the liquid from the first glass (Gardner et al., 1996). This phenomenon results from several other characteristics of the stage: *centration,* or narrow focus, on a single feature of an object or event, reliance on perception rather than logic, and *irreversibility,* the inability to mentally undo an action (Gardner et al., 1996). Children in this stage also tend to believe that all people see things from the child's own physical and mental perspective (Feldman, 2004) and confound psychological and physical events (McDevitt & Ormrod, 2002). The result is often *transductive reasoning*, reasoning based on unrelated observations brought together in syncretistic ways (McDevitt & Ormrod, 2002). While the four- or five-year old child is certainly hard at work theorizing about how the world works, the cognitive tools s/he employs to construct those theories are quite different from the tools available to older children and adults, and thus their conclusions differ drastically as well.

The transition towards concrete operations.

Feldman’s (2004) revisions to Piaget’s stages systematically account for the facts that different mental processes from the concrete operations stage appear before others across children and that each child begins using these processes only inconsistently at first. The major characteristics of the period of transition between pre-operations and concrete operations are that the child becomes interested in exploring concepts and cognitive processes that previously seemed irrelevant, and that some of the concrete operational abilities appear, albeit inconsistently. In general, the child gradually uses symbol systems to begin constructing categories, hierarchies, and other relationships between objects, ideas, and events in his/her own experience, although the child’s grasp of these concepts will not solidify until later on (Feldman, 2004; Gardner et al., 1996). During this period, children may switch back and forth between patterns of thought characteristic of the developmental level they are leaving and the level they are entering (Feldman, 2004).

Concrete operations.

At around six years, a child is likely to be making the switch from pre-operational to concrete operational patterns of thinking (Lightfoot, Cole, & Cole, 2009). S/he is now familiar with representing the world with both mental and physical symbols. The work of the second phase of pre-operations, actively elaborating on and applying symbol systems and using intuitive reasoning to build understandings, has prepared the child’s cognitive structures for transformation towards those required for more logical reasoning (Feldman, 2004). At first, the child shows interest in perspectives and cognitive challenges not previously attended to, and over the next two or three years, the child consolidates the cognitive structures necessary to successfully apply this range of new skills (Feldman, 2004).

Children this age begin to consider multiple aspects of situations or objects simultaneously, which allows them to work with conservation of quantities and the concepts of categories and hierarchies (Feldman, 2004). A six-year-old increasingly understands physical and psychological points of view other than his/her own, for instance in conversations with peers, which become more bidirectional and less like “collective monologues” (Lightfoot et al., 2009, p. 263). Around this time, a child also relies increasingly on logical reasoning about causal relationships and the distinction between appearances and reality (Lightfoot et al., 2009) by letting logical conclusions take precedence over perceptions (McDevitt & Ormrod, 2002). The six-year-old becomes more able to plan a series of actions to fulfill a goal and to think flexibly in doing so (Lightfoot et al., 2009). While a child this age typically uses concrete materials to build mental models or representations (Feldman, 2004), s/he does not rely on the physical object’s immediate presence to mentally consider and manipulate it (Lightfoot et al., 2009). Children’s cognition in this stage is also aided by increasing memory capacity and meta-cognition (Lightfoot et al., 2009).

Cognitive Styles

Beyond the broad impact of cognitive development, cognitive styles also influence the ways a person tends to approach a wide range of cognitive activities. Varying definitions of cognitive styles, which can overlap with personality traits and learning styles, are found within the literature, but cognitive styles are generally considered to be reasonably stable individual differences in modes of perceiving, recalling, and thinking about information and experience (Kozhevnikov, 2007; Messick, 1984, as cited in Bishop-Clark, 1995; Shipman & Shipman, 1985). Each cognitive style is conceived of as a continuum or dipole with opposite modes of cognition on either end. For example, on the analytic/holistic spectrum, a person who is more analytic tends to be structured and logical in their problem-solving and looks to distill a problem into core factors and their relationships; a holistic person is more likely to use common sense as a problem-solving approach and rely on guess-and-check strategies (Bishop-Clark, 1995). Research on cognitive styles has shown correlations to performance in academic, vocational, and social domains (Bishop-Clark, 1995; Kozhevnikov, 2007; Sternberg & Grigorenko, 1997). However, a coherent theory integrating existing research on cognitive styles and achievement has yet to emerge (Kozhevnikov, 2007; Sternberg & Grigorenko, 1997).

People tend toward one mode or the other of each cognitive style continuum, although variation within individuals across contexts exists as people attempt on some level to apply the best strategies for a given situation, even in the face of natural preferences (Gilbert & Swanier, 2008; Kogan & Saarni, 1990; Kozhevnikov, 2007; Krechevsky, 1998; Sternberg & Grigorenko, 1997). The expression of some cognitive styles also depends in part on a person's level of cognitive development and is therefore expected to change over childhood and into early adulthood. For instance, one cognitive style which has been shown to evolve with development is conceptualizing style, also called categorizing style, which measures the breadth and type of categories people define to sort objects or concepts (Shipman & Shipman, 1985). Despite evidence of variability within individuals, many studies of cognitive styles assume their stability over time and contexts, at least within the timeframe and content of the study.

Cognitive styles may have important implications for learning to program because of their relationship to reasoning and problem-solving. Styles of particular interest include the analytic/holistic style (described above), the reflective/impulsive style, and internal/external locus of control (Bishop-Clark, 1995). When faced with a difficult problem, reflective people usually consider the implications of various possible solutions before trying one; impulsive thinkers – who are distinct from those with impulsive personalities – tend to act quickly and have little worry about making errors or achieving the best outcomes (Sternberg & Grigorenko, 1997). Locus of control refers to whether a person feels that s/he has the power to influence aspects of the surrounding world or that s/he is a product of the environment (Bishop-Clark, 1995).

These cognitive styles may impact distinct stages of computer programming differentially, as opposed to relating to programming as a whole. Bishop-Clark (1995) proposed that four sets of correlations might exist: a) introversion/extroversion and field dependence/independence with problem representation, b) reflectivity/impulsivity and field dependence/independence with solution design, c) the Myers-Briggs thinking/feeling dimension with coding, and d) locus of control and reflectivity/impulsivity with debugging (Bishop-Clark, 1995). No follow-up research to this proposed framework was found in the literature.

Research has also supported field-independence, the ability to distinguish an object or form from its background context, as influencing achievement by novices in programming (Mancy & Reid, 2004). However, the construct may actually be conflated with aspects of intelligence including spatial ability and the ability to think flexibly in a novel situation (Sternberg & Grigorenko, 1997). Other cognitive or personality traits that may prove relevant include locus of control, persistence, and attitudes about computers – such as anxiety and anthropomorphism (Ackermann, 1991; Bishop-Clark, 1995; Levy & Mioduser, 2008; Perkins et al., as cited in Robins et al., 2004). Cognitive factors which vary developmentally as well as individually can also be expected to differentially impact programming ability. For instance, the cognitive gains from children's working memory capacity significantly increasing over the course of early childhood and beyond may support reasoning abilities needed to learn a programming language (Lightfoot et al., 2009; Reisberg, 2010).

Contextual Factors

Many factors external to the child's mind also impact learning outcomes. Parental involvement in primary school education shapes children's self-perception as learners and expectations for achievement, indirectly but significantly influencing educational outcomes (Desforges & Abouchaar, 2003). The nature and intensity of parents' involvement in education varies over a number of factors, including: socio-economic status, mother's highest level of education and mental health, number of parents in the household, age of the child, encouragement by the child for parents to become involved, and the child's current level of academic achievement (Desforges & Abouchaar, 2003).

Only a small portion of the literature on parents' role in education addresses children's outcomes in early elementary school science, technology, engineering, or math (STEM). However, research has shown that parent's positive views of science lead them to engage in science-related activities with their older children. These shared activities show children that their parents perceive science as a worth-while and interesting subject and foster children's self-image as capable of learning science, in turn leading to increased achievement in school science (George & Kaplan, 1997). On the other hand, elementary school children and their parents seem to also share limiting notions of gender stereotypes regarding math and science achievement and careers (Andre, Whigham, Hendrickson & Chambers, 1997). Achievement is tied to many factors, and important among them are parental attitudes and involvement in education at home. As the inclusion of STEM fields in elementary school expands, new research may further describe how early at-home exposure to positive attitudes and experiences with these domains impacts school achievement.

Cognitive and contextual factors, particularly the Feldman/Piaget portrait of the four- to six-year-old's cognition, provide a developmentally-based starting point for examining how children engage in problem-solving through programming robots. Having a sense of whether a child tends toward pre-operational or concrete operational reasoning patterns, along with knowledge of the cognitive processes required by a programming environment or activity, can inform the specification of appropriate learning expectations, curricula, and support activities for different children. These understandings can also lead to programming technologies which are (re)designed to meet the needs of children at specific points in their cognitive development. The following section presents the methodology for examining the relationships between children's cognitive development and their approaches to and achievement in programming. Other cognitive, demographic, experiential, and parental factors are also considered as elements of the larger picture that development and specific activities fit into.

Chapter 5: Research Design

The analysis presented in this thesis draws on the TangibleK laboratory-based study, which was designed to describe in detail what kindergarteners can understand about programming and robotics, their learning trajectory through the relevant concepts, and how different interfaces might impact their learning. The experience of conducting the study sessions raised questions about why children exhibited such a range in their uses of the programming language and in their success in programming a robot to dance the “Hokey-Pokey.” Some children were simply uninterested in the challenge – although they had plenty of other ideas for using CHERP; other children were interested but thoroughly stumped by the task; and yet others enjoyed and solved it quickly. Cognitive developmental level was hypothesized to be a crucial element in understanding the variability of children’s focus and acquisition of programming skills.

Research Questions

This analysis sought primarily to answer the question: **What patterns may be found in children’s approaches to and achievement in programming based on their level of cognitive development?** Secondary follow-up questions included:

- How do individual differences in sequencing ability, certain cognitive styles and personality traits, prior related experience, and socio-cultural background predict deviations from developmentally expected trends, as defined in response to the primary research question?
- How do the difficulties experienced by the preschool and kindergarten participants of this study compare to the common difficulties older novice programmers have?

Hypotheses

The general hypothesis regarding developmentally-based differences in programming was that children would differ according to their level of cognitive development on their approaches to programming and that this would lead to differentiated achievement. More specifically, children in Piaget's pre-operational stage were expected to be less likely to go beyond making correct correspondences between individual robot actions and individual programming instructions in their programming. Conversely, children near or in concrete operations were hypothesized to be more likely to systematically work towards a correct sequence of programming instructions to accomplish a specific goal. Furthermore, it was hypothesized that factors which are either relatively stable over time, such as cognitive styles, or which are experience-based, such as prior computer experience and family-related factors, would interact with level of cognitive development, with certain conditions enhancing or mitigating developmental impacts on programming approaches and achievements.

Variables

The conceptual variables used to answer the primary and secondary research questions ranged from children's level of cognitive development, approach to programming, programming achievement at two levels, sequencing ability, demographics, and prior experience with computers, robotics, and programming, to parents' level of education and prior experience with programming and robotics (see Table B1 for an overview of variable definitions). Some of these variables were measured at the time of the study, but several required measures to be created and assessed from video footage of the study sessions. This section discusses each variable and how it was measured. See Tables B2 and B5 for summaries of the primary and secondary analysis measures and their derivations.

Cognitive developmental level.

A large part of the work in this analysis involved creating a framework for measuring children's stage of cognitive development, the primary independent variable, because no metric exists to assess it from a task like programming. However, programming is a rich context for assessing logico-mathematical and deductive reasoning, the types of thinking that most informed Piaget's theories (Case, 1984) and which form much of the description of cognitive stages in the literature. The framework was designed to categorize each child into one of three developmental categories based on Feldman's revised Piagetian model and the age range of the sample (Feldman, 2004). These categories were: pre-operations, phase 2; transitional; and concrete operations, phase 1. Characteristics of the late pre-operational and early concrete operational stages were compiled from a review of the literature and mapped to programming behaviors which were logically expected to result from the cognitive characteristics and which could be observed from video footage of the study sessions. For instance, if children only begin to empirically test their ideas late in pre-operations, then a developmentally younger child in the study's age range will be unlikely to test and revise any programs, whereas a developmentally older child will be more likely to watch his/her robot run a program as a means to find out how close it is to the solution.

To arrive at an overall stage measure, developmental level sub-scores were give to three core elements of children's approach to the programming challenge: the child's goal in programming, the strategy used in the initial solution, and the strategies used in debugging (iteratively testing and revising) the program. An overall developmental level was determined for each child based on the trend of these three component scores and by taking into consideration other operations relating to physical versus psychological agency and perspectives, and multiple classifications as needed. Table B3 provides all development-to-programming mappings and the rubric used for assessment can be found in Table B4.

The mappings of developmental characteristics to programming behaviors and strategies are based on the overall intuitive versus logical nature of the child's thought processes and on the child's ability to work flexibly towards a given goal. These cognitive characteristics were chosen because they evolve over the age range of the sample, for their relevance to programming, and because they are observable in the existing video data. Two general patterns of programming approaches are derived from the literature on cognitive development traits during different stages and were also observed in the study: 1) a shift from interest in self-defined, exploratory goals in pre-operations towards interest in a given, structured goal in concrete operations, and 2) a growth from intuitive to increasingly logical strategies for problem solving. (When this pattern did not hold true, it seemed that interest preceded ability. Some children who ultimately were scored as pre-operational were intrigued by the "Hokey-Pokey" task, but they were at a loss to find a successful strategy.)

Children in phase 2 of pre-operations have recently developed the cognitive structures necessary to elaborate and apply on familiar symbol systems (Feldman, 2004). However, they would not be expected to have already developed the abilities needed to plan flexibly towards a given goal. Therefore, we expected to see children in this phase use CHERP prolifically and enthusiastically but without pursuing the assigned goal (or at least not too far). Children nearing the end of pre-operations and transitioning toward concrete operations should have more interest in pursuing the given goal, and might be able to make some attempts at it, but their new structures of logical thought would not be sufficiently consolidated to see the "Hokey-Pokey" goal through to completion. We expected children in concrete operations to be interested in a problem of this nature and to have developed enough of the logical reasoning needed to systematically and successfully solve a problem of this level of difficulty.

Programming approach.

Approach to programming was intended to be a distinct dependent variable in the primary analysis. However, it overlaps to a great extent with expected characteristics of programming at different stages of cognitive development as defined above. Children's developmental level of goal orientation, initial strategy, and debugging strategy each are measures of programming approach – but here they are used as a proxy for cognitive developmental level (see Table B4). One of these measures examines children's level of interest in using CHERP for the given goal as opposed to a self-assigned, open-ended exploration. The other two examine whether children relied on intuitive versus logical strategies, or something in between. Children's scores on these measures are not necessarily identical to their overall developmental level score, so each sub-score can potentially correlate differently to programming achievement than developmental level does, and analysis of achievement by sub-scores can be presented. However, it would be necessary in future iterations of this study to employ separate measures of cognitive development and programming approach to validate the presently assumed – and theoretically based – relationship between cognitive development and various elements of approach to programming.

Programming achievement.

Programming achievement, a second dependent variable in the primary analysis, was divided into two levels of understanding: *correspondence* and *overall completeness* of the child's final program. Correspondence was defined as the ability to purposefully match programming instructions to planned robotic actions. This measure was based on whether instructions the child chose matched the "Hokey-Pokey," but did not measure whether the child chose *all* the necessary instructions. It was assessed on a Likert scale from 0 (cannot achieve) through 5 (achieves without assistance), with the intermediary values represent increasing levels of support needed by the child to successfully apply this cognitive skill (see Appendix B5 for the full rubric). By the time it was evident that some children were not going to be able to solve the challenge independently, they were also too tired to focus on solving it with support. If reasonable, correspondence scores in these cases were estimated based on the level of difficulty and reasoning exhibited by the child during the independent work period.

Program completeness is a composite skill requiring not only making action-instruction correspondences but also sequencing the instructions correctly according to the last verse of the “Hokey-Pokey” song. This measure was assessed based on how many changes would need to be made to a child’s program for it to match a defined set of possible solutions. The scale for program completeness ranged from 0 (unrecognizable as an attempt at the “Hokey-Pokey”) to 4 (all the required instructions, in exact order). Only the actions corresponding to the first five lines of the “Hokey-Pokey” verse were scored. Whether a child correctly used the ‘Begin’ and ‘End’ instructions or included sounds at the start or end of the program were disregarded. Children received full credit for reasonable replacements for actions, such as a series of ‘Turns’ instead of ‘Spin,’ and received lower scores for instructions that were out of order, missing, or extraneous (see Appendix B6).

As with programming approach, these two measures of programming achievement are inherently tied to the measures used to assess cognitive development in this analysis. For instance, a characteristic of pre-operational thinking, the use of a guess-and-check problem-solving strategy, inherently correlates to less effective use of correspondence, one of the key programming skills measured in the TangibleK study. Although the two sets of measures, programming achievement and cognitive development, are not completely overlapping, the theoretical model predicts conflation between the measures. As with programming approach, future versions of this study should use independent sources of programming achievement and cognitive development data to resolve the lingering question here of what portion of the results stem from the overlap in measures and what portion demonstrates developmental patterns in programming.

Individual and contextual factors.

Secondary independent variables were also measured where possible to examine other factors which might interact with cognitive development and correlate with programming achievement. These additional variables include: pre-intervention sequencing ability and the change in sequencing scores following the intervention; demographic information; children's prior experience with computers, robotics, and programming; parental education attainment; and parental background in STEM degrees or careers (see Table B5).

Cognitive styles and other personality factors would have been of great interest to this analysis, particularly the analytic/holistic style, the reflective/impulsive style, and locus of control. However, these constructs posed methodological problems in their assessment from the existing data set. Impulsivity/reflectivity and holistic/analytic cognitive styles parallel key changes in thought patterns during the growth from pre-operational to concrete operational thinking, making it difficult to disentangle the relative influence of each factor from a single data source. (Adults with an impulsive cognitive style tend to act without as much planning or focus on best outcomes compared to those with reflective styles, which mirrors some of the developmental changes from pre-operations to concrete operations. Similarly, holistic thinkers rely on intuition, as do pre-operational children, whereas analytic thinkers rely on reason, as do concrete operational children, overall.) There were not enough data to guarantee an indication of locus of control for every child from a set of activities which were not specifically designed to measure this construct. Therefore, for the purpose of the analysis at hand, observed cognitive patterns are assumed to be primarily developmental, with some apparent exceptions explored in Chapter 7 (Discussion), and analysis of potential impacts by cognitive styles and personality traits on cognitive development and programming are delegated to future follow-up studies.

Sequencing.

Children's sequencing ability, a cognitive baseline assessment, was measured prior to and following the three programming sessions. Baron-Cohen's Picture Sequencing assessment, a validated and standardized metric, was used (Baron-Cohen, Leslie & Frith, 1986). Five stories were used as a pre-assessment and another five were used as a post-assessment. The original ratio of story types was maintained in each session. This analysis uses two measures based on the picture sequencing assessments: children's pre-intervention picture sequencing score, and the difference between their pre- and post-intervention scores.

Demographics.

This category of variables included the child's age (based on birthday and first study session date), gender, grade in school, and whether they lived in a suburban or urban neighborhood. Parents supplied this information on a questionnaire prior to their child's second study session.

Child experience.

Information on the child's prior experiences with computers, robots, and programming was documented in the parent questionnaire as well as in an oral survey children responded to during their first session. Measures of prior experience included: whether the child used a computer at home (yes/no); the child’s level of computer expertise compared to age-mates, as perceived by the parent (beginner/average/expert); whether the child had programming experience (yes/no) and the type of robots the child had previously encountered (none; movies, books, and pre-programmed toys, or programmable robots). The categories of robotics experience were inferred from broader categories parents checked off or listed on the survey.

Parental education and experience.

Parental level of education, involvement in STEM-related fields, and experience with programming and robotics were documented in the written survey parents filled out prior to their child’s second session. Parental education was measured for this analysis as the highest level of education attained by either parent: high school, some college, bachelor’s degree, master’s degree, or doctoral degree. Involvement in a STEM field was measured two ways, based on parents’ descriptions of their job and degree fields: 1) whether the latest degree of either parent was in a STEM-related field, and 2) whether either parent in the family had a STEM-related job. Parents’ experience with programming and robotics was captured in two yes/no measures documenting whether at least one parent had experience with each field.

Sample

The original TangibleK laboratory-based study included a total of 36 children: 34 who completed all four sessions and another two who only completed the first two. Recruitment occurred via the TangibleK website, emails to relevant DevTech Research Group contacts, and the snowball effect. All participants came from several local towns and cities.

Inclusion criteria.

The requirements for inclusion in the data analysis of this thesis are as follows:

- attendance of at least the introductory group session and the first individual session,

- existence of adequate video or written data to assess and validate the necessary measures,
- no evidence that the child was insufficiently comfortable in the study setting or unable to focus on the activity, either of which violates the assumption that assessments are based on each child's best efforts under standard conditions, and
- sufficient communication by the child, either verbally or behaviorally, of his/her goals, ideas, and thought process during programming and debugging to assess the measures.

Of the initial 36 children, 29 children met all the criteria for inclusion in data analysis. Seven children were excluded from the present analysis due to at least one of the following factors:

- high levels of distraction or shyness during the activity,
- having shows significant difficulty in solving the problem and then, after support was given in structuring the problem solution, surprisingly high levels of independent and systematic strategizing, and/or
- choosing to undertake a specific but different challenge than the one assigned, which made it hard to compare success in programming to that of children who took on the given challenge.

Sample demographics.

Of the 29 children included in analysis, 11 children, or 38% of the sample, were girls, and 18 children, or 62% of the sample, were boys. Kindergarteners (20 children) made up 69% of the sample and preschoolers (9 children) made up the remaining 31% of the sample. Ages, collected for 28 of the 29 children, ranged from 4.4 years to 6.6 years at the time of each child's first session, with the mean age at that session being 5.6 years old. The group was split about evenly between urban and suburban neighborhoods. Eleven children (38%) attended public schools and

18 (62%) attended private schools. (Note that public preschools are less available than public kindergartens, which, along with the pool of initially recruited families, probably accounts for a large part of the skew.)

Almost half the families (43%) had at least one parent with a master's degree, and almost another half of families (46%) had a parent with a doctoral degree. Only two parents had attained less than a bachelor's degree, and in no families was the highest degree lower than a bachelor's degree. The sample included children from several ethnic and cultural backgrounds including several bilingual children, although data was not collected specifically on these points. Three-quarters of the children used a computer at home, about a third had played with programmable robots, and none had programmed, according to their parents. (There appear to be some discrepancies between researchers and parents on this last point regarding what activities constitute 'programming.')

Data Collection

The data were collected during one-on-one work with preschoolers and kindergarteners during a TangibleK Robotics Project study as well as afterward, from video footage and notes of the sessions. The study collected detailed information on what and how four- to six-year olds learn about programming and robotics, given developmentally appropriate tools and lessons (Bers, 2010). Over the course of the study, each participant attended four sessions. The first was a group session (usually three to four children) for pre-assessments and an introduction to the robotics and programming technologies. Participants also attended three individual sessions in which children learned a new programming concept and attempted a specific challenge. At the end of the third individual session, post-assessments were also completed.

During the individual sessions, each child reviewed previously taught programming instructions and concepts, built a robot, and tested that it worked. Once s/he had a sturdy and functional robot, the researcher provided instruction on new programming and robotics concepts. The child was presented with a related programming challenge and given time to try to solve it without conceptual assistance from the researcher. After a set amount of time, the researcher assisted the child in completing any unfinished aspects of the challenge. Finally, the child was asked about his/her understandings of core concepts from the activity. Each child who completed all sessions spent about five hours in the study.

The analysis presented here focuses on one activity in particular: the programming challenge during the first individual session. During this session, the child learned and reviewed all the programming instructions and how to build a program out of physical or on-screen blocks, send a program to the robot, and run the program. Other than the 'Begin' and 'End' blocks, which all CHERP programs must have, this activity used only 'action blocks,' instructions which directly correspond to a single robotic action (see Figure D5). The programming challenge in this session was to create a program with CHERP that would make the robot dance the last verse of the well-known children's song the "Hokey-Pokey," which the child and researcher had sung and danced to ensure the child's familiarity with the song and its sequence of actions before programming. The following outline shows how it was expected that children would map the last verse of the "Hokey-Pokey" to CHERP instructions (also see Figure D6 for the CHERP graphical program). Some variations were counted as correct, particularly the exclusion of the less directly mapped musical instructions.

You put your whole self (robot) in,	→	Forward
You put your whole self (robot) out,	→	Backward
You put your whole self (robot) in,	→	Forward

And you shake it all about!	→	Shake
You do the Hokey-Pokey,	→	Spin
And you turn yourself (robot) around,		
And that’s what it’s all about!	→	Sing
(Clap, clap!)	→	Beep, Beep

Over the first two-thirds of the time allotted to programming, the researcher reflected back any questions the child had regarding how to solve the challenge. This ensured that the child was engaged with the researcher and felt supported. At the same time this procedure allowed assessment of the child’s independent ability to solve the challenge. Following the independent work period, the researcher provided whatever support was necessary for the child to accomplish the programming goal and understand the key concepts. This provided a feeling of success for all children and ensured that each child had an adequate minimum level of understanding of one set of concepts before moving on to the next. The child’s final program completeness was scored based on the independent work period. The amount of assistance the child required to achieve 10-15 skills and conceptual understandings, including the correspondence measure used in this analysis, was assessed at the end of the activity.

Pre- and post-intervention data collected on children's abilities to sequence picture stories also helped paint a picture of children's cognitive strategies. Standardized procedures for administering the Baron-Cohen Picture Sequencing cards were used (Baron-Cohen et al., 1986). The researcher presented the child with the first picture of a four-part picture-story and randomly placed the other three pictures near it. The researcher then asked the child to arrange the pictures to make a story, keeping the designated card first. Once the child put together the story, s/he was asked to narrate the story s/he had made. Children were scored on each story for whether they put the story in correct order, a defined partially correct order, or the wrong order, and a composite score was created by summing the scores on each item.

All background information on parents and children was gathered in a survey that one parent from each family filled out online or in paper form prior to the child's second study session. This included children's demographic information and prior experiences with computers, robotics, and programming as well as parental background in education, career, and robotics and programming experience.

Analysis Methods

Several statistical tests were used to determine the significance of the relationships between variables. The first stage of analysis tested relationships between a single predictor variable and one outcome variable. Chi-squares, t-tests, ANOVAs, and regressions were run depending on the categorical or continuous nature of the variables in question. To follow up on statistically significant initial findings, analysis of covariate tests (ANCOVAs) were used to examine the relationship of each of two predictors to a single outcome variable. Table B6 specifies the test used for each analysis.

Some of the statistical analyses include less than the intended sample of 29 children, for the two following reasons. The range of scores observed on measures of sequencing included several outlier scores, which were excluded. The only other cases in which children were excluded from a given analysis resulted from the parent having omitted a response on the background survey. Measures of development, programming approach, and programming achievement scores were complete for the 29 children.

Chapter 6: Results

The following results illustrate differences in programming approaches and achievement among children at each of three levels of cognitive development. For comparison, differences in programming are also explored based on children's sequencing ability, another cognitive – but not explicitly developmental – measure. Demographic, experiential, and parental factors are also explored for their potential impact on programming beyond that of cognitive development. Case studies are presented to exemplify cognitive patterns typical of the framework, and comparisons are made between the children in this study and older novice programmers in terms of the aspects of programming that proved difficult for them.

Developmental Patterns

Comparison of developmental levels.

Using the cognitive development framework described in Chapter 5, children were categorized as late pre-operational (phase 2), transitional, or early concrete operational (phase 1). Of the 29 children included in analysis, 8 children, or 28% of the sample, fell into the late pre-operational category; 7 children, or 24% of the sample fell into the transitional category; and the remaining 14 children, or 48% of the sample, fell into the early concrete operational category. Due to time and personnel constraints, interscorer reliability was not assessed for this measure, which ideally would have been carried out to verify the distribution of children among developmental levels and the characteristics of each group presented here. The skew towards concrete operational reflects the preferential recruitment and selection of kindergarteners over preschoolers to meet the original goals of the study. Each developmental level was comprised of generally the same proportions of each demographic, experiential, and parental characteristic as the overall study, although a few theoretically interesting differences among the groups existed.

The average age of children in each successive developmental category was higher than that of children in the preceding categories ($F(2,27) = 5.6$, $p < .05$). Pre-operational children (except one for whom this data could not be collected) were, on average, 5.1 years old (*SD* = 0.52) at the start of the study. Children in the transitional category were half a year older on average, 5.6 years old (*SD* = 0.69), and those in concrete operations were the oldest on average: 5.9 years old (*SD* = 0.40). However, only the difference between the mean age in lowest and highest developmental categories was statistically significant ($p < .05$). This finding makes sense in light of the wide range of ages within the transitional developmental level which overlapped with the age ranges of the other levels (Figure C1).

In each of the following comparisons, the overall sample ($N = 29$) was broken into six or nine groups along two dimensions: developmental level and a demographic or prior experience variable. These analyses show differences between the developmental groups in this study but do not imply conclusions about larger populations of 4- to 6-year olds.

There were no overall demographic differences between the children in each developmental level. Preschoolers were slightly but not statistically significantly overrepresented in the pre-operational category and underrepresented in the others; conversely, kindergarteners were somewhat overrepresented in the concrete operational category and underrepresented otherwise ($\chi 2(2, N = 29) = 3.64$; $p = 0.16$). Each developmental level had nearly the same ratio of boys to girls as the overall study – roughly 60% male and 40% female ($\chi 2(2, N = 29) = 0.10$; $p = 0.96$). The pre-operational group had slightly more suburban children and slightly fewer urban children than expected, but overall, the children were proportionally distributed by suburban / urban home area in each developmental level ($\chi 2(2, N = 29) = 4.49$; $p = 0.11$). Children attending private school were statistically significantly over-represented in the pre-operational category ($\chi 2(2, N = 29) = 7.16$; $p < 0.05$), but this can be attributed to the relatively lower availability of public preschools compared to public kindergartens.

Of the three measures of children's experience with computers and robotics, only one showed a statistically significant difference between the three developmental levels. Fewer of the children in the concrete operational level used a computer at home than expected by proportional representation, while in the lower developmental levels, more of the children used a computer at home than expected ($\chi 2(2, N = 28) = 9.33, p < .01$). Although the differences were not statistically significant, children perceived by a parent as having 'beginner' computer skills for their age group were over-represented in concrete operations while those perceived as 'average' were over-represented in the pre-operational and transitional levels ($\chi 2(2, N = 28) = 7.34, p = .12$). (This was tied to the fact that many of children in concrete operations did not use a computer at home.) Each developmental category had the same proportion of children at each level of prior robotics knowledge as the overall study ($\chi 2(2, N = 28) = 1.73, p = .79$). No children were reported by their parents as having programming experience, so there were no differences between the groups in that regard.

Children with parents who had programming experience were slightly over-represented in the pre-operational category ($\chi 2(2, N = 29) = 6.71, p < .05$). There were no differences between each developmental category in terms of parents' level of education ($\chi 2(4, N = 28) = 1.06, p = .90$), a parent having a STEM-related job ($\chi 2(2, N = 28) = 2.73, p = .26$) or degree ($\chi 2(2, N = 28) = 1.33, p = .52$), or parental exposure to robotics ($\chi 2(2, N = 28) = 3.11, p = .21$).

No statistically significant differences were observed among developmental levels in terms of average pre-assessment sequencing scores or change in sequencing score in the post-assessment. However, this relationship may be complex. Analysis with a larger data set is underway to further examine the relationship between developmental levels and children's improvement in sequencing after participation in robotics and programming activities.

The following sections describe the programming trends seen within each developmental category, with case studies presented in developmental order to exemplify the characteristics of cognitive strategies, programming approaches, and achievement representative of each cognitive developmental level.

Pre-operations, phase 2.

Eight children (28% of the sample) were scored as exhibiting the cognitive patters of later pre-operational reasoning, based on their responses to the “Hokey-Pokey” challenge. Half of these children disregarded the “Hokey-Pokey” challenge and instead focused exclusively on open-ended explorations of what they could make the robot do with CHERP. Two children claimed that their exploratory programs matched the “Hokey-Pokey,” perhaps out of a desire for the researcher to perceive them as compliant. The other half of the children scored as pre-operational did try to solve the “Hokey-Pokey” challenge, at least temporarily. They were unable to think of more than one or two actions of the solution and relied heavily on trial and error. It seemed easier for several of these children to start over than to revise their programs. Having exhausted their intuitive strategies, children waited for researcher assistance, expressed interest in doing a more familiar activity like building with LEGOs®, or moved to open explorations. Some of these children were concerned that they could not make satisfying progress toward the “Hokey-Pokey” goal, while others were excited by their general explorations and seemed not to be bothered by the unaddressed or unfinished challenge. Case Study 1 details the work of a child in the pre-operational category (described in Chapter 4 and Table B4) as he tried to solve the “Hokey-Pokey.”

Case study 1.

Caleb (name changed) was a four-and-a-half year-old at the time of the study, living in a suburban area, and attending a private preschool. He used a computer at home and was considered by a parent to have average computer skills for his age, but the TangibleK study was his first exposure to robotics. His parents' jobs and most recent degrees, a bachelor's and a master's, were in liberal arts fields. Neither parent had had any programming or robotics experience.

Caleb began the programming activity by exploring CHERP with the tangible programming blocks. He was particularly concerned about making sure that the robot really could carry out the action instructions he chose, 'Turn Left,' 'Forward,' and 'Shake.' "I'm going to see if it can do all of this," he said, and ran the program. He noticed with confusion that his program had five blocks (including the 'Begin' and 'End' blocks necessary in every program) but that the robot only did three actions. He had trouble understanding that programming blocks could have multiple categories of meaning. Once he was reminded that the 'Begin' and 'End' instructions work differently than action instructions, he still had trouble matching the robot's actions to the program he had made, but eventually did so with support. After a reminder to try the "Hokey-Pokey" challenge, he switched his focus to it. He tried hard to come up with a solution, but his strategies, which were characteristically pre-operational, left him stumped.

Caleb was excited about the idea of making a program that matched the "Hokey-Pokey" song. It would need "All the instructions! The moves!" he decided. He looked in the bin of tangible instruction blocks for 'Spin.' "That's one of the 'Hokey-Pokey!' ... We need the music," he added, humming. He was unsure what other instructions to choose, so the researcher sang the "Hokey-Pokey" verse as a reminder of its lines. Caleb enthusiastically pointed out the 'Spin' and 'Sing' blocks he had selected again. "But we still don't have 'put yourself in'…Maybe we can…I don't know what we can do to make it. Maybe we can find one on this computer and put it on.” He used the graphical programming blocks to build ‘Begin’ ‘Backward’ ‘Turn Left’ ‘Beep’ ‘End’ ‘Shake’ ‘Spin’ ‘Sing.’ “Let’s see if that’s the ‘Hokey-Pokey.’” He uploaded this program to his robot, not noticing the actions after the ‘End’ instruction, which would not be carried out. He did notice when the robot only enacted three of his six instructions, but he was unsure how to fix this. Caleb moved back to the tangible blocks, deciding, “Maybe I should put them right here (the spot for tangible blocks to be uploaded). I’m going to make a new one.” He quickly assembled ‘Begin’ ‘Shake’ ‘Backward’ ‘Turn Left’ ‘Forward’ ‘Spin’ ‘Sing’ ‘End’ without noticing which blocks he chose. This program was no longer made as an effort to solve the “Hokey-Pokey” challenge, but rather a general exploration of what he can make a robot do with CHERP. After watching the robot run this program, he was finished.

Caleb’s work on the “Hokey-Pokey” activity relied on intuitive rather than systematic strategies in both his recollection of the song’s components to create an initial solution and in his ideas about improving that first plan. His first effort, the selection of ‘Spin’ and ‘Sing,’ was based on memorable actions from the song rather than a systematic revisiting of it line by line. While he wanted to make his program more complete, he was at a loss to systematically go about it. Instead, he twice hoped that switching to a different interface would yield more successful results and later became distracted from the “Hokey-Pokey” goal. The efforts he made at debugging were clearly a struggle and were also unrelated to the overarching goal. (He wanted his robot to carry out all six of the instructions he had chosen without realizing that even the three instructions it did carry out did not match the song.) Finally, he contented himself in simply moving on to an exploratory program. The challenges Caleb had in focusing on and accomplishing the goal of building a complete “Hokey-Pokey” program illustrate characteristic patterns of pre-operational thought late in the stage.

Transitional.

There were 7 children (24% of the sample) who exhibited a mix of characteristics of pre-operations and concrete operations, placing them in this intermediate category. All of the children were interested in the given activity and made some systematic progress towards a “Hokey-Pokey” program. However, they encountered difficulties in fully applying systematic and empirical strategies. The abilities and general approaches seen within this group varied much more than in the other groups. It is likely that this category comprises children who were just beginning to use some concrete operational structures as well as some children who were further along in the transition.

Four of the children started with an intuitive guess and were able to make some systematic revisions before getting stuck. These four all ended up with either a program that nearly matched the “Hokey-Pokey” or a program which matched the song in length and in a few specific actions. One of these four children knew what actions she wanted to add to her program – and in the correct order – but she was unable to interpret CHERP’s ‘Forward’ and ‘Backward’ instructions as representations of the “in” and “out” movements in the song’s first three lines. Two other children began with an immediately systematic approach. One decided his program was close enough right away and declined to try improving it; the other systematically improved his program – with encouragement – before resorting to a guess-and-check strategy and ending up with an almost complete solution.

Children in the transitional group each had difficulty with at least one of the following aspects of debugging: recognizing a problem with the current solution, generating a hypothesis as to the cause, and attempting to solve the problem. In response to these challenges, children in the transitional category variously insisted that they were stuck, reverted to unsuccessful intuitive strategies, or decided that an incomplete solution was satisfactory. Sometimes children applied systematic debugging towards interesting problem variations. One child worked hard to match his program to the song's length even though the specific actions did not match, and another tried to direct the order and timing of lines of the song as the researcher sang them so that the song would match his program rather than the other way around.

There seemed to be two categories of final program completeness within the transitional group. Half of the children ended up with programs that resembled prototypes of the "Hokey-Pokey" using only two of the five actions, and the children were quite aware that their programs were incomplete. The other half ended up with programs which were only one instruction off, but they seemed to have no interest or awareness of this difference. Although these nearly complete final programs are almost on par with the final programs of children in the concrete operational group, there are some clear-cut differences in how these children arrived at their final programs. In the concrete operational group, the children with incomplete programs (three of the fourteen) had gone through two iterations of improvements and testing, while 2 of the 3 such transitional children arrived at this solution immediately and felt no need to debug. The child who did debug used a developmental mix of changes based at different points on observation and guess-and-check. Another child made an exploratory program first, then made a short and intuitive "Hokey-Pokey" attempt, and finally systematically made a "Hokey-Pokey" program, which she did not recognize as slightly incomplete. She seemed to start over with a more effective strategy from a higher developmental level each time she wanted to improve her program, rather than debugging or tweaking the first one, as was expected most children would do.

Differences also existed in how children in each group employed musical instructions, which do not map directly to any line in the song but which can be reasonably interpreted to fill in the last line and the two claps that follow it. Two of the three transitional children who used a musical instruction did so to represent that the “Hokey-Pokey” is a song, and one did so specifically to match the clapping after the last sung line. In contrast, of the five concrete operational children who used musical instructions, four did so after the dance instructions to fill in the last line of the verse and/or the claps, and one put alternating sounds after each action so that his robot would (almost) simultaneously sing and dance the “Hokey-Pokey” by itself. The uses of sounds by the concrete operational children show a higher level of systematic mapping than those in the transitional group.

There were interesting differences among children within the transitional category in terms of the unique mixes of systematic and intuitive strategies used. Overall, the children in this category all exhibited a key characteristic of transitioning between stages: switching between developmentally-based strategies even though one is more effective than another. The following case study shows the work of a transitional child, who used strategies characteristic of both pre-operations and concrete operations.

Case study 2: transitional phase.

Parker (name changed), a kindergartener almost five-and-a-half years old at the time of the study, was excited about the opportunity to play with robots. He was attending a public school and living in an urban neighborhood. According to a parent, he had average computer skills for his age, used a computer at home, and had exposure to robots through movies and books. Both parents held bachelor’s degrees. One parent held a degree in a STEM field as well as a related job, and had experience with programming and robotics.

Parker dove right into the “Hokey-Pokey” challenge. “So that (‘Forward’) could be the "in" and that ('Backward') could be the "out!" he quickly noted. He paused as he assembled the blocks on-screen. "And you shake … And I think I'm just going to do that." He programmed his robot, and, watching it, described the actions it carried out: "It went in and out, and then it shook." When asked if there was anything about his program to change, he decided, "Maybe I'll just make a really long program." After that exploration and several suggestions to try the "Hokey-Pokey" again, he picked up the tangible blocks and rebuilt his initial program, humming as he did so. "It says, 'Put your robot in, put your robot out, and shake it all about!'" He had his robot do this program, and was asked again if there was anything he wanted to change. This time he noticed something: "Oh! Turn it around! Spin it!" He sang the "Hokey-Pokey" verse to himself as he rebuilt his program on-screen and added a 'Spin' as the last action.

After he watched his robot do this program, the researcher asked whether he thought his robot had done the "Hokey-Pokey," or if the program was different from the song. He replied, "It was a little different, 'cause it didn't do the 'Hokey-Pokey.' I don't know [what to change]. 'Shake?' Maybe a double 'Shake,' 'cause we shake our bodies when we do the 'Hokey-Pokey.' So I think I'll do two 'Shakes.'" He had noticed something was off (about the timing of his program) but was not sure how to find out exactly what, so he made the best guess he could. He added the second 'Shake' after the first, uploaded the program to his robot, and watched it run. "Hey! It did it!" he exclaimed. Now he was certain that his program matched the song (even though the third action did not match the content of the line at that point) and was proud of his accomplishment.

Parker exhibited several instances of the systematic and empirical strategies typical of concrete operations. He also had moments in which he temporarily abandoned the given challenge for open explorations, relied on trial-and-error, or accepted a not-quite-complete program as successful, all behaviors more typical of children in pre-operations. Parker’s mixed work exemplifies transitional patterns of thought in programming: interest in the given challenge and in open explorations, some systematic program creation and/or debugging, and some reliance on intuitive, guess-and-check strategies or sometimes being at a loss to extend a systematic strategy through further debugging iterations. Different children showed varying proportions of concrete operational versus pre-operational strategies, indicating that these children were at different points in the transitional period.

Concrete operations, phase 1.

Fourteen children (48% of the sample) were scored as using cognitive strategies typical of concrete operations, phase 1. Their work during the “Hokey-Pokey” programming activity is markedly different from that of the pre-operational children and, to a somewhat lesser extent, from that of the transitional children. The children scored as concrete operational tended to stay on task once they started the “Hokey-Pokey,” focusing on the challenge until they arrived at a complete or nearly complete solution. The children in this developmental category used systematic approaches to create and tweak their programs and were quick to notice errors and try to fix them. To create their first programs, children in concrete operations relied more on thinking through the song step by step than an intuitive recollection of the song. Then they used the empirical results of watching their robot carry out the program while the researcher sang the “Hokey-Pokey” verse to determine whether the solution was correct. Some also “read through” their programs block by block while saying the song to themselves to make sure the program and song matched.

Children in concrete operations recognized whether an instruction was missing, unnecessary, or out of order, something children in the pre-operational category had great difficulty doing. Children in this category were rarely satisfied with a program that was not entirely complete, as opposed to children in the transitional and pre-operational categories who were more likely to call a partially complete or even completely unrelated program successful. This evolution of self-imposed standards seems to parallel the increasing orientation towards exactness in “bring[ing] a productive situation to completion” during the elementary school years (Erikson, 1998, p. 72).

Three of the fourteen concrete operational children put together a correct solution on their first try. The others revised and tested (debugged) their programs between one and five times. Of the 11 children who used such an iterative trouble-shooting process, about half (6 children) began with a long program of 4 to 7 instructions, which they then corrected by re-ordering, adding, or removing instructions as needed. The other half (5 children) began with a smaller portion of the final solution – 2 or 3 instructions – and built up to the final solution with each debugging iteration.

The following case study presents the programming work of a child who exemplified the cognitive strategies of the early concrete operational stage described in Chapter 4 and Table B4.

Case study 3: concrete operations, phase 1.

Will (name changed) was a six-and-a-half year old kindergartener at the time of the study. His programming demonstrated the typical patterns seen among the group of children in the first phase of concrete operations. He attended a public school and lived in a suburban neighborhood. His parents said that he did not use a computer at home and had no prior experience with programming or robotics. Both his parents had master’s degrees in fields outside of STEM, and both work in fields related to their degrees. Neither one had prior experience with programming or robotics.

Will decided to use the tangible programming blocks because they were “easier.” Showing his ability to plan ahead, he chose ‘Begin’ and ‘End’ blocks from the bin and explained that “The ‘End’ will be for later.’” Then he named actions as he selected their corresponding blocks: a ‘Forward,’ “Then ‘Backward.’ – I’m making the robot do the ‘Hokey-Pokey!’ – And turn yourself around?” He was not quite sure. “That’s the ‘Hokey-Pokey’ for the robot, right?” He uploaded his program (‘Begin’ ‘Forward’ ‘Backward’ ‘Spin’ ‘End’) to his robot and watched it run twice. “This time I should do a ‘Shake,’ he noticed, and added one before the ‘Spin.’ He reprogrammed the robot and watched it run. “This is funny, this robot,” he decided, and then, getting an idea, exclaimed “Oh! This will be even funnier!” He added an instruction for the robot to ‘Sing’ at the end of the four movements. After watching this program, he reconsidered: “I want to make it honk instead of music.” He exchanged the ‘Sing’ for a ‘Beep’ instruction, uploaded the new program and watched his robot carry it out.

Since he seemed happy with the actions he had in the program, the researcher asked him if his program did the whole “Hokey-Pokey.” At first he thought it did, but after watching it once more, he wanted to make two changes. “First let’s get rid of the ‘Beep,’ and I need another ‘Forward’ – Oh! That was a ‘Backward!’” He found the correct ‘Forward’ block, added it between the ‘Backward’ and the ‘Shake’ in his program, and removed the ‘Beep’ from the end. “Ok, let’s program it to do this now.” He was then satisfied that his program accomplished the “Hokey-Pokey” goal.

The strategies Will used in creating and revising a “Hokey-Pokey” program match the phase 1 concrete operational characteristics: focusing on and planning flexibly toward a goal; relying on empirical evidence; and use of logical rather than intuitive reasoning. Only his initial attempt was scored lower, transitional, because it was unclear how systematically he was thinking through the song, particularly in light of his uncertainty about what comes after the second instruction. However, once he realized that he did not remember any more of the song, Will turned to testing his program by watching the robot carry it out to find out how to complete it. He focused on the “Hokey-Pokey” goal without much redirection from the researcher, and, step by step, made his program closer and closer to the “Hokey-Pokey” song until he felt it was complete. In fact, he expressed delight throughout the activity not just at the experience of playing with a robotic toy but also at the process of getting closer and closer to a program which accomplished the given goal.

Children at each level of cognitive development varied qualitatively in their approaches to programming in terms of their goal focus and the nature of their strategizing. The following section discusses correlations between cognitive developmental level and two measures of programming achievement.

Developmental Level and Achievement

Achievement of two programming concepts, correspondence and final program completeness, was high on average but the distribution of scores across the sample warrants further investigation. On the measure of making correspondences between programming instructions and robotic actions, the overall mean was 3.86 out of 5, (*SD* = 1.66). Interscorer reliability was found to be very high (2 items; *α* = .99). A histogram of correspondence scores shows that 62% of the children achieved the highest score while the remaining 38% were distributed among the lower scores (see Figure C2).

On the measure of program completeness, or how close the child came to creating a program with all the correct instructions in the correct order, the overall sample average was 2.31 out of 4 (*SD* = 1.69). Interscorer reliability was again nearly perfect on this measure (2 items; *α* =

.99). The frequency of program completeness scores followed a bimodal distribution (see Figure C3). Forty-one percent of children's final programs were either very rough attempts at the "Hokey-Pokey" or even unrecognizable as an attempt at the "Hokey-Pokey." The other 59% of children's programs had the five basic actions solution in order, or needed to fix a single instruction. Interestingly, no children scored in between (having a final program that needed two changes to be correct).

Developmental level, as measured in this analysis, correlates remarkably strongly to both measures of achievement and explains the unusual distribution of scores seen on both measures. Although the measure for cognitive development is grounded in theory, it was coded from the children's programming activity, so a portion of the criteria of each developmental level maps onto important aspects of achievement in programming. The following results, which are unusually high for the behavioral sciences, should be considered with that limitation in mind.

On the correspondence measure, developmental level predicts 64% of the variation in correspondence scores ($F(2,26) = 23.3$, $p < .001$) (see Table C1 and Figure C4). Pre-operational children scored the lowest, needing significant to step-by-step intervention to select instructions based on their actions ($M = 1.87$ out of 5, $SD = 1.46$). Those in the transitional category scored statistically significantly higher ($M = 3.86$, $SD = 1.46$, $p < .05$); these children needed only periodic to little support, on average, with correspondence. Those in the concrete operational category ($M = 5.00$, $SD = 0.00$) also scored statistically significantly higher than the pre-operational group ($p < .001$) and needed no help in applying this skill.

On the measure of program completeness, developmental level predicts 87% of the variation in scores $F(2,26) = 90.3$, $p < .001$). Children in the pre-operational group again scored the lowest, with an average near zero out of 4 possible points ($M = 0.13$, $SD = 0.35$), meaning that their programs were generally unrecognizable as attempts at the “Hokey-Pokey.” Children in the transitional category scored statistically significantly higher (*M* = 1.86, *SD* = 1.07, *p* < .001). While this average score indicates that their programs required 2 to 3 fixes to be complete, the data show that these children actually either made a nearly complete program or a nearly unrecognizable program with no children scoring in the middle (needing 2 fixes). Children in the concrete operational group scored the highest, with at most one change needed to make their program complete (*M* = 3.79, *SD* = 0.43, *p* < .001 compared to both other groups). Table C1 and Figure C5 summarize these results.

While it was not possible to compare approaches to programming by developmental level because of the conceptual overlap in the relevant measures, it is possible to examine the programming achievement of children grouped by developmental level of each cognitive development sub-score. Indeed, differences in achievement are seen in each component of the cognitive developmental metric: goal orientation, characteristics of the initial solution, and characteristics of debugging. The differences in achievement based on overall developmental level versus component scores exist because each child might have component scores in multiple developmental categories which combined to a single overall category. These comparisons are as close a measurement of programming approach, separate from cognitive development, as was feasible.

Children’s level of focus on the assigned goal impacted achievement differently from overall developmental level (*F*(2,26) = 35.2, *p* < .001 for correspondence and *F*(2,26) = 168.1, *p* < .001 for program completeness). (See Table C2 and Figures C6 and C7.) Children whose focus (or lack thereof) on the “Hokey-Pokey” was assessed as pre-operational or transitional scored statistically significantly lower on both measures of programming achievement than those assessed as concrete operational on goal orientation ($p < .001$ for both comparisons). In other words, children who did not attempt the “Hokey-Pokey” challenge, who were easily distracted from it, or who moved on to other goals after becoming stuck needed help matching instructions to “Hokey-Pokey” actions – anywhere from minimal support to step-by-step instructions. They also ended up with programs that did not resemble the “Hokey-Pokey.” On the other hand, children who remained interested in the “Hokey-Pokey” despite the need to debug their programs needed no help matching instructions to actions in the song and ended up with complete or nearly complete programs.

Differences in achievement were also seen based on the type of strategies children used in their initial solution to the “Hokey-Pokey” challenge ($F(2,26) = 13.4$, $p < .001$ for correspondence and $F(2,26) = 58.5$, $p < .001$ for program completeness; see Table C3 and Figures C8 and C9). Children whose initial solution strategy was characteristically pre-operational differed in both achievement scores compared to those with transitional or concrete operational strategies ($p < .01$ for correspondence and $p < .001$ for program completeness). Scores for program completeness also differed statistically significantly between children with transitional first solutions compared to children with concrete operational initial strategies ($p < .05$). Children whose first attempt at the “Hokey-Pokey” was based on intuitive strategies (if they made an attempt at all) needed, on average, significant help with selecting instructions that matched the song and their final programs were extremely rough prototypes at best – including perhaps a relevant instruction or two. In contrast, children whose initial attempt relied at least partially on systematic strategies were able to select instructions that matched the song with little to no help and most of these children ended up with four or five of the five necessary actions for a complete program. Of note, children in the transitional category for initial strategy either needed no help or significant help with correspondence, as opposed to children in the lower category whose scores covered the full range of help needed but tended towards significant help, or the children in the highest category, who needed no help at all.

Finally, statistically significant differences in mean scores on both programming achievement measures were found based on children's strategies used in debugging ($F(2,26) = 15.9$, $p < .001$ for correspondence $F(2,26) = 52.5$, $p < .001$ for program completeness; see Table C4 and Figures C10 and C11). For correspondence, this result was driven by differences between children whose debugging was scored as pre-operational versus concrete operational ($p < .001$). For program completeness, there were statistically significant differences among all groups ($p < .001$ comparing pre-operational and transitional to concrete-operational and $p < .05$ for pre-operational to transitional).

Although it was not an original goal of this analysis to examine, there is the interesting question of whether developmental differences also exist in children's baseline sequencing scores or changes in sequencing scores from before to after the robotics sessions. A definite relationship in the larger population has not yet been established from the present data and basic analysis, perhaps due to the small sample size, (see Table C5); however, some patterns are observed (Kazakoff & Bers, 2011). Children in late pre-operations scored lower on average than children in early concrete operations on both pre- and post-intervention assessments and made the smallest improvements, on average, between the assessments. Children in the transitional phase scored lowest and had the widest variation in scores on the pre-assessment; they made the largest gains at the post-assessment. This may have been due to a developmental readiness to acquire new cognitive skills. Children in concrete operations began with the highest scores and made middling gains. Presumably, they would have made larger gains if many of them were not already achieving the highest possible score. Additional studies a larger sample and more complex analysis are needed to establish what relationship might exist between cognitive development and sequencing ability.

Individual Differences

Children's demographics.

Data was collected on children's age at the start of the study, gender, grade in school, and whether the child lived an in urban or suburban area. As a second cognitive variable that might influence programming, children's baseline and delta sequencing scores were also analyzed to see if either predicted programming achievement. Age was statistically significantly and positively correlated with scores on the completeness of children's final programs when tested as a single factor. However, age makes no contribution to predicting program completeness after cognitive developmental level has been accounted for. No other demographic or cognitive baseline factors correlated with either measure of programming achievement. (See Table C6 for relationships with cognitive baseline measures and Tables C6 and C7 for relationships with children and parent background measures).

Children's computer, programming, and robotics experience.

Information on children's prior experience with computers in general and robots and programming in particular was also collected. Of the 28 children whose parents completed the background information survey, 75% played games on the computer at home, and a few children also used a computer to make art or music, for communication, to look up information, for word processing, or to use educational software. Seven children (25%) did not use a computer at home. Half the children were perceived as computer "beginners" for their age group by their parent. Nearly another half were perceived as "average" for their age group and one child was perceived as an expert for the age group. Over half of the children had no prior experience with robots as reported by parents. Ten percent of children had experience with non-programmable robots, such as watching a movie with a robot in it or playing with a pre-programmed toy. About a third of the children had experience with robotics classes, workshops, museum activities, and robotics kits, which are assumed to involve programmable robots. Interestingly, no parents believed their children had any programming experience.

Home computer use negatively correlated with scores on measures of programming achievement ($t(20) = 3.68$, $p = .001$ for correspondence and $t(26) = 4.28$, $p < .001$ for program completeness); however this result is accounted for by the fact that all of the children who did *not* use computers at home are in the concrete operational cognitive developmental category. Home computer use lost its statistically significant correlation to achievement after controlling for cognitive developmental level. While children perceived as having “average” computer skills for their age did slightly better than those perceived as “beginners” on programming achievement scores, the difference was not statistically significant, and is also accounted for by the disproportionate number of beginner concrete operational children. Only one child was perceived as having “expert” computer skills so comparisons with that category were not useful. No other categories of children’s prior computer, programming, or robotics experience correlated with programming achievement measures.

Parental education and STEM background.

Nearly half of the families had at least one parent with a doctoral degree; nearly another half of families had a master’s degree as the highest level of education by either parent. Over half, 59%, of children had at least one parent whose latest degree was STEM-related, and 53% of families had at least one parent currently working in a STEM-related field. Almost a third of children had at least one parent with prior exposure to robotics, and over two-thirds of the children had at least one parent with prior experience programming.

No significant relationships were found between either measure of children's programming achievement and parental educational, occupational, or experiential factors. The lack of statistically significant relationships may be due to the study's low *N* or to the specific measures used rather than an absence of relationship between the factors. Children with at least one parent whose most recent degree was STEM-related or at least one parent who had prior experience with robotics did modestly better (less than half a point) on both programming achievement measures than children whose parents did not. Children with at least one parent with programming experience scored about half a point higher on program completeness. Larger studies with more precise measures would be needed to rule out this study's finding that these patterns are seen by chance. No patterns were found based on the highest level of parental education, a rough proxy for socio-economic status, but the sample included hardly any variation on the metric: 90% of the families had at least one parent with at least a master's or doctoral degree.

Novice Difficulties

Children in the study exhibited different difficulties in completing the programming challenge based on their assessed level of cognitive development. Children in the first phase of concrete operations had few points of difficulty learning the components of the CHERP programming environment, how to program a robot with an arbitrary CHERP program, and how to use CHERP to solve the "Hokey-Pokey" challenge. Only 3 of these 14 children ended the activity with an error in their programs: an extra action, a missing action, and confusion between 'Forwards' and 'Backwards.' Children in this stage tested and systematically revised their programs with relative ease. By contrast, children in the late pre-operations and transitional phases regularly exhibited several types of errors or difficulties. Those who attempted the “Hokey-Pokey” frequently:

- (By the definition of the developmental categories used in this analysis), relied exclusively or partially on unsystematic strategies: guess-and-check or thinking of as many necessary instructions as possible without checking the song line by line;
- Noticed errors but declined to try to improve that aspect of the program;
- Claimed that programs matched the “Hokey-Pokey” song when this was not the case;
- Felt that a program successfully matched the song if the robot’s movements and the song had the same duration;
- Knew what action they wanted the robot to do, but were not sure which CHERP instruction corresponded or even whether CHERP had such an instruction;
- Moved on to open-ended explorations rather than improving their program; or
- Needed encouragement to keep working on the “Hokey-Pokey” challenge.

It seems that children in late pre-operations and the transition period towards concrete operations experienced some similar difficulties as compared to older novice programmers (as discussed in Chapter 3), but that the difficulties were alleviated for children in early concrete operations, perhaps due to a good match between children’s cognitive abilities and the design of the programming language. Older novices experience frustrations in learning exactly how a language’s instructions and syntax work, have difficulties flexibly combining instructions to solve a given problem, often fail to test their programs or trace through code to locate errors, have difficulty doing so if they try, and may make haphazard changes rather than systematic changes whose implications have been thoroughly considered. These match some of the challenges faced by children in pre-operations and the transitional period, with the notable exception of the children's tendency to claim success for actually ineffective solutions.

There are several implications of the patterns of difficulties in the TangibleK lab study compared with typical older novice challenges. First, as the younger two developmental categories seemed to have similar issues as older novices, these issues may pertain to common characteristics of human cognition and problem-solving styles that people use without specific instruction to do otherwise. Such challenges may be tempered or altered by unique developmental characteristics across ages. Secondly, since concrete operational children did not share these challenges during the "Hokey-Pokey" activity, difficulties novices experience may be a function not only of the strategies the novice relies on but also of the match or fit between the person and the programming language. This concept underlies research on more intuitive and simplified introductory programming languages as well as the development of languages which fill gaps between introductory languages and full languages intended for professional use.

It should be noted, however, that while most studies with older programming students involve a semester or more of programming, the analysis of preschoolers and kindergarteners in this study is only of a single activity following less than an hour of work with CHERP. Further research and analysis is needed to discover whether children in early concrete operations experience the pattern of challenges seen in the other groups with the introduction of control flow structures, and how children's main points of difficulty evolve over a few months or a year of experience.

The quantitative, qualitative, and case study results presented above paint a picture of many of the factors that contribute to various degrees to preschool and kindergarten children's success in a programming task.

Chapter 7: Discussion

The findings of this study range from the relationship between level of cognitive development and programming for four- to six-year-olds, the impact of individual and background factors on this relationship, and a comparison of the challenges faced by children in this study with those experienced by older novice programmers. Clear patterns were seen in children's programming strategies and achievement based on their cognitive developmental levels, but not on most other variables.

It was found that children in the latter half of pre-operations tended to explore the possibilities and boundaries of CHERP rather than engaging in the specific given challenge. The intuitive problem-solving strategies characteristic of this group made the "Hokey-Pokey" goal unattainable. They were more at ease ignoring that goal and following self-directed explorations of CHERP.

Children in the first phase of concrete operations responded quite differently to the programming task: they were enthusiastic about generating an iteratively more precise solution. Unlike their pre-operational peers, they relied on empirical feedback and systematic logic to reach the goal. When they took on self-defined challenges, their goals were contextualized; they wanted to use CHERP to accomplish an imagined scenario rather than simply understand more about how CHERP works, as the pre-operational children did.

This study also included children who fell in the transitional phase between late pre-operations and early concrete operations. Although the results looked different from child to child, the transitional group was in general interested in solving the challenge, like the concrete operational group, and made some similar if inconsistent systematic or empirically-based progress, but, like the pre-operational group, they became stuck, and often moved on to open-ended explorations.

While further studies are needed to confirm a relationship between programming outcomes and an independent measure of cognitive development, the framework sheds light on the wide range of programming abilities seen during the study. By contrast, other measures of cognitive ability (sequencing), demographics, parental background, and prior experience by the child did not correlate statistically significantly with measures of programming achievement, at least not after taking into account developmental level or skewed subsamples.

Examination of the programming activities of children not included in the main analysis uncovered some interesting phenomena not systematically captured by the defined variables. These children had been excluded due to distraction or shyness, having taken on a structured challenge other than the "Hokey-Pokey," or for having shown unexpected increases in achievement following minimal support. For instance, three children displayed reasoning characteristic of two different stages before and after the systematic line-by-line strategy was introduced. During their initial attempts, these children had similar difficulties as children in the pre-operational category: distractibility from the activity, difficulty matching CHERP action instructions to "Hokey-Pokey" lines, and no overall, systematic strategy for creating and debugging a solution. However, once the researcher began to help them complete the activity by modeling how to find the matching instruction line by line, these children enthusiastically took the initiative to finish the rest of the program almost entirely on their own. Although the data for these children did not meet the requirements for inclusion in the study, their initial struggles and surprisingly rapid appropriation of the line-by-line strategy might suggest that these children were at just the right transitional point in their cognitive development to learn such a strategy.

An alternative interpretation of these children's varied use of strategies stems from the children's personalities. Two of these three children were quite shy – much more so than any other children in the study. By the time the researcher began supporting the children's programming, (or perhaps, because the researcher did so) they started to feel more comfortable in the study setting and with problem-solving under observation. This might have led them to think more freely, take more chances, and willingly test out potentially incomplete or inaccurate programs. The third child was socially at ease, but his distractibility seemed to increase proportionally with the level of challenge a given task presented. For instance, although he barely focused on the third programming challenge, he concentrated immediately, energetically, and without redirection on the more concrete post-intervention assessments that immediately followed.

Whether the patterns exhibited by these three children can be attributed to the interaction of the activity with core personality traits or to developmental readiness, either in terms of Vygotsky's zone of proximal development for learning or the neo-Piagetian concept of micro-development, the observations raise questions about whether and how to expand the analysis presented in this thesis to more fully capture the cognitive processes and other factors involved in programming. Other personality factors which appeared to have a possible role include: impulsivity, compliance, perfectionism, and self-consciousness. It also would have been interesting and pertinent to examine the impact of cognitive styles and personality traits on children's achievement and the interaction between these individual differences and developmental factors.

The most compelling result of this analysis is the categorization by cognitive developmental level of the cognitive strategies children use when programming robots, the skills they are able to succeed on, and the extent of their achievement on each skill. Due to various limitations of the study, including the analysis of a single programming activity per child, follow-up studies would likely provide insights into how to revise the framework. The degree of variation within the transitional group shows one area where refinement is needed. The transitional group appears to be made up of two possible sub-groups, with one group developmentally ahead of the other but both groups clearly between the pre-operational and concrete operational levels. The evidence for this stems from the fact that part of the transitional group scored just slightly higher than the pre-operational group on programming achievement and the other part scored just under the concrete-operational group. While follow-up studies should confirm or disprove and expand on the initial findings presented here, the cognitive developmental framework for programming can nonetheless provide specific points of departure for future design or re-design of young children's programming tools and the curricula and learning goals accompanying them.

Implications for Programming in Early Childhood

Based on the evidence that children program strikingly differently depending on their level of cognitive development, what supports for learning computer programming can be given to young children while respecting their developmental and other individual characteristics? This section discusses differentiated learning expectations, curricula, and programming and robotics tools for young children.

Learning expectations and curricula.

The most significant implications of the observed differences in cognition and programming outcomes are those regarding learning expectations, curricula, and pedagogy. Educational goals and methods must be framed according to the recognition that the motivation and abilities of children using pre-operational cognitive structures are quite distinct from those of children using concrete operational structures. At the very least, children in pre-operations may need more time to learn the basic functionality of a programming environment than older children. More significantly, the typical goals of their programming are quite different from those of concrete operational children, and the aspects of programming that challenge them differ dramatically as well.

Children in pre-operations pursue goals that allow them to learn the boundaries and capabilities of the programming and robotics materials: What is the longest program I can make? What happens if I use all the purple instructions? Can I make a pattern of repeating instructions? Will the robot do the same thing if I make the program on the screen instead of with the wooden blocks? What is the difference between 'Forward' and 'Backward?' Curricula for programming and robotics in preschool and early kindergarten should foster such explorations. Activities and teacher support should provide pathways toward explorations beyond those which children generate on their own and allow opportunities for students to articulate and discuss what they observed.

Noticing results and using them to generate ideas for follow-up explorations is a key skill for teachers of this age group to model and for children to try out themselves. While it must be understood that children in this group are unlikely to create or revise programs systematically or with adult-like logic, if teachers model and support such strategies regularly – but without superseding children's own explorations – then children will not only have confidence in their own ideas but will also have these strategies at their disposal when they reach the transition to concrete operations. Especially as children are just beginning to develop logical and systematic strategies, exposure to a variety of ideas can help them fully explore their growing abilities. In sum, the curriculum for a group of late-stage pre-operational children should value and support children’s explorations of the programming environment. The cognitive goals for this group should be to build observation and articulation of programming outcomes and to use those observations, with support, to refine the program or generate a new goal.

While the children in pre-operations tend to focus on explorations, children in concrete operations are more likely to have a contextualized goal which can be achieved by designing a program for their robot: ‘Can I make my robot push this pile of bricks off the table?’ ‘Can I make my robot dance all the ‘Hokey-Pokey’ verses?’ ‘How close can I get my robot to the wall without crashing?’ While these children still benefit from open-ended explorations, they are also capable of and enjoy taking on more structured, problem-solving intensive challenges. Whether their goals are self-selected or given by a teacher, it is important for children in early concrete operations to articulate and hear from one another about the logic and strategies used in creating and debugging their programs. A curriculum for this group would include many opportunities to solve challenges of varying difficulty within each category of instructions (actions; actions and control flow structures with numeric parameters; and control flow structures with sensor parameters). Introduction of each new control flow structure and parameter type should be accompanied by thorough exploration of how these instructions work, similar to the exploration of action-only programs by pre-operational children. Curricula for this developmental group should push them to use their budding skills in systematic logic to solve increasingly complex challenges.

The specific types of challenges, activities, or explorations appropriate for children in the later portion of pre-operations differ from those for children in the early part of concrete operations, but the pedagogy remains the same. Curricula and learning goals for either age group should sustain the initial excitement children feel when they first program robots, encourage children to follow up on their ideas, and support children in honing and extending their thinking within that child-focused context.

Programming and robotics tools.

Developmentally appropriate programming interfaces for late pre-operational versus early concrete operational children may not need to be as differentiated as learning expectations and curricula for the two groups, at least within the age range of the TangibleK study. Both late pre-operational and early concrete operational thinkers benefit from a programming interface that does not require extremely fine motor skills or eye-hand coordination. Although some children (across developmental categories) had difficulty using a mouse or connecting on-screen blocks, future iterations of the software using touch-screens and improving the ‘snap-together’ behavior of graphical blocks may address these challenges.

On the other hand, one could reasonably revisit the question of whether the hybrid interface benefits all children in the developmental range covered by the study. Some children persevered in using the on-screen interface, in spite of great difficulty in manipulating the mouse, because computer time was seen as a rare and precious opportunity. This caused them to spend more time physically building programs than thinking about what instructions to include or how to debug a program in progress. Other TangibleK work has highlighted that certain classroom management decisions cause all children to use one interface or the other, either purposefully or unintentionally, with positive outcomes for management of materials and adult support. Are there circumstances when individual children (rather than the class as a whole) would benefit from having access to only one interface or the other? Is it worth the time necessary to help children independently evaluate each interface and choose the one that best matches their goals and abilities? Do children settle on an interface that they can use reasonably efficiently if access to

CHERP is a common rather than rare experience? Future studies, including longitudinal research are needed to address these questions.

In the TangibleK study, the relatively small, high-level set of instructions seemed compelling to the full developmental range of children. By some age or by some level of experience, children will be ready for more customizable robotic actions, but this particular study cannot address that point. Studies following children over longer periods of experience with CHERP could demonstrate at what point children feel limited by the instruction set and are cognitively ready – both from development and experience – to construct their own relatively lower-level instructions, such as those in LEGO®'s WeDo™ programming language.

Another interesting question deals with children who do not debug with CHERP, or who have trouble doing so: how would their programming approaches and outcomes be different if the programming interface included a specific tool to support debugging, perhaps by tracking through the program as the robot ran it? It is possible that this would support children who are interested in improving their programs but who do not have the cognitive requisites to simultaneously monitor the robot's actions and compare them to the program. On the other hand, such a mechanism might prevent or de-motivate children from taking on this cognitive task as they are able. Research is needed to shed light on what children do with such a feature and determine the potential benefits or drawbacks.

Finally, it is important that the programming and robotics materials function consistently. This was not always the case with early versions of CHERP and with the LEGO® sensors, which led to unnecessary confusion in understanding, for instance, how control flow structures and sensor parameters influenced what the robot did. While this may seem like a problem with a clear solution, the functionality of robotic components raises an interesting dilemma.

Programming robots rather than animations necessarily involves more factors than just the program: real-world, physical factors like gravity, friction, sturdiness, and battery power. To completely obscure this reality deprives the child of investigating authentic problems in the robotics domain. The ideal characteristics of the robotics materials may depend on how heavily weighted the learning goals are towards robotics versus programming. However, it is too much of a cognitive load for preschoolers or kindergarteners to simultaneously debug a finicky robotics system as well as a program for its behaviors. Whether they are focused on one aspect of the relationship between their program and the robot's behavior or are limited in working memory capacity for systematically accounting for multiple factors, this age group benefits from a fairly cleanly working set of introductory robotics materials so that they can explore the logic and affordances of the programming instructions.

Overall, the CHERP programming language and hybrid interface worked well as an introduction to programming for most of the children in the one-on-one study. Recommendations for future research on programming interfaces, already touched on throughout this section, include exploring how children's use of CHERP changes with prolonged use and age to determine at what point children are ready for a more complex instruction set, examining how that readiness is driven by development and experience, and exploring what the next level of complexity might look like in terms of the instruction set and possibly the interface as well.

Limitations of the Study

Every piece of research or analysis has its limitations; the TangibleK one-on-one study and the analysis contained in this thesis are no exception. Following are the major points which could or should be revised in future research.

To begin with, this study had a relatively small sample. Although the correlation between developmental level and achievement was significant and strong, statistical significance was all but unreasonable to expect in the more detailed analyses, which took into account multiple predictor variables and thus compared groups of less than 12 children. Analysis of a larger sample is necessary to paint a thorough picture of the primary cognitive developmental and other factors contributing to early childhood programming experiences and outcomes.

A pervasive limitation stemmed from this thesis relying on data that the TangibleK one-on-one study was not specifically set up to measure and which thus had to be coded from existing video. It was not possible to go back in time and take independent measures of cognitive developmental level, key cognitive styles, or ask children follow-up questions that would have been relevant. Development was measured from existing video data because it was possible to map cognitive developmental traits of Feldman's revised Piagetian stages (Feldman, 2004) to the highly reasoning-oriented activity of programming. The consequence was that the measures of cognitive development, approach to programming, and programming achievement were not fully independent, and the high correlations found among these variables must be validated with future studies using completely separate measures. Furthermore, using a single, unvalidated activity to measure a trait assumed to be consistent over time or contexts leaves open the possibility that the child actually exhibited the observed characteristics only during that activity. While the analysis in this thesis assumes, based on developmental theory, that this is not the case, it would be better to use a measure that does not require such an assumption.

Perhaps most significant after the lack of an independent cognitive development measure, the original study piloted a large number of measures created by the research team, and some need revision to better address the concepts that they were intended to document. Parents' and children's background information that was collected did not provide detailed enough information to realistically connect these factors with children's achievement during the study. Perhaps the variables regarding parents' having STEM-related degrees or occupations could have been replaced or supplemented with a measure of systematic problem-solving in everyday home life or with measures of technological literacy and fluency – knowledge and comfort with using a variety of computer-based technologies. Some of the measures, such as the child's prior experience with robots, could have asked more directly for the information of interest so as to avoid ambiguous answers. The measures of children's prior experience could also have covered more contexts (classroom and after-school settings) and a wider genre of computational devices (including smartphones, video games, electronic toys, etc.). Furthermore, when data on the same measure was collected from both parents and children, family members sometimes gave conflicting information, and a few individual parents even gave self-conflicting information. This unfortunate finding might have been due to inadequately defined terms such as 'robotics kit,' 'programming,' or 'average,' which could be interpreted in many different ways. More narrowly focused studies with revised variables and measures should follow up on this pilot research.

For this thesis, the same person conducted the study sessions, coded the data, and analyzed it. Interscorer reliability testing was completed with a second scorer (also knowledgeable of the study and the goals of this analysis) only for measures of programming achievement. Future work would ideally include blind scoring and interscorer reliability tests of the developmental measures as well.

Finally, the original study provided data for three separate programming activities, but due to time constraints, only one was analyzed in the present work. It would have been very interesting to compare children's approach and success in the "Hokey-Pokey" activity to the later activities, when the use of control flow structures disrupted the linear flow of programs and the one-to-one relationship between instructions and actions performed by the robot. Further developmental differences, such as multiple classifications, could have been analyzed through these activities, since the control flow instructions represent a qualitatively different category of instruction than the action instructions. (A hint of developmental differences in understanding different types of programming instructions is highlighted by the fact that one concrete operational child remarked with pleasure at his own understanding that the program 'Begin' 'Forward' 'End' "does one thing even though it has three blocks," while the child figuring in the pre-operational case study was confused why the 'Begin' and 'End' instructions did not result in visible robotic actions.) Alternatively, it would have been interesting to observe children's approaches and achievement over a series of activities similar in cognitive requirements to the "Hokey-Pokey." This would extend the current picture of children's early learning of CHERP and of programming strategies. It would also ensure that children were assessed once they were more familiar with CHERP and with the format of problem-solving under one-on-one observation.

Future Directions

The analysis of cognitive and individual factors in children's programming during the TangibleK one-on-one study led to many interesting observations and results. Nonetheless, the research was a pilot in many respects. Furthermore, the analysis presented in this thesis relied on variables that the original study was not specifically designed to measure. Were a study carried out to address precisely the research questions presented in this thesis, the shortcomings discussed above would certainly need to be revised. To summarize, independent and validated (or at least more targeted) measures for each variable must be included. If possible, a larger sample should be drawn from a more socio-economically diverse pool. Analysis should also be conducted on a larger set of programming activities than was possible in the scope of this thesis by including several programming tasks per child, preferably after a thorough introduction to CHERP. Finally, future studies should be implemented with a polished version of CHERP whose graphical interface is as physically easy to manipulate as the tangible blocks. This would eliminate the question of how much the finicky interface detracted from children's potential reasoning and programming outcomes.

Should such a revised study uphold the basic findings presented in this thesis, follow-up research could test the suggested implications of the current findings on learning expectations, curricula, and robotics and programming materials in the section above. Classroom-based work is needed to validate or refine the proposed differentiated learning expectations and curricula for children at each level of cognitive development and provide a deeper understanding of how curricula shift to match and support children's new thinking as they transition between stages. Longitudinal studies can shed light on how children's programming evolves over the course of long-term use of CHERP. Studies of slightly older children and of children who are experienced CHERP programmers could subsequently inform the design of a follow-up version of CHERP, specifically for children with more sophisticated cognitive developmental structures at their disposal. Future work can also continue examining issues relating to children's use of the hybrid interface and explore the impact of alternative features for CHERP and the robotics materials.

Chapter 8: Conclusions

Children's use of new technologies from smartphone apps to electronic learning toys is prevalent as never before, and this trend is unlikely to change course. What will evolve is how parents, educators, and society distinguish the enriching designs and uses of technology from the detrimental. Within the context of constructivist development and constructionist learning, powerful yet age-appropriate new computational materials for building and programming robots make the digital world children's own. This thesis has addressed some of the major cognitive developmental issues surrounding how children between the ages of four and six use CHERP, a graphical-tangible hybrid programming language for programming robotic vehicles.

To illustrate and make sense of the striking differences seen among children's programming goals, approaches, and outcomes in the one-on-one TangibleK study, a descriptive framework of children's cognitive developmental stages was created by extrapolating programming behaviors from characteristics of phase 2 of pre-operations, phase 1 of concrete operations, and the transitional period in between, as found in the literature on Piaget's theory and Feldman's revisions of it (Feldman, 2004). The framework appears to be a successful first attempt at measuring cognitive development from a non-Piagetian yet reasoning-rich task. Children within each level of the cognitive development framework were consistent, overall, in their focus on the task, the nature of their reasoning, and in their correspondence and sequencing achievement. Furthermore, their work was increasingly systematic and effective from one developmental group to the next.

It was found, in support of the developmental framework's validity, that while age and developmental level correlate statistically significantly, developmental level does a far better job of predicting achievement scores than age does. Furthermore, no other cognitive, demographic, experiential, or parental background factors statistically significantly predicted differences in achievement beyond the contribution made by developmental level, suggesting that this variable plays the predominant role in the early childhood programming activities explored in this study. It was expected that parental background in STEM fields and children's prior experience with computers, robotics, and/or programming would have had some impact on achievement.

However, such impacts may have been overshadowed by the correlation with developmental level. Alternatively, the measures for these variables may not have fully captured the intended concepts.

Based on this analysis, comparisons were made between novice programmers of different ages. It appears that novices in early childhood through adulthood face many of the same high-level problem-solving challenges but that these may be alleviated if the complexity of the programming language matches the cognitive characteristics, developmental or otherwise, of the programmer. Recommendations were then made regarding appropriate learning goals, curricula, and programming tools, each differentiated for different stages of early cognitive development. These recommendations focused on explorations of CHERP itself in late pre-operations and contextualized goals in early concrete operations. It was noted that CHERP seemed compelling to children across the developmental range of the study, but that at some later point, whether after a long enough exposure to CHERP or following a particular developmental milestone, children would be ready to use lower-level programming instructions and create their own units of robotic actions. It is hoped that, through effective teaching and learning contexts, technologies like CHERP can better support children's exploration of the digital realm.

Programming robots' behaviors is an engaging and cognitively rich activity. It also has relevance for modern education in children's comprehending and exerting ownership over the now ubiquitous computer and acquiring digital literacies and fluencies for the 21st century. Programming and robotics materials do more than introduce technological content domains; they provide versatile digital tools for construction, modern extensions of traditional materials for children-driven exploration and expression. However, children today frequently experience their most empowering and creative uses of technology in the time and spaces outside of the classroom (New Media Consortium, 2005). This leads to inequity in access to technology, erratic acquisition of the knowledge and skills that are attainable through technology, and an absence of adult guidance on ethical issues that arise (Jenkins et al., 2009). There is debate, though, as to how technology should figure into state and federal curriculum frameworks when there is growing pressure to focus on standardized testing and traditional basics like math and literacy. Work like the TangibleK Robotics Project shows how schools may be able to incorporate rich technologies into existing curricula.

Many children in the US today have access to technology from a surprisingly young age whether or not the adults in their lives intentionally plan it. The National Association for the Education of Young Children emphasizes that the use of technology must be balanced with other activities and be grounded in knowledge of the children, the technology, clear educational and developmental goals, and other known best practices, as is the case when non-technological materials are used (NAEYC, 2011). Some parents have been hesitant for their child to engage in programming and robotics, citing a preference to limit their child's 'screen-time.' This well-founded concern highlights the need for a differentiation of screen types in the contemporary media vocabulary, as there is a vast and crucial difference in the cognitive activity fostered by screen-time as consumers (i.e. video games and television) compared to screen time as producers (i.e. programming and engineering design).

In using tools like CHERP, children spend their time only partly in front of a screen. They also move physically and cognitively between building a robot, planning out its actions, constructing programs (from wooden blocks or on the computer screen), and iteratively observing, analyzing, and altering the robot and its program according to initial goals and subsequent discoveries. Because the tangible programs and robots exist off-screen, children are drawn socially to investigate the work of their peers, to collaborate, and to negotiate over materials. The artifacts – the robot and the program – serve as points of discussion and reminders of the activity content even after the computer has been shut down. In this rich process of creation that bridges the physical and digital worlds, children actively engage in problem-solving, discovery-based learning, and uncovering powerful ideas from computer science and robotics. These skills, both general and domain-specific, are crucial in today’s world, where the creation and sharing of digital content are empowering means of expression and communication.

The International Technology Education Association has modeled a comprehensive framework of key technology knowledge and skills (ITEA, 2007), and several technologically rich media and literacy frameworks and theories for education now exist as well (e.g. Jenkins et al., 2009). As the new uses of computers and the internet become more familiar and as ideas about how to include them in education are clarified and tested, it can be hoped that teachers, administrators, schools, and states will adopt such frameworks and incorporate technology through investigation-based pedagogies. In the meanwhile, further research into cognitive and other aspects of young children’s interactions with technology can paint a more detailed picture of what developmentally appropriate technologies can afford throughout childhood as well as reasonable learning expectations and curricula to accompany them.

References

Ackermann, E. (1991). The "agency" model of transactions: Toward an understanding of children's theory of control. In I. Harel & S. Papert (Eds.) *Constructionism: Part 4. Cybernetics and construction* (pp. 367-379). Norwood, NJ: Ablex.

Andre, T., Whigham, M., Hendrickson, A., & Chambers, S. (1997, March). *Science and mathematics versus other school subject areas: Pupil attitudes versus parent attitudes.* Paper presented at the Annual Meeting for the National Association for Research in Science Teaching, Chicago, IL. Paper retrieved from http://www.eric.ed.gov/PDFS/ED416092.pdf

Arduino (2011). *Arduino*. Retrieved from http://www.arduino.cc/

Barab, S., & Squire, K. (2004). Design-based research: Putting a stake in the ground. *Journal of the Learning Sciences*, *13*(1), 1-14. doi:10.1207/s15327809jls1301_1

Baron-Cohen, S., Leslie, A. M., & Frith, U. (1986). Mechanical, behavioral, and intentional understanding of picture stories in autistic children. *British Journal of Developmental Psychology, 4*, 113-125.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48-54. doi:10.1145/1929887.1929905

Bergen, D. (2001). Learning in the robotic world: Active or reactive? *Childhood Education. 77*(4), 249-250.

Bers, M. (2008). *Blocks to robots: Learning with technology in the early childhood classroom.* New York, NY: Teachers College.

Bers, M. (2010). The TangibleK Robotics Program: Applied computational thinking for young children. *Early Childhood Research & Practice, 12*(2). Retrieved from http://ecrp.uiuc.edu/v12n2/bers.html

Bers, M. U., & Horn, M. S. (2010). Tangible programming in early childhood: Revisiting developmental assumptions through new technologies. In I. R. Berson & M. J. Berson (Eds.) *High-tech tots* (pp. 49-70). Greenwich, CT: Information Age.

Bishop-Clark, C. (1995). Cognitive style, personality, and computer programming. *Computers in Human Behavior, 11*(2), 241-260. doi:10.1016/0747-5632(94)00034-F

boyd, d. (2008, January 10). Technology and the world of consumption. [Web log post]. Retrieved from http://www.zephoria.org/thoughts/archives/2008/01/10/technology_and.html

Brown, A. L. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *Journal of the Learning Sciences*, 2(2), 141-178. doi:10.1207/s15327809jls0202_2

Buckingham, D. (2007). Digital media literacies: Rethinking media education in the age of the Internet. *Research in Comparative and International Education, 2*(1), 43-55. doi:10.2304/rcie.2007.2.1.43

Carnegie Mellon University (2011). *What is Alice?* Retrieved from:

http://www.alice.org/index.php?page=what_is_alice/what_is_alice

Case, R. (1984). The process of stage transition: A neo-Piagetian view. In R. Sternberg (Ed.) *Mechanisms of cognitive development* (pp. 19-44). San Francisco, CA: Freeman.

Center for Computational Thinking: Carnegie Mellon. (2011). *What is computational thinking?* Retrieved from http://www.cs.cmu.edu/~CompThink/

Clements, D. H. (1986). Effects of Logo and CAI environments on cognition and creativity. *Journal of Educational Psychology, 78*(4), 309-318. doi:10.1037/0022-0663.78.4.309

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*(6), 1051-1058. doi:10.1037/0022-0663.76.6.1051

Clements, D. H., & Meredith, J. S. (1992). *Research on Logo: Effects and efficacy*. Retrieved from http://el.media.mit.edu/logo-foundation/pubs/papers/research_logo.html

Cooper, L. Z. (2005). Developmentally appropriate digital environments for young children. *Library Trends, 54*(2), 286-302. doi:10.1353/lib.2006.0014

Desforges, C., & Abouchaar, A. (2003). The impact of parental involvement, parental support, and family education on pupil achievement and adjustment: A literature review (Research Report RR433). Retrieved from UK Department of Education website https://www.education.gov.uk/publications/standard/publicationdetail/Page1/RR433

DevTech Research Group (2010). *C.H.E.R.P.*. Retrieved from http://ase.tufts.edu/DevTech/tangiblek/research/cherp.asp

Engle, R. W., Tuholski, S. W., Laughlin, J. E., & Conway, A. R. A. (1999). Working memory, short-term memory, and general fluid intelligence: A latent-variable approach. *Journal of Experimental Psychology, 128*(3), 309-331. doi:10.1016/j.edurev.2006.08.005

Erikson, E. H. (1998). Eight stages of man. In C. L. Cooper & L. A. Pervin (Eds.) *Personality: Critical concepts in psychology* (pp. 67-77). London, UK: Routledge.

Farr, W., Yuill, N., & Raffle, H. (2010). Social benefits of a tangible user interface for children with autistic spectrum conditions. *Autism, 14*(3), 237-252. doi:10.1177/1362361310363280

Feldman, D. H. (2004). Piaget's stages: The unfinished symphony of cognitive development. *New Ideas in Psychology, 22,* 175-231. doi:10.1016/j.newideapsych.2004.11.005

Fernaeus, Y., & Tholander, J. (2006). Finding design qualities in a tangible programming space. In R. Grinter, T. Rodden, P. Aoki, E. Cutrell, R. Jeffries & G. Olsen (Eds.) *Proceedings from SIGCHI '06: Conference on Human Factors in Computing Systems* (pp. 447-456). New York, NY: ACM. doi:10.1145/1124772.1124839

Fischer, K. (1980). A theory of cognitive development: The control and construction of hierarchies of skills. *Psychological Review, 87*(4), 477-531. doi:10.1037/0033-295X.87.6.477

Flavell, J. H. (1996). Piaget's legacy. *Psychological Science, 7(*4), 200-203. doi:10.1111/j.1467-9280.1996.tb00359.x

Fry, B., & Reas, C. (n.d.). *Overview: A short introduction to the Processing software and projects* *from the community*. Retrieved from http://processing.org/about/

Gallardo, D., Julià, C. F., & Jordà, S. (2008). TurTan: A tangible programming language for creative exploration. In *Proceedings of the Third IEEE International Workshop on Horizontal Interactive Human Computer Systems* (pp. 89-92). doi:10.1109/TABLETOP.2008.4660189

Gardner, H., Kornhaber, M. L., & Wake, W. K. (1996). *Intelligence: Multiple perspectives.* Fort Worth, TX: Harcourt Brace College.

George, R., & Kaplan, D. (1997). A structural model of parent and teacher influences on science attitudes of eighth graders: Evidence from NELS: 88. *Science Education*, *82*(1), 93-109. doi:10.1002/(SICI)1098-237X(199801)82:1<93::AID-SCE5>3.0.CO;2-W

Gilbert, J. E., & Swanier, C. A. (2008). Learning styles: How do they fluctuate? *Institute for Learning Styles Journal 1*(Fall), 29-40.

Granott, N., & Parziale, J. (2002). Introduction. In N. Granott & J. Parziale (Eds.), *Microdevelopment: Transition processes in development and learning* (pp. 1-28). Cambridge, UK: Cambridge University.

Gutnick, A. L., Robb, M., Takeuchi, L., & Kotler, J. (2011). *Always connected: The new digital media habits of young children*. New York, NY: The Joan Ganz Cooney Center at Sesame Workshop.

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM, 51*(8), 25-27. doi:10.1145/1378704.1378713

Heinz-Martin, S., Oberauer, K., Wittman, W. W., Wilhelm, O., & Schulze, R, (2002). Working-memory capacity explains reasoning ability - and a little bit more. *Intelligence, 30*(3), 261-288. doi:10.1016/S0160-2896(01)00100-3

Horn, M. S., Crouser, R. J., & Bers, M. U. (2011). Tangible interaction and learning: The case a hybrid approach. *Personal and Ubiquitous Computing, Special Issue: Tangibles and Children*. doi:10.1007/s00779-011-0404-2

Horn, M. S., Solovey, E. T., & Jacob, R. J. (2008). Tangible programming and informal science learning: Making TUIs work for museums. In *Proceedings of the Seventh International Conference on Interaction Design and Children* (pp. 194-201). New York, NY: ACM. doi:10.1145/1463689.1463756

International Technology Education Association (2007). *Standards for technological literacy: Content for the study of technology*. Retrieved from http://www.iteaconnect.org/TAA/PDFs/xstnd.pdf

International Society for Technology Education & The Computer Science Teachers Association (2011). *Operational definition of computational thinking for K-12 education*. Retrieved from http://www.iste.org/Libraries/PDFs/Operational_Definition_of_Computational_Thinking.sflb.ashx

Ishii, H. (2008). Tangible bits: Beyond pixels. In *Proceedings of the Second International Conference on Tangible and Embedded Interaction* (xv-xxv). New York, NY: AMC. doi:10.1145/1347390.1347392

Ito, M. (2009). *Engineering play.* Cambridge, MA: MIT.

Jenkins, H., Purushotma, R., Weigel, M., Clinton, K., & Robison, A. J. (2009). *Confronting the challenges of participatory culture: Media education for the 21st century.* Cambridge, MA: MIT.

Jones, S., & Burnett, G. (2008). Spatial ability and learning to program. *Human Technology: An Interdisciplinary Journal on Humans in ICT Environments, 4*(1), 47-61.

Kahn, K. (1996). ToonTalk™ - An animated programming environment for children. *Journal of visual languages and computing, 7*(2), 197-217. doi:10.1006/jvlc.1996.0011

Kazakoff, E. R., & Bers, M. U. (2011). *The impact of programming robots on sequencing skills in kindergarten.* Manuscript submitted for publication.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys, 37*(2), 83–137. doi:10.1145/1089733.1089734

Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive psychology, 20*(3), 362-404. doi:10.1016/0010-0285(88)90004-7

Kogan, N., & Saarni, C. (1990). Cognitive styles in children: Some evolving trends. In O. N. Saracho (Ed.), *Special aspects of education: Vol. 12. Cognitive style and early education* (pp. 3-31). Amsterdam, Netherlands: Gordon and Breach.

Kozhevnikov, M. (2007). Cognitive styles in the context of modern psychology: Toward an integrated framework of cognitive style. *Psychological Bulletin, 133*(3), 464-481. doi:10.1037/0033-2909.133.3.464

Krechevsky, M. (1998). *Project Spectrum: Preschool assessment handbook*. New York, NY: Teachers College.

Kuhn, D. (1995). Microgenetic study of change: What has it told us? *Psychological Science, 6*(3), 133-139. doi:10.1111/j.1467-9280.1995.tb00322.x

Lau, W. W. F., & Yuen, A. H. K. (2009). Exploring the effects of gender and learning styles on computer programming performance: Implications for programming pedagogy. *British Journal of Educational Technology, 40*(4), 696–712. doi:10.1111/j.1467-8535.2008.00847.x

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., . . . Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32-37. doi:10.1145/1929887.1929902

Lee, M. C., & Thompson, A. (1997). Guided instruction in Logo programming and the development of cognitive monitoring strategies among college students. *Journal of Educational Computing Research, 16*(2), 125-144. doi:10.2190/PW3F-HLFD-1NNJ-H77Q

Levy, S. T., & Mioduser, D. (2008). Does it “want” or “was it programmed to…”? Kindergarten children’s explanations of an autonomous robot’s adaptive functioning. *International Journal of Technology and Design Education, 18*(4), 337-359. doi:10.1007/s10798-007-

9032-6

Liao, Y.-K. C., & Bright, G. W. (1991). Effects of computer programming on cognitive outcomes: A meta-analysis. *Journal of Educational Computing Research, 7*(3), 251-268. doi:10.2190/E53G-HH8K-AJRR-K69M

Lewis, M. D. (2000). The promise of dynamic systems approaches for an integrated account of human development. *Child Development, 71*(1), 36-43.

Lightfoot, C., Cole, M., & Cole, S. (Eds.) (2009). *The development of children* (6th ed.). New York, NY: Worth.

Logo Foundation (2000a). *Logo Software.* Retrieved from http://el.media.mit.edu/logo-foundation/products/software.html

Logo Foundation (2000b). *What Is Logo?* Retrieved from http://el.media.mit.edu/logo-foundation/logo/index.html

Mancy, R., & Reid, N. (2004). Aspects of cognitive style and programming. In E. Dunican & T. R. G. Green (Eds.) *Proceedings of the Sixteenth Workshop of the Psychology of Programming Interest Group* (pp. 1-9).

Marshall, P. (2007). Do tangibles enhance learning? *Proceedings of the First International Conference on Tangible and Embedded Interaction* (pp. 163-170). New York, NY: ACM. doi:10.1145/1226969.1227004

Martin, F., Mikhak, B., Resnick, M., Silverman, B., & Berg, R. (2000). To mindstorms and beyond: Evolution of a construction kit for magical machines. In A. Druin & J. A. Hendler (Eds.) *Robots for kids: Exploring new technologies for learning* (pp. 9-33). San Francisco, CA: Morgan Kaufman.

Massachusetts Department of Education (2006). *Massachusetts science and technology/engineering curriculum framework*. Retrieved from http://www.doe.mass.edu/frameworks/scitech/1006.pdf

Massachusetts Department of Education (2008). *Massachusetts technology literacy standards and expectations.* Retrieved from http://www.doe.mass.edu/edtech/standards/itstand.pdf

McDevitt, T. M., & Ormrod, J. E. (2002). *Child development and education.* Upper Saddle River, NJ: Merrill/Prentice Hall.

McGill, T. J., & Volet, S. E. (1997). A conceptual framework for analyzing students' knowledge of programming. *Journal of Research on Computing in Education, 29*(3), 276-297.

Mioduser, D., Levy, S., & Talis, V. (2009). Episodes to scripts to rules: Concrete-abstractions in kindergarten children's explanations of a robot's behaviors. *International Journal of Technology and Design Education, 19*(1), 15-36. doi:10.1007/s10798-007-9040-6

Mioduser, D., & Levy, S. (2010). Making sense by building sense: Kindergarten children's construction and understanding of adaptive robot behaviors. *International Journal of Computers for Mathematical Learning, 15*(2), 99-127. doi:10.1007/s10758-010-9163-9

NAEYC (2011). *Technology in early childhood programs serving children from birth through age 8* (Draft). Retrieved from http://www.naeyc.org/files/naeyc/file/positions/PSTECH98.PDF

New Media Consortium (2005). *A global imperative: The report of the 21st century literacy summit*. Retrieved from http://www.nmc.org/pdf/Global_Imperative.pdf

Nir-Gal, O., & Klein, P. S. (2004). Computers for cognitive development in early childhood: The teacher's role in the computer learning environment. *Information Technology in Childhood Education Annual, 2004*(1), 97-119.

Papert, S. (1987). Computer criticism vs. technocentric thinking. *Educational Researcher, 16*(1), 22-30. doi:10.3102/0013189X016001022

Papert, S. (1993). *Mindstorms: Children, computers, and powerful ideas.* New York, NY: Basic Books.

Papert, S. (1999). Logo: What is Logo? Who needs it? In *Logo philosophy and implementation*. Retrieved from www.microworlds.com/company/philosophy.pdf

Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal, 39*(3 & 4), 720-729. doi:10.1147/sj.393.0720

Papert S., & Harel, I. (1991). Situating constructionism. In S. Papert & I. Harel (Eds.) *Constructionism* (pp. 1-11). Norwood, NJ: Ablex.

Patten, J., Griffith, L., & Ishii, H. (2000). A tangible interface for controlling robotic toys. In *CHI '00 Proceedings Extended Abstracts on Human Factors in Computing Systems* (pp. 278-279). New York, NY: ACM. doi:10.1145/633292.633454

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research, 2*(1), 25-36. doi:10.2190/689T-1R2A-X4W4-29J2

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*(2), 137-168. doi:10.1016/0732-118X(84)90018-7

Pea, R. D., Kurland, D. M., & Hawkins, J. (1985). Logo and the development of thinking skills. In M. Chen & W. Paisley (Eds.) *Children and microcomputers: Research on the newest medium*. Beverly Hills, CA: Sage.

Peppler, K. A., & Kafai, Y. B. (2007). From SuperGoo to Scratch: Exploring creative media production in informal learning. *Learning, Media and Technology, Special Issue: Media Education Goes Digital, 32*(2), 149-166. doi:10.1080/17439880701343337

Perlman, R. (1976). Using computer technology to provide a creative learning environment for preschool children (AI Memo 360: Logo Memo No. 24). Cambridge, MA: MIT Artificial Intelligence Laboratory.

Raffle, H. S., Parkes, A. J., & Ishii, H. (2004). Topobo: A constructive assembly system with kinetic memory. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 647-654). New York, NY: ACM. doi:10.1145/985692.985774

Reisberg, D. (2010). *Cognition: Exploring the science of the mind.* New York, NY: W. W. Norton.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the Forty-First ACM Technical Symposium on Computer Science Education* (pp. 265-269).

New York, NY: ACM. doi:10.1145/1734263.1734357

Resnick, M. (2006). Computer as paintbrush: Technology, play, and the creative society. In D. Singer, R. Golikoff, & K. Hirsh-Pasek (Eds.) *Play = learning: How play motivates and enhances children's cognitive and social-emotional growth.* Oxford, UK: Oxford University.

Resnick, M., (2007). Sowing the seeds of a more creative society. *Learning and Leading with Technology, 35*(4), 18-22. doi:10.1108/14777280710828549

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., . . . Kafai, Y. (2006). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60-67. doi:10.1145/1592761.1592779

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

Robins, A., Rountree, J., & Rountree, N. (2003). Teaching and learning programming: A review and discussion. *Computer Science Education, 13*(2), 137-172. doi:10.1076/csed.13.2.137.14200

Salomon, G., & Perkins, D. N. (1987). Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research, 3*(2), 149-169. doi:10.2190/6F4Q-7861-QWA5-8PL1

Sharf, F., Winkler, T., & Herczeg, M. (2008). Tangicons: Algorithmic reasoning in a collaborative game for children in kindergarten and first class. In *Proceedings of the Seventh International Conferences on Interaction Design and Children* (pp. 242-249). New York, NY: ACM. doi:10.1145/1463689.1463762

Shipman S., & Shipman, V. C. (1985). Cognitive styles: Some conceptual, methodological, and applied issues. *Review of Research in Education, 12*, 229-291. doi:10.3102/0091732X012001229

Shuler, C. (2007). *D is for digital: An analysis of the children's interactive media environment with a focus on mass marketed products that promote learning*. New York, NY: Joan Ganz Cooney Center at Sesame Workshop.

Shute, V. J. (1991). Who is likely to acquire programming skills? *Journal of Educational Computing Research, 7*(1), 1-24. doi:10.2190/VQJD-T1YD-5WVB-RYPJ

Siegler, R. S., & Crowley, K. (1991). The microgenetic method: A direct means for studying cognitive development. *American Psychologist, 46*(6), 606-620.

Smith, A. C. (2007). Using magnets in physical blocks that behave as programming objects. In *Proceedings of the First International Conference on Tangible and Embedded Interaction* (pp. 147-150). New York, NY: ACM. doi:10.1145/1226969.1226999

Smith, D. C., Cypher, A., & Tesler, L. (2000). Novice programming comes of age. *Communications of the ACM, 43*(3), 75-81. doi:10.1145/330534.330544

Sternberg, R. J., & Grigorenko, E. L. (1997). Are cognitive styles still in style? *American Psychologist, 52*(7), 700-712. doi:10.1037/0003-066X.52.7.700

SuperPowers of Play. *The SuperPowers*. Retrieved from http://www.superpowersofplay.com/?page_id=47

Suzuki, H., & Kato, H. (1995). Interaction-level support for collaborative learning: AlgoBlock – an open programming language. In *Proceedings of the First International Conference on Computer Support for Collaborative Learning* (pp. 349-355). Hillsdale, NJ: Erlbaum.

van Geert, P. (1998). A dynamic systems model of basic developmental mechanisms: Piaget, Vygotsky, and beyond. *Psychological Review, 105*(4), 634-677. doi:10.1037//0033-295X.105.4.634-677

Vernier (n.d.). *ROBOLAB and NXT*. Retrieved from http://www.vernier.com/nxt/robolab.html

Wartella, E. A., & Jennings N. (2000). Children and computers: New technology – old concerns. *The Future of Children, 10*(2), 31-43.

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society – Series A, 366*, 3717-3725. doi:10.1098/rsta.2008.0118

Winslow, L. E. (1996). Programming pedagogy: A psychological view. *ACM SIGSCE Bulletin 28*(3), 17-25. doi:10.1145/234867.234872

Wyeth, P., & Wyeth, G. (2001). Electronic blocks: Tangible programming elements for preschoolers. In M. Hirose (Ed.) *Proceedings of IFIP INTERACT01: Conference on Human-Computer Interaction* (pp. 496-503). Tokyo, Japan: IOC.

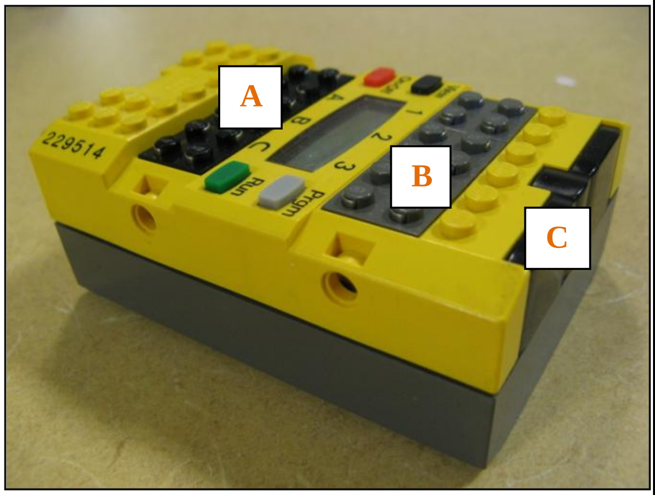
Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *Journal of the Learning Sciences, 17*(4), 517-550. doi:10.1080/10508400802395069

Appendix A – Resources for Learning to Use CHERP with RCX™ Robots

1. Parts of a Lego™ RCX™ Robot
2. Building a Program with CHERP
3. Downloading a Program to a Robot

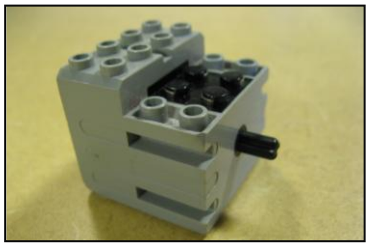
Parts of a Lego™ RCX Robot

RCX / "Brain"



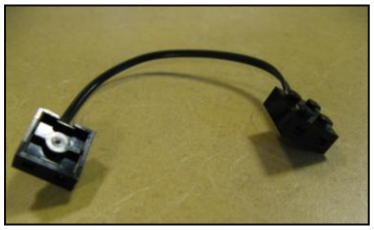
- The red button turns the RCX on and off. The green button starts and stops programs. The grey button switches between 5 programs, indicated as 1-5 on right side the RCX's screen.
- ***Tip:*** Make sure you know which number your program is!
- **A**: The black 'ports' A-C power motors and lights from batteries inside the RCX according to your program.
- **B**: The grey 'ports' 1-3 provide data from sensors to the RCX. CHERP only uses Port 1.
- **C**: The infra-red port "listens" for instructions from the computer and sends them to the "brain" inside the RCX.

Motors



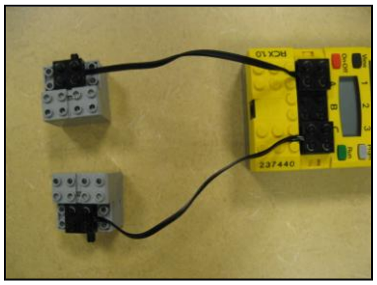
- Connect to Ports A and C with wires that go from Port A/C to the black port seen on the motor.
- ***Tip:*** **The orientation of the wire ends on each port affects the direction the motors turn. See tip below.**

Wires



- Provide electrical connections from ports on the RCX to components like motors and lights.
- ***Tip:*** **Make sure the wires do not rub the robot's wheels – this can slow them down!**

Wire Orientations



- ***Tip:*** **This orientation of wire ends and motors will result in your robot moving as expected.**

Lights

- Connects to Port B directly or via a wire.
- ***Tip:*** **See programming tips on light blocks.**

Building a Program with CHERP

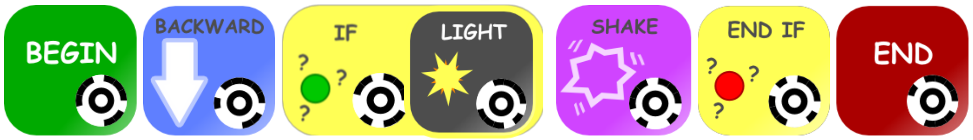
You can make the same programs with the on-screen (graphical) blocks and the wooden (tangible) blocks.

- Typing Control+1 (or 2 or 3) reveals the corresponding number of rows of programming instructions in the on-screen interface. This does not work in full-screen mode. Hit Enter / Escape to enter and exit full-screen mode.

- Every program must start with a BEGIN block and end with and END block:

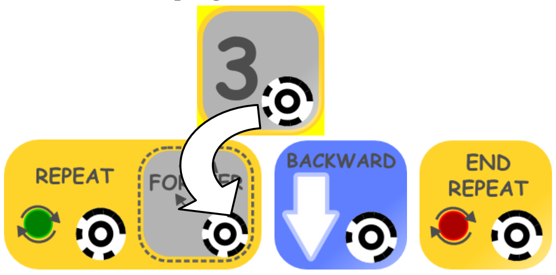
- Graphical blocks will ONLY connect to a BEGIN block or to a sequence of connected blocks. Unconnected graphical blocks will appear pale and will not be downloaded to the robot.

- REPEATs and Ifs must be paired with their corresponding END block. The relevant action(s) go in between, like this:



- REPEAT and IF blocks have a light or dark grey space, respectively, for a parameter – additional information that says how many times the instructions will be repeated. The REPEAT parameter is optional since the default is to REPEAT FOREVER.

- With the tangible interface, any parameters' circular barcodes must align with those of the other blocks and must be visible to the camera to download the program to a robot.



Keep in mind:

- Attach new graphical blocks to a program by dragging and dropping the new block wherever you want it. Click on any graphical block in the program to move that block and all blocks connected to its right.

- To get rid of graphical blocks from the workspace, drag them to anywhere in the rows of available blocks at the bottom of the screen. To clear all attached graphical blocks at once, click the BEGIN block and drag down.

- Blocks will be interpreted by the robot sequentially starting with the BEGIN block. For instance, in the IF example above, the robot will go backwards once and then, if the light sensor detects bright light, it will shake.

- Once you download any program (graphical/tangible) to a robot, an editable on-screen version of it appears.

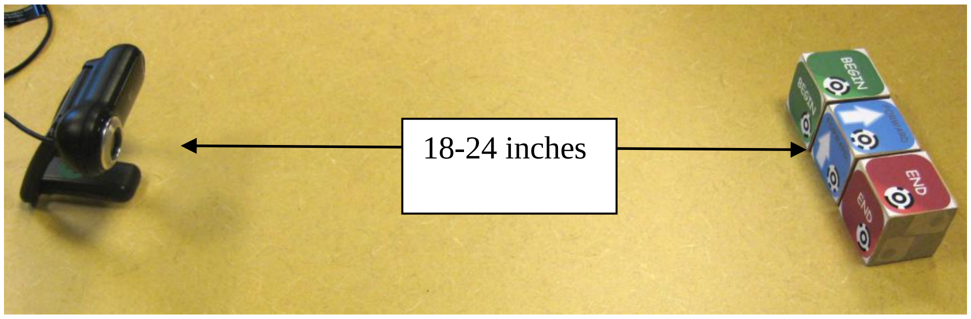
- The motion and sound blocks instruct the robot to do an action for half a second, then stop. **The light blocks work differently.** LIGHT ON turns the light on until LIGHT OFF is used. If there is no LIGHT OFF before the end of the program, the light will already be on at the start of the next program. The light blocks also make the robot do the next instruction immediately, not after half a second. It helps to think through the state of the light throughout your program. Challenge: How might you make the light blink?

LIGHT ON
LIGHT OFF

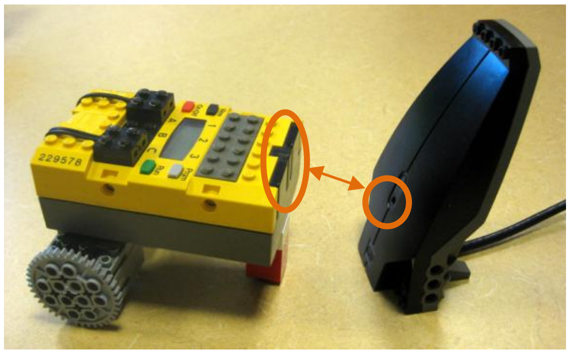
Downloading a Program to a Robot

If you are using the graphical blocks, skip to #2.

1. Place the tangible blocks directly facing the webcam about 18-24 inches away from it so the computer vision can detect your program properly. You may be prompted to include a BEGIN block in your program. If you do have a BEGIN block and it appeared within the image shown on-screen during the attempted download, change the distance or angle between the webcam and your program and re-download the program. ***Tip:*** You can put the blocks across the table from the webcam or on the floor under the downward-pointing webcam.



2. Position the IR port (the smooth black rectangle) of your RCX-based robot near the front of the LEGO™ USB Tower and make sure that the RCX is turned on and has firmware loaded (you will see numbers counting up on the RXC's screen).



3. Press the appropriate on-screen download button (the mouse for graphical, left; the blocks for tangible, right). The RCX will play a rising series of beeps when the download is complete.
4. Place your robot where it can safely move around (usually an open space on the floor) and press the green "Run" button on the RCX to run the program.

IMPORTANT: If your robot does not turn on, or if it turns on but has no numbers counting up on the screen, or if your program has all the right parts but CHERP gives you this error message: "Your program is missing something," it needs new firmware.

ALWAYS turn off your robot while you are not downloading a program to it or running a program. It will use up its batteries very quickly if it is left on!

Also Note: To download a graphical program it is **NOT** necessary to remove tangible blocks from in front of the webcam; likewise, to download a tangible program, it is **NOT** necessary to remove any graphical blocks from the screen. **However**, it **IS** a good idea to remove extraneous tangible blocks from the webcam's view when downloading a tangible program.

Appendix B – Variables and Instruments

Table B1

Overview of Conceptual Variables

Variables	Definitions	Nature
Primary Analysis		
Stage of cognitive development[1]	Pre-operations, Phase 2; Transitional; or Concrete Operations, Phase 1	Independent
Approach to programming[2]	Characteristics of goals and cognitive strategies used in programming	Dependent
Programming achievement: correspondence	Ability to match programming instructions to planned robot actions	Dependent
Programming achievement: final program completeness	Ability to make correct action-instruction correspondences and to sequencing the instructions to achieve the given goal	Dependent
Secondary Analysis (includes the same dependent variables as above)		
Sequencing	Ability to sequence four-part picture stories.	Independent
Demographics	Child’s age, grade, gender, and home area type	Independent
Child’s prior experience	Prior experience with computers, robotics, and/or programming	Independent
Parents’ level of education	Highest degree attained by either parent	Independent
Parents’ STEM involvement	Whether a parent has a STEM degree or job	Independent

Notes: [1]See Table B4 for further definitions of cognitive developmental levels.

[2]Confounded with stage of cognitive development.

Table B2

Primary Analysis Variables, Measures, and their Derivations

Variables	Measure	Derivation	Type
Stage of cognitive development	Framework of Cognitive Developmental Traits Mapped to Programming	Composite of sub-scores.[1]	Ordinal
Programming approach	Framework of Cognitive Developmental Traits Mapped to Programming	Each of the sub-scores on: goal focus, initial attempt, and debugging. [1, 2]	Ordinal
Programming achievement	Correspondence of instructions to actions	Likert scale: 0 (cannot achieve) to 5 (achieves with little or no help)	Ordinal
	Final program completeness (correct instructions, in order)	Likert scale: 0 (did not attempt) to 4 (correct instructions and order). See Appendix B6for rubric.	Ordinal

Notes: [1]See Appendices B3 and B4 for derivation and definition.

[2]Note that programming approach is confounded with the definition of cognitive developmental stage.

Table B3

Map of Cognitive Developmental Characteristics to Programming Behaviors

Ability or Activity	Stage and Phase	Expected Programming Observations
Symbol-system elaboration	Seen during phase 2 of pre-operations	Has a lot of ideas and enthusiasm for exploring with CHERP
Determines causal relationships empirically	Appears in late pre-operations, solidifies in concrete operations	Systematic rather than random or intuitive debugging
Relies on logic over perception	Appears in late pre-operations, solidifies in concrete operations	Changes debugging hypotheses based on evidence
Differentiates physical & psychological events	Appears in late pre-operations, solidifies in concrete operations	Attribution of robot's unexpected behaviors to the program vs. intentionality on the part of the robot
Decentration from own physical perspective	Appears in late pre-operations, solidifies in concrete operations	Constrained by orientation of self vs. computer vs. blocks vs. robot while building the program vs. robot on map
Decentration from a single (superficial) aspect	Appears in late pre-operations, solidifies in concrete operations	Debugs single vs. multiple aspects of the robot and software system
Classification of single objects into multiple categories and of multiple objects into hierarchies	Develops over concrete operations	Differentiates blocks with qualitatively different functions; pursuit of the “best” answer
Reasons deductively	Develops over concrete operations	Logical reasoning in how to debug
Plans towards a goal & thinks flexibly in strategizing towards its successful completion	Develops over concrete operations	Works towards adult-given goal; tries a Plan B if Plan A fails

Table B4

Cognitive Stage Markers in Programming Rubric

Pre-Operational	Transitional	Concrete Operational
	Goal Orientation	
Focuses primarily or exclusively on open-ended exploration. May try the Hokey-Pokey (HP) nominally or cursorily. [1]	Tries HP (with interest and effort) but leaves it due to interest in other explorations or being unable to debug further (may claim an incomplete program is successful).	Focuses primarily on HP with little or no redirection through to a nearly or fully complete solution. May explore openly before/during HP.
	Initial Solution	
Nominal, cursory, or no attempt. OR Intuitive approach (considers actions but not order).	Intuitive approach with limited systematic logic (order).	Logical approach (step-by-step sequencing).
	Debugging Attitudes and Strategies	
Indifferent to the need to debug or to the results of any unsuccessful efforts.	Interested in improving the program but cannot figure out how.	Driven to find best answer. OR Gets answer right away and knows it.
Nominal, cursory, or no attempt. OR Intuitive approach (e.g. guess-and-check).	Mixed approach intuitive / logical & empirical. Limited / inflexible ideas on how to systematically debug.	Logical / empirical approach. Flexible if one idea does not work.
	Perspective and Classification	
Attributes agency inappropriately to self versus the robot.		Attributes agency appropriately to self versus the robot.
Confused by different orientations of the computer, blocks, robot, map, and self.		Unconstrained by different orientations of the computer, blocks, robot, map, and self.
Single classification for “blocks.”		Multiple classifications for “blocks.”

Notes: “HP” stands for “the Hokey-Pokey.”

[1]This includes situations in which the child verbally claims to be working on the Hokey-Pokey, perhaps in an effort to avoid conflict with a perceived authority figure, but actually makes a completely unrelated program and shows through other behavior or speech that the Hokey-Pokey is not the actual goal.

Correspondence Achievement Rubric

<u>Instructions</u>: Choose the level of assistance the child requires to successfully apply the concept of selecting the correct block for the program based on its corresponding action. If necessary and possible, score may be based on the gap between the child's independent work and a complete understanding.

Level	Definition
5	Achieves without assistance
4	Achieves with minimal assistance
3	Achieves with periodic assistance
2	Achieves with significant assistance
1	Achieves with step-by-step assistance
0	Cannot achieve
NA	Cannot be assessed

Hokey-Pokey Program Completeness Assessment Rubric

Instructions

Acceptable programs must represent **Forward** | **Backward** | **Forward** | **Shake** | **Spin**.
Do not score **Begin**/**End**. The program may have **Turns** instead of **Spin**, **sounds** at the beginning or end, or a second, consecutive **Shake** or **Spin**.

Definitions of Fixes – Use the fewest possible fixes to reach an accepted solution.

Addition: One of the 5 basic solution instructions is missing and needs to be added.
Swap: 2 consecutive instructions need to be switched.
Deletion: An instruction needs to be removed.
Distinguish: The child consistently confused 2 similar instructions (e.g. **Backward** and **Forward**), so these instructions need to be exchanged.

Scale and Examples

4 – No fixes needed

3 – One fix

- 1 addition **Forward** | **Backward** |**Shake** | **Spin**
- 1 swap **Forward** | **Forward** | **Backward** |**Shake** | **Spin**
- 1 deletion **Forward** | **Backward** | **Forward** | **Shake** | **Backward** | **Spin**

2 – Two fixes

- 2 additions **Forward** | **Backward** | **Shake**
- 2 swaps **Forward** | **Forward** | **Backward** | **Spin** | **Shake**
- 2 deletions **Forward** | **Backward** | **Shake** | **Forward** | **Shake** | **Spin** | **Right**
- 1 addition and 1 swap **Backward** | **Forward** | **Spin** | **Shake**
- 1 addition and 1 deletion **Forward** | **Sing** | **Backward** | **Forward** | **Shake**
- 1 swap and 1 deletion **Backward** | **Forward** | **Forward** | **Shake** | **Left** | **Spin**

1 – Three+ fixes

The program is a Hokey-Pokey skeleton (only 2 correct actions) but clearly was an attempted solution, based on why the child chose the actions or on their correct relative order or proximity.

- 2 instructions, no reduplications **Forward** |**Shake** | **Sing**
- 2 instructions, with reduplications **Backward** | **Backward** | **Sing** | **Sing** |**Spin** |**Spin**
- A copy of CHERP's GUI palette, **given** the child's recognition that parts of it match the song.

0 – Avoidance of task or unrecognizable attempt.

- The child did not attempt to make a Hokey-Pokey program. OR
- The program is so incomplete as to be unrecognizable as a clear attempt at the Hokey-Pokey. (The child may or may not have claimed it to be a Hokey-Pokey attempt.)

Table B5

Secondary Analysis Variables, Measures, and their Derivations

Variables	Measure	Derivation	Type
Child's age	Child's birthday	Time between child's first study session and birthday	Scale
Child's school year	Child's grade in school	Preschool / Kindergarten	Dichotomous
Child's gender	Child's gender	Male / Female	Dichotomous
Home area	Home neighborhood type	Suburban / Urban	Dichotomous
Sequencing, pre-intervention	Baron-Cohen Picture Sequencing	Total score on 5 stories (0-10 points possible)	Interval
Sequencing change	Baron-Cohen Picture Sequencing	Difference in pre- and post- assessment scores	Scale
Child's computer use at home	Child uses a computer at home?	Yes / No	Dichotomous
Child's computer skill level	Child's skill at using a computer, for his/her age	Beginner / Average / Expert	Ordinal
Child's experience with programming	Child has prior experience with programming?	Yes / No	Dichotomous
Child's experience with robotics	Type of robots with which the child has experience	None / Media or pre-programmed toys / Programmable robots	Ordinal
Parents' level of education	Highest education level by either parent	Scale: 0 (high school) to 4 (doctoral)	Ordinal
Parents' STEM degrees	Is the latest degree of 1+ parent in a STEM field?	Yes / No	Dichotomous
Parents' STEM jobs	Is the current job of 1+ parent in a STEM field?	Yes / No	Dichotomous
Parents' experience with programming	Does 1+ parent have programming experience?	Yes / No	Dichotomous
Parents' experience with robotics	Does 1+ parent have experience with robotics?	Yes / No	Dichotomous

Table B6

Statistical Methods Employed

	Developmental Level	Correspondence	Completeness
Cognitive Factors			
Developmental Level	n/a	ANOVA	ANOVA
Goal Focus	n/a	ANOVA	ANOVA
Initial Attempt	n/a	ANOVA	ANOVA
Debugging	n/a	ANOVA	ANOVA
Cognitive Baselines			
Sequencing (Pre)	ANOVA	Regression	Regression
Sequencing (Delta)	ANOVA	Regression	Regression
Child Demographics			
Age	ANOVA	Regression	Regression
Grade	Chi-squares	2 sample t-test	2 sample t-test
Gender	Chi-squares	2 sample t-test	2 sample t-test
Home Area	Chi-squares	2 sample t-test	2 sample t-test
Child Experience			
Home computer use	Chi-squares	2 sample t-test	2 sample t-test
Computer skill level	Chi-squares	ANOVA	ANOVA
Robotics experience	Chi-squares	ANOVA	ANOVA
Parent Experience			
Education Level	Chi-squares	ANOVA	ANOVA
STEM Education	Chi-squares	2 sample t-test	2 sample t-test
STEM Job	Chi-squares	2 sample t-test	2 sample t-test
Programming experience	Chi-squares	2 sample t-test	2 sample t-test
Robotics experience	Chi-squares	2 sample t-test	2 sample t-test

Appendix C – Results

1. Mean age of each cognitive developmental level (graph)
2. Frequency of correspondence scores within the entire sample (graph)
3. Frequency of program completeness scores within the entire sample (graph)
4. Programming Achievement by Cognitive Developmental Level Statistics
5. Mean correspondence score by cognitive developmental level (graph)
6. Mean program completeness by cognitive developmental level (graph)
7. Programming Achievement by Goal Orientation Statistics
8. Mean correspondence score by goal orientation (graph)
9. Mean program completeness by goal orientation (graph)
10. Programming Achievement by Initial Program Approach Statistics
11. Mean correspondence score by initial strategy (graph)
12. Mean program completeness by initial strategy (graph)
13. Programming Achievement by Debugging Approach Statistics
14. Mean correspondence score by debugging strategy (graph)
15. Mean program completeness by debugging strategy (graph)
16. Sequencing Scores by Developmental Level Statistics
17. Programming Achievement by Sequencing Scores Statistics
18. Child and Parent Background and Correspondence Achievement Statistics
19. Child and Parent Background and Program Completeness Achievement Statistics

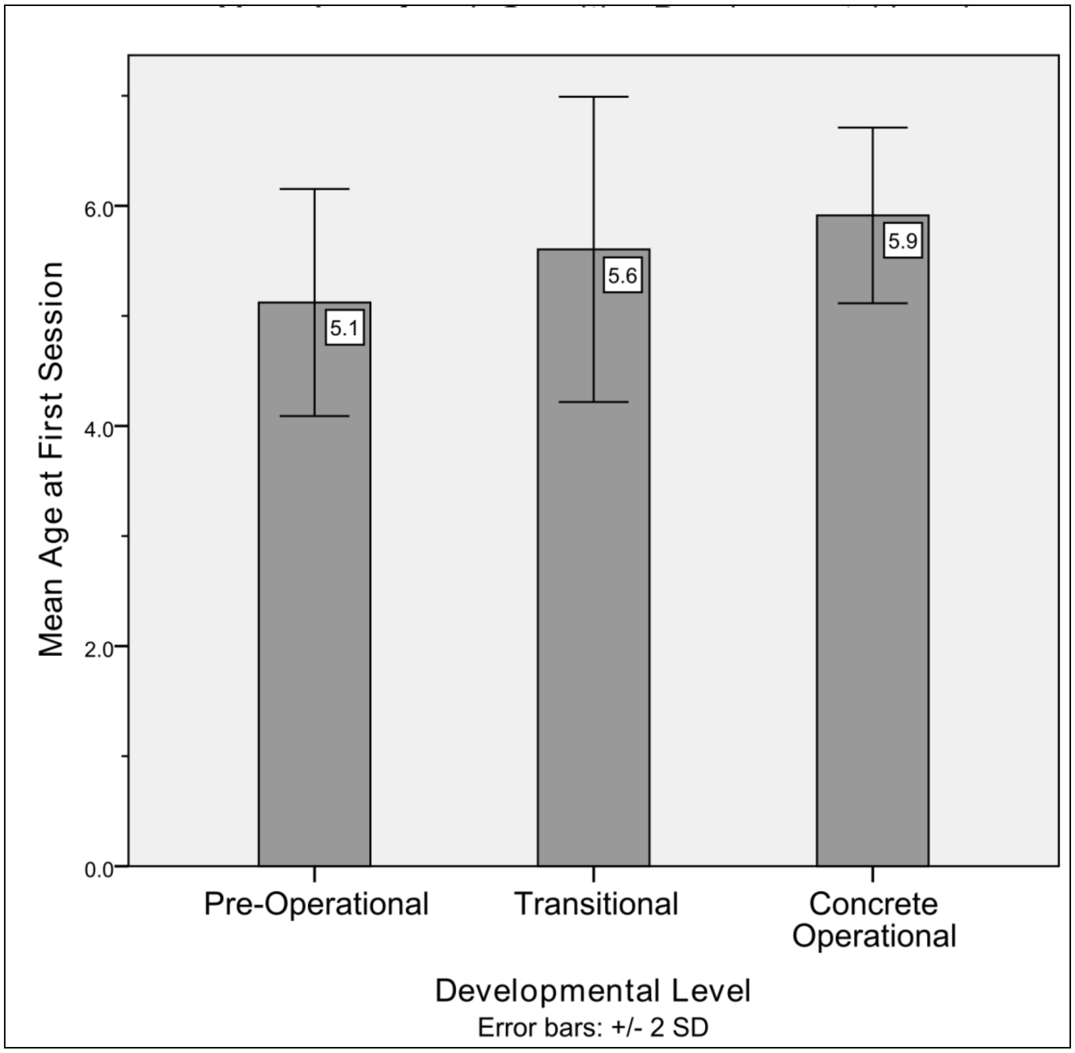


Figure C1. Mean age of each cognitive developmental level.

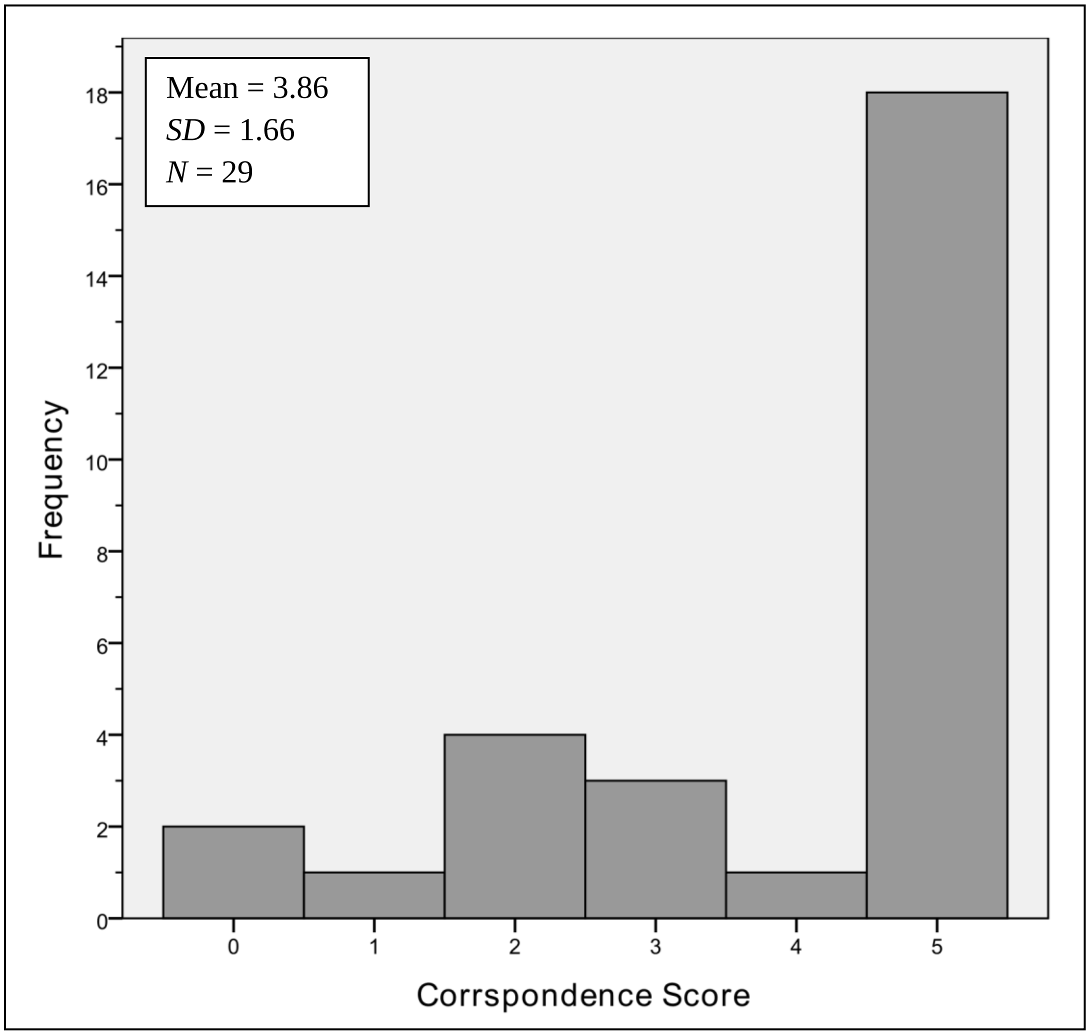


Figure C2. Frequency of correspondence scores within the entire sample.

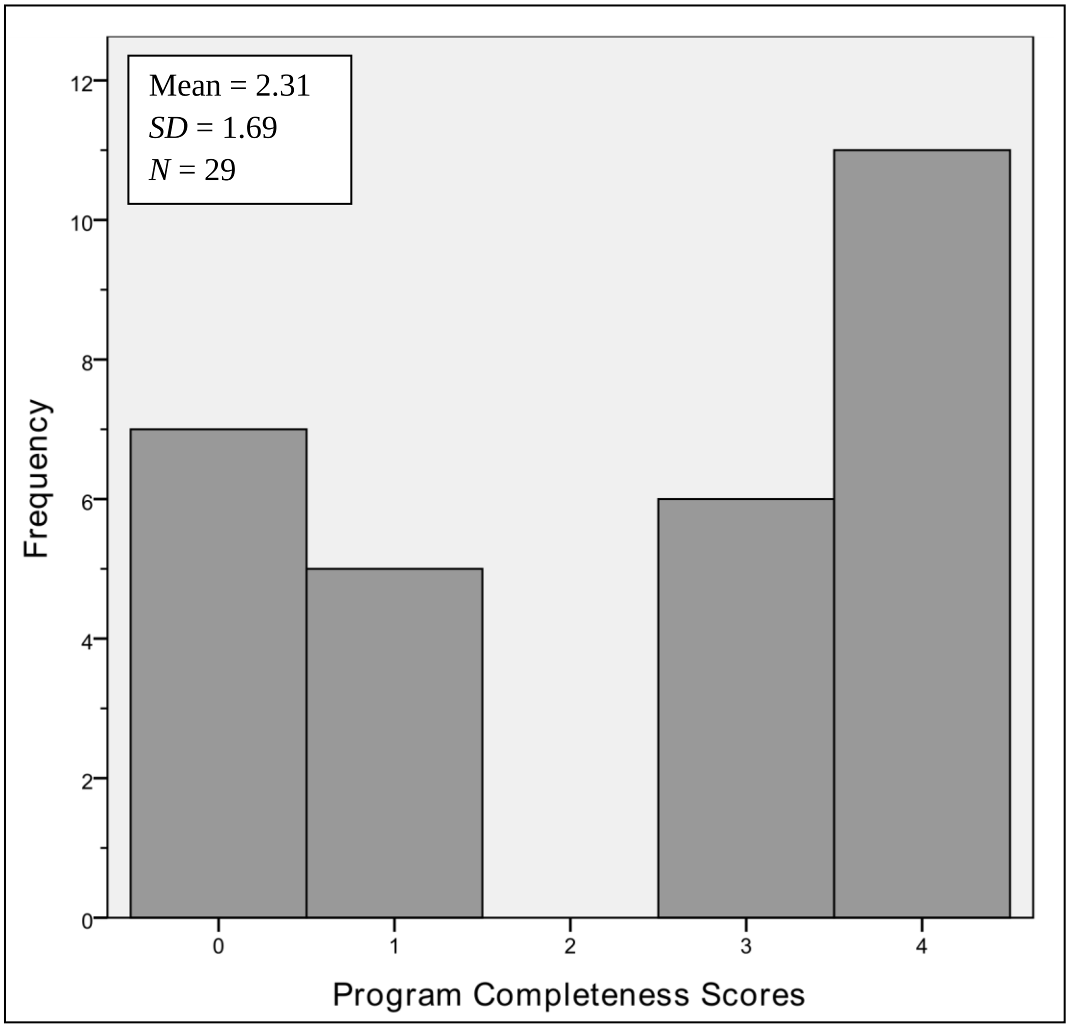


Figure C3. Frequency of program completeness scores within the entire sample.

Table C1

Programming Achievement by Cognitive Developmental Level

Outcome Variable	Level	*n*	M	SD
Correspondence	Pre-Operational	8	1.87	1.46
	Transitional	7	3.86	1.46
	Concrete Operational	14	5.00	0.00
Program completeness	Pre-Operational	8	0.13	0.35
	Transitional	7	1.86	1.07
	Concrete Operational	14	3.79	0.43

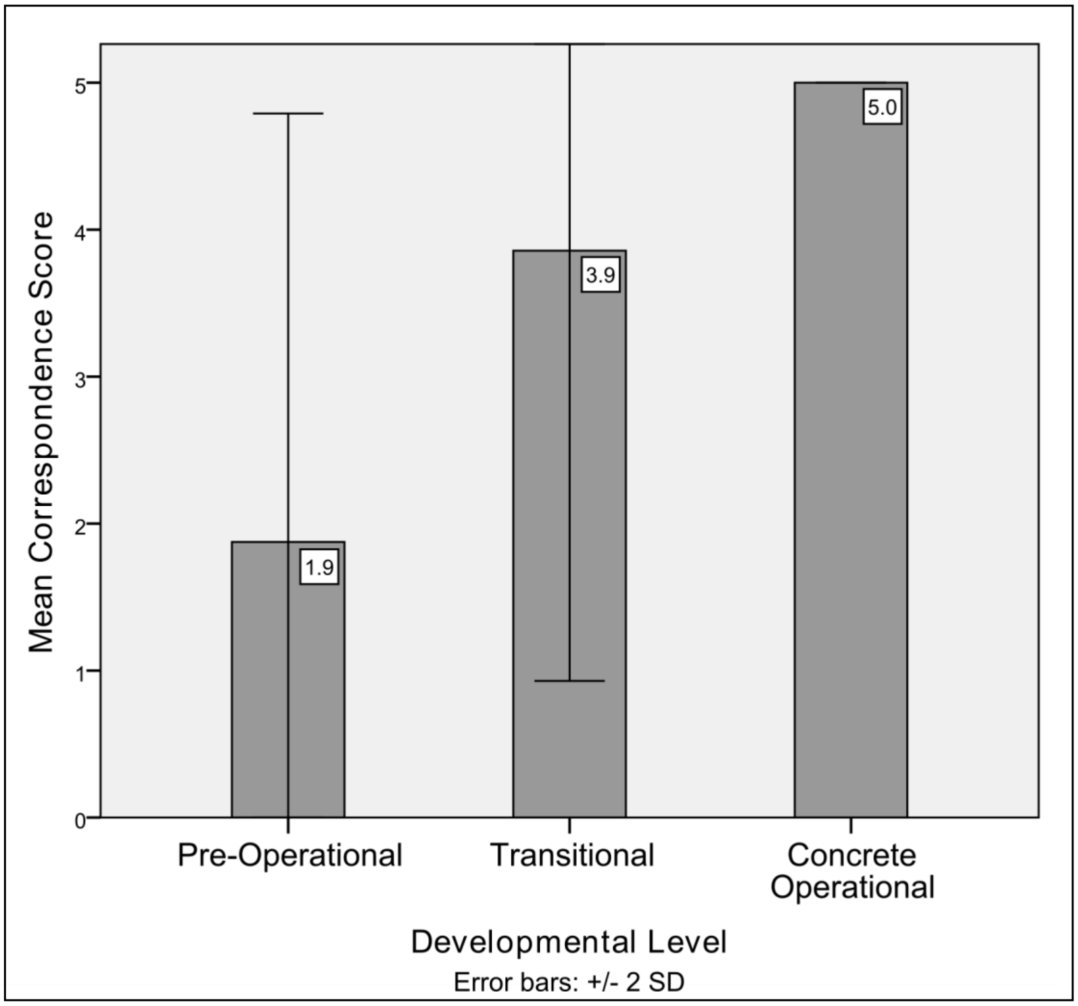


Figure C4. Mean correspondence score by cognitive developmental level.

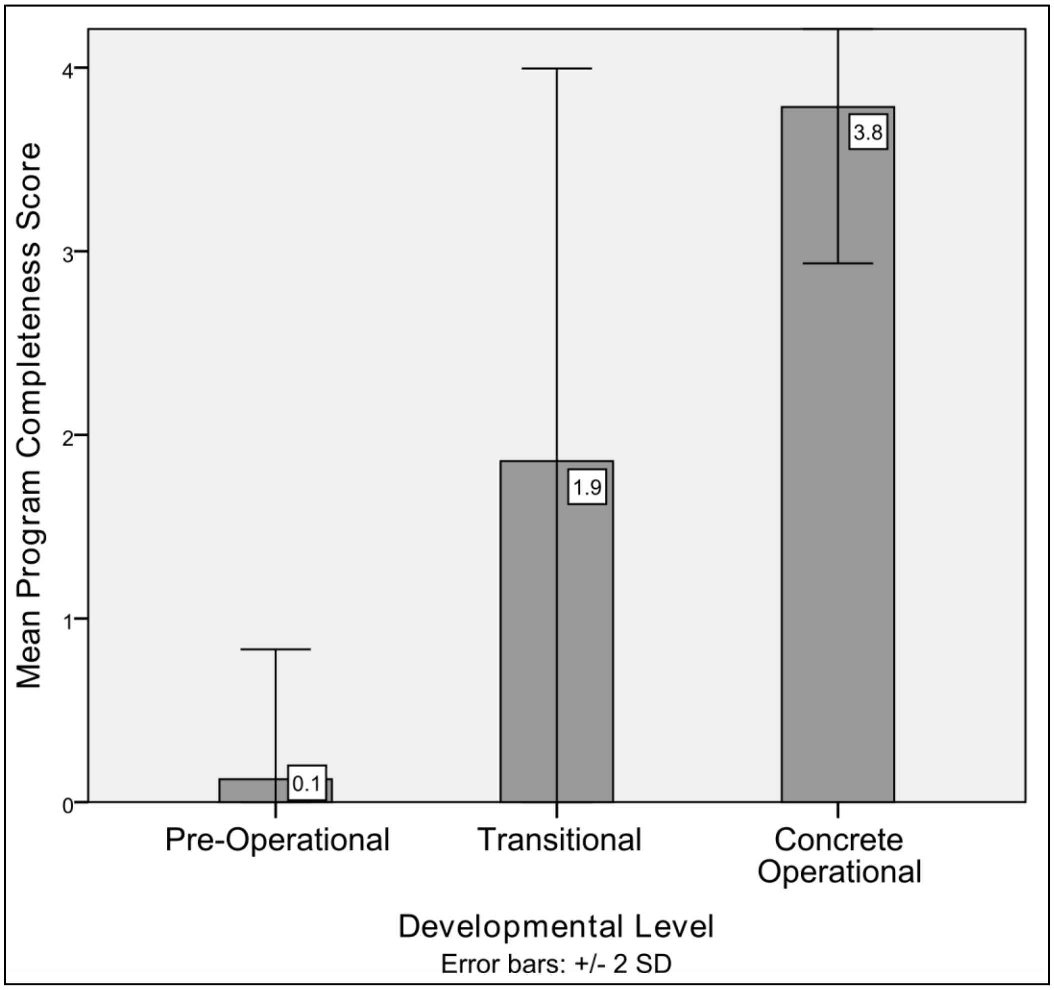


Figure C5. Mean program completeness by cognitive developmental level.

Table C2

Programming Achievement by Goal Orientation

Outcome Variable	Level	*n*	*M*	*SD*
Correspondence	Pre-Operational	4	1.50	1.29
	Transitional	8	2.62	1.51
	Concrete Operational	17	5.00	0.00
Program completeness	Pre-Operational	4	0.00	0.00
	Transitional	8	0.63	0.52
	Concrete Operational	17	3.65	0.49

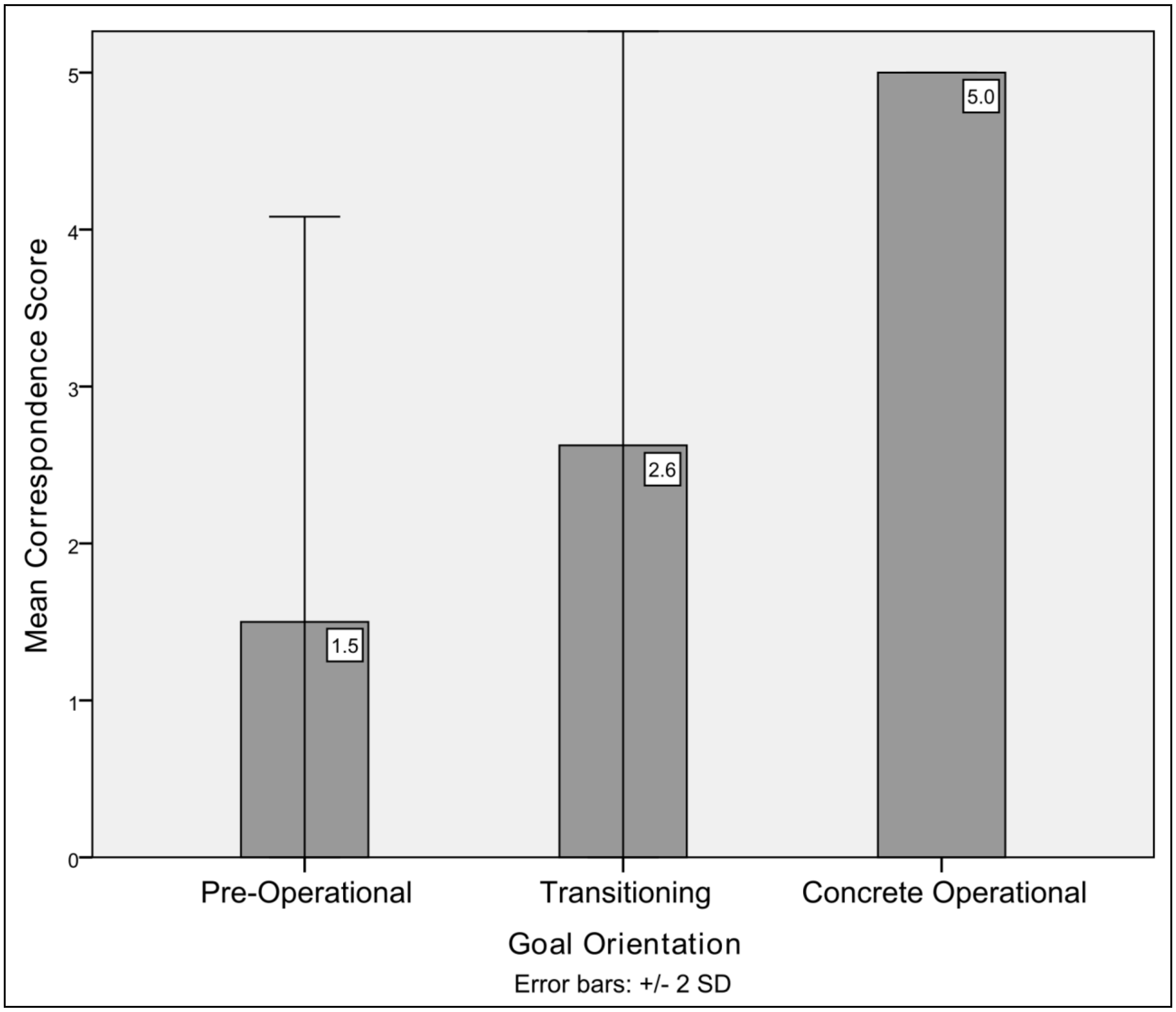


Figure C6. Mean correspondence score by cognitive developmental level of goal orientation.

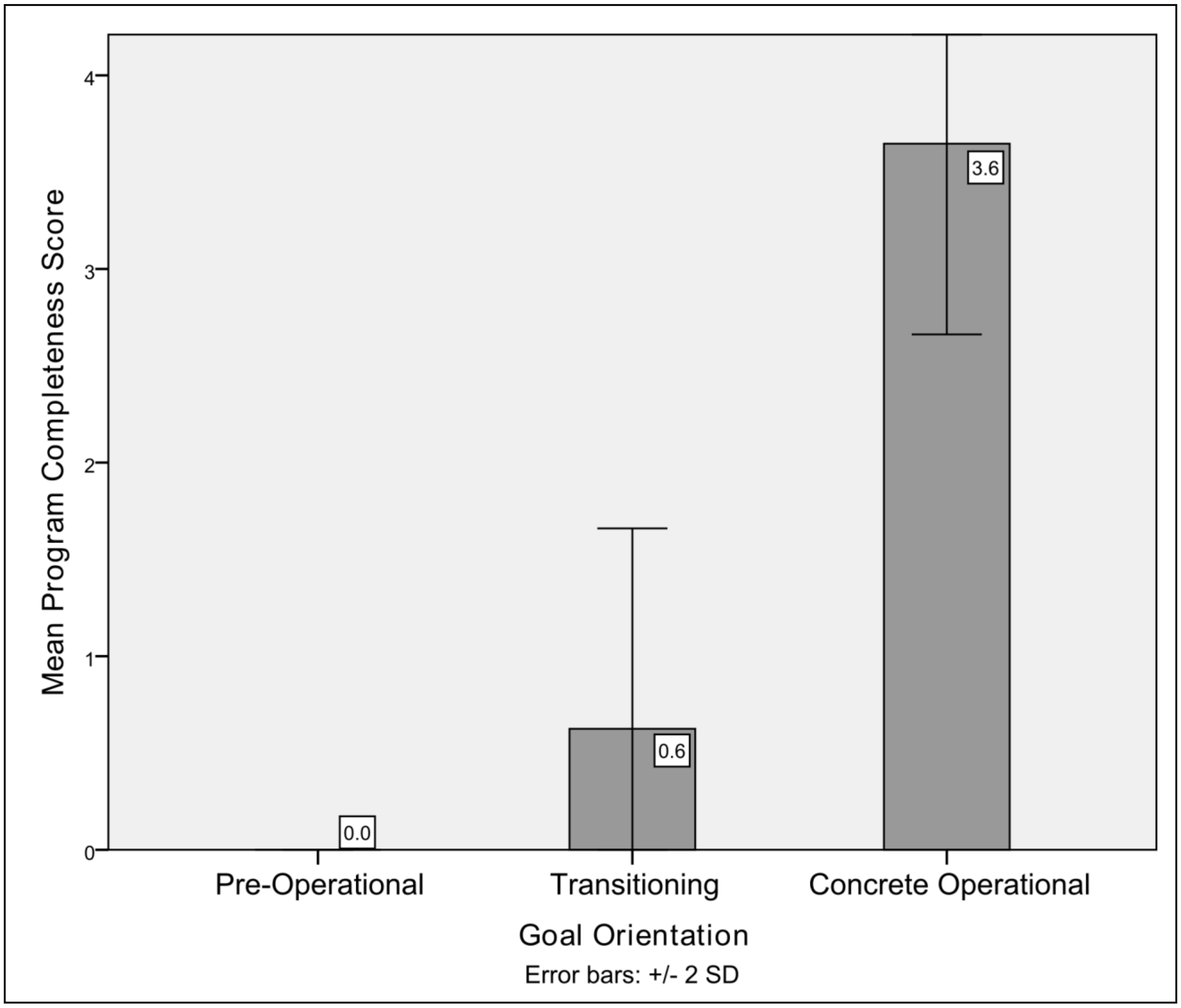


Figure C7. Mean program completeness by cognitive developmental level of goal orientation.

Table C3

Programming Achievement by Initial Program Approach

Outcome Variable	Level	*n*	*M*	*SD*
Correspondence	Pre-Operational	10	2.30	1.64
	Transitional	9	4.33	1.32
	Concrete Operational	10	5.00	0.00
Program completeness	Pre-Operational	10	0.30	0.48
	Transitional	9	2.89	1.17
	Concrete Operational	10	3.80	0.42

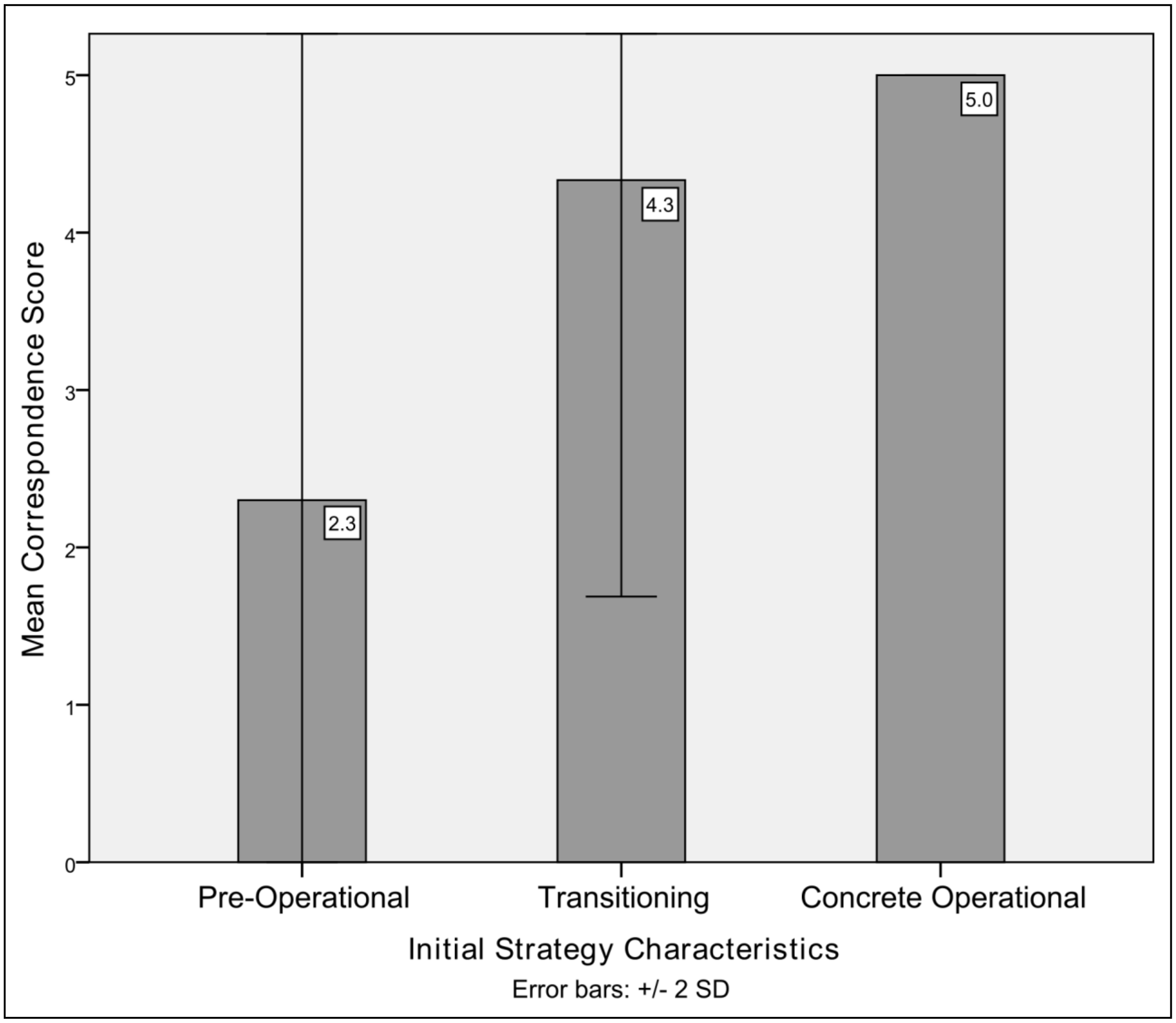


Figure C8. Mean correspondence score by cognitive developmental level of initial strategy.

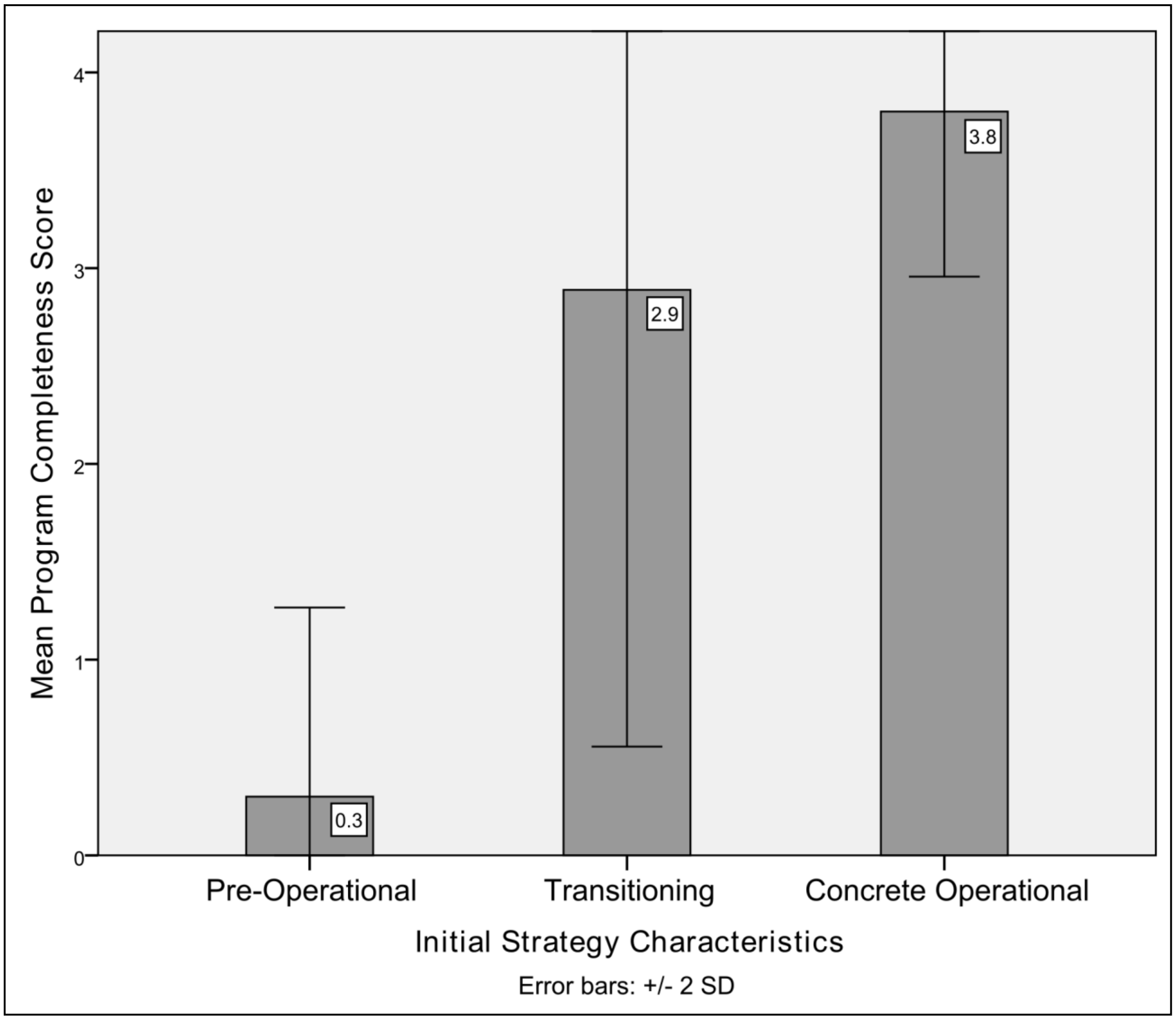


Figure C9. Mean program completeness by cognitive developmental level of initial strategy.

Table C4

Programming Achievement by Debugging Approach

Outcome Variable	Level	*n*	*M*	*SD*
Correspondence	Pre-Operational	9	2.22	1.72
	Transitional	6	3.67	1.51
	Concrete Operational	14	5.00	0.00
Program completeness	Pre-Operational	9	0.44	1.01
	Transitional	6	1.67	1.03
	Concrete Operational	14	3.79	0.43

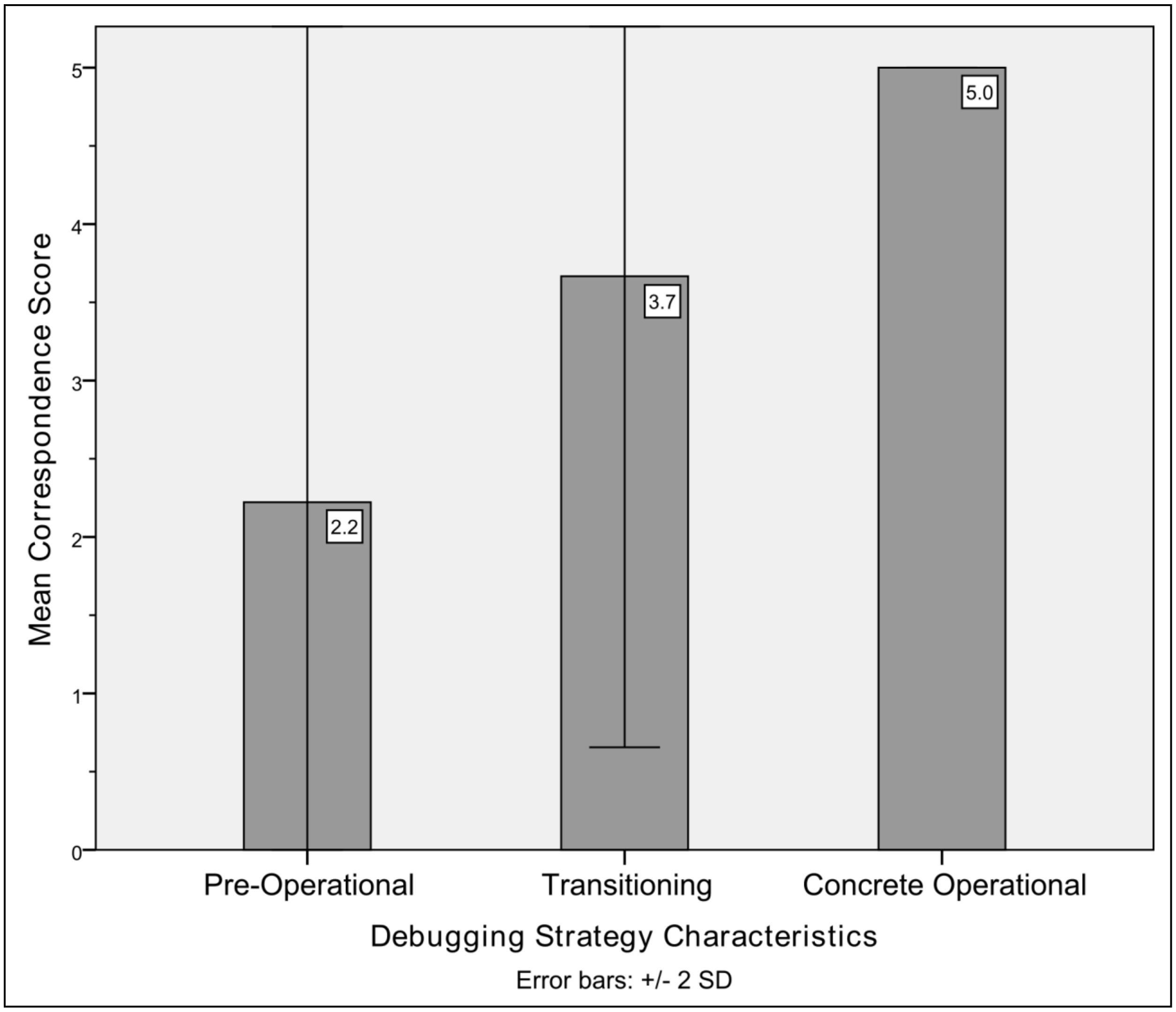


Figure C10. Mean correspondence score by cognitive developmental level of debugging strategy.

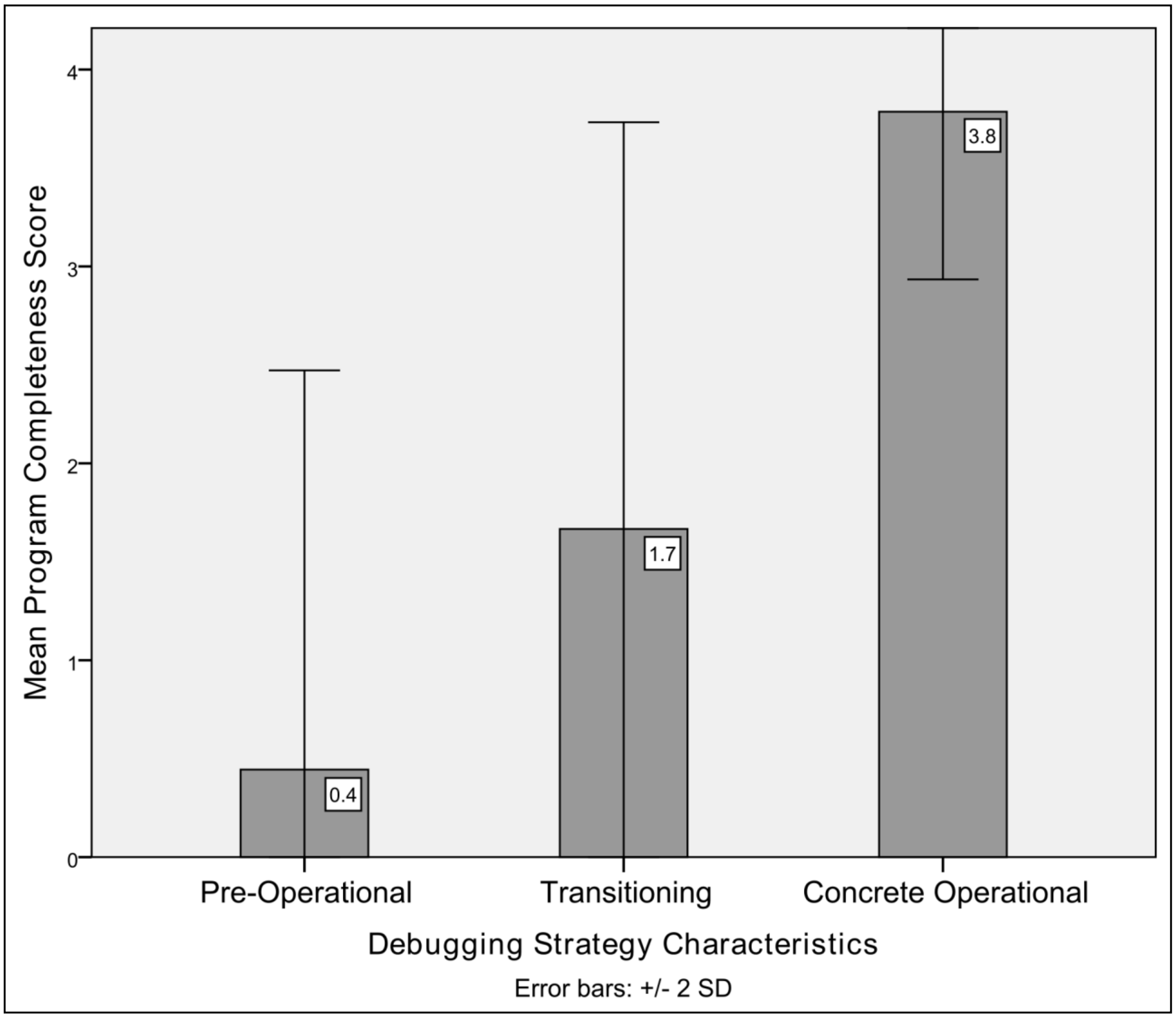


Figure C11. Mean program completeness by cognitive developmental level of debugging strategy.

Table C5

Sequencing Scores by Developmental Level

Developmental Level	*n*	*M*	*SD*
Sequencing Pre-Assessment Scores			
Pre-Operational	7	7.14	1.07
Transitional	6	6.67	2.73
Concrete Operational	14	7.86	1.35
Total (F(2,24) = 1.17, p = .328)	27	7.41	1.69
Sequencing Pre/Post Score Delta			
Pre-Operational	6	0.00	2.60
Transitional	5	2.60	2.51
Concrete Operational	13	1.15	1.77
Total (F(2,21) = 2.28, p = .127)	24	1.17	2.12

Table C6

Programming Achievement by Sequencing Scores

Programming Measure	Sequencing Measure	*F*	*p*
Correspondence	Pre-assessment	F(1,25) = 0.04	.839
	Post-assessment	F(1,23) = 2.10	.161
	Delta	F(1,23) = 1.45	.241
Program completeness	Pre-assessment	F(1,26) = 0.66	.425
	Post-assessment	F(1,25) = 0.30	.589
	Delta	F(1,23) = 0.06	.812

Table C7

Child and Parent Background and Correspondence Achievement

	R^2	*F*	*p*	*Beta*	*t*	*p*
Demographic Factors						
Age	.12	(1,26)=3.54	.07	0.35	(27)=1.88	.07
Grade	-	-	-	-	(27)=0.66	.52
Gender	-	-	-	-	(27)=0.34	.73
Urban/suburban home		-	-	-	(25)=0.80	.43
Child Experience Factors						
Computer use at home	-	-	-	-	(20)=3.68	.001*
Computer skill level	-	(2,25)=1.88		-	-	.17
Robotics exposure	-	(2,26)=1.69	-	-	-	.20
Parent Experience Factors						
Highest level of education	.00	(1,26)=0.05	.83	-0.04	(27)=-0.22	.83
STEM degree	-	-	-	-	(25)=-0.64	.53
STEM job	-	-	-	-	(26)= 0.21	.84
Programming experience	-	-	-	-	(26)=0.02	.99
Robotics experience	-	-	-	-	(26)=-0.40	.70

Notes: *Denotes statistical significance at the p <= .001 level. The finding is an artifact of all the children in the highest developmental category being non-computer-users at home.

Table C8

Child and Parent Background and Program Completeness Achievement

	R^2	*F*	*p*	*Beta*	*t*	*p*
Demographic Factors						
Age	.16	(1,26)=4.91	.04	0.40	(27)=2.22	.04*[1]
Grade	-	-	-	-	(27)=1.40	.17
Gender	-	-	-	-	(27)=0.31	.76
Urban/suburban home		- -		-	(25)=1.74	.09
Child Experience Factors						
Computer use at home	-	-	-	-	(26)=4.28	<.001*[2]
Computer skill level	-	(2,25)=2.63	-	-	-	.09
Robotics exposure	-	(2,26)=1.48	-	-	-	.25
Parent Experience Factors						
Highest level of education	.02	(1,26)=0.44	.51	-0.13	(27)=-0.66	.51
STEM degree	-	-	-	-	(25)=-0.62	.54
STEM job	-	-	-	-	(26)=-0.47	.64
Programming experience	-	-	-	-	(26)=-0.88	.39
Robotics experience	-	-	-	-	(26)=-0.60	.56

Notes: *Denotes statistical significance at the p <= .001 level.

[1]Age does not predict achievement after taking developmental level into account.

[2]The finding is an artifact of all the children in the highest developmental category being non-computer-users at home.

Appendix D – Robotics and Programming Figures

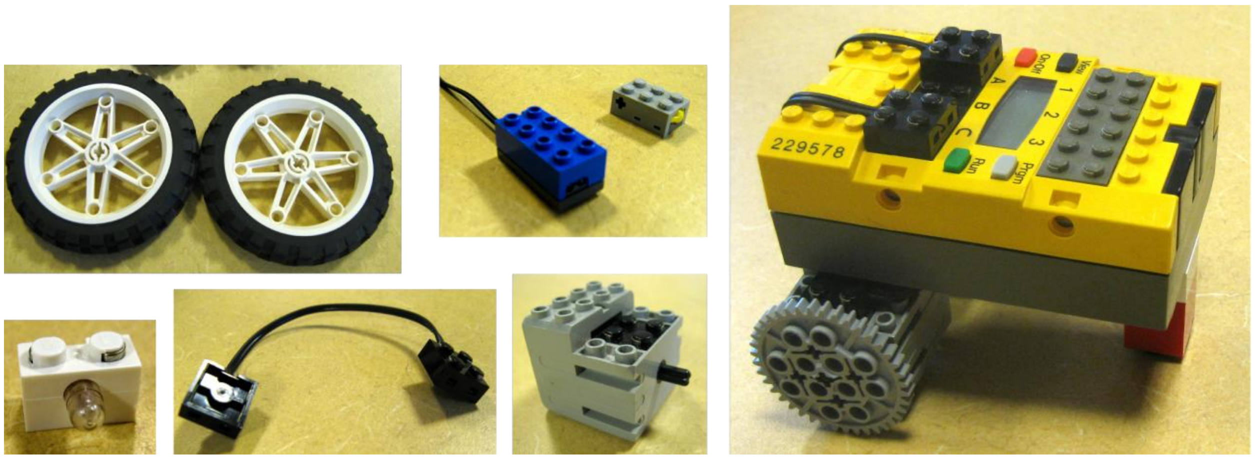


Figure D1. The robotic car components and a complete RCX™ robotic vehicle. Children built a robot from: (clockwise from top left) wheels, sensors (used in the third activity), a complete RCX™ robotic vehicle, a motor, a wire, and a light bulb. Additional LEGO® bricks and a rounded slider made the front 'leg;' gears were used for the wheels.

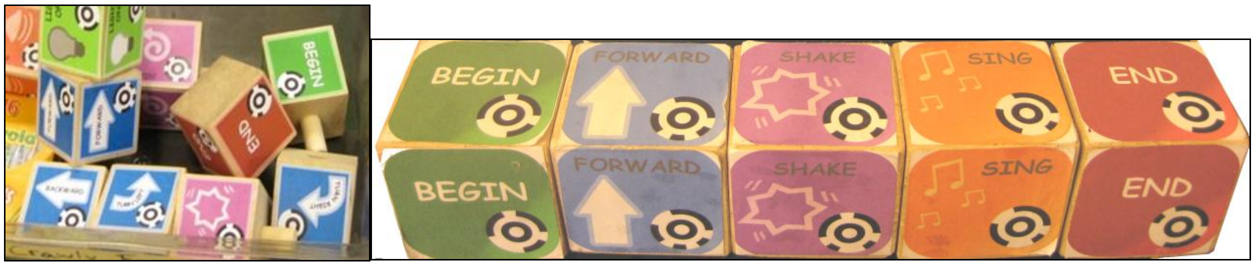


Figure D2. CHERP’s tangible interface. It consists of interconnecting wooden blocks.

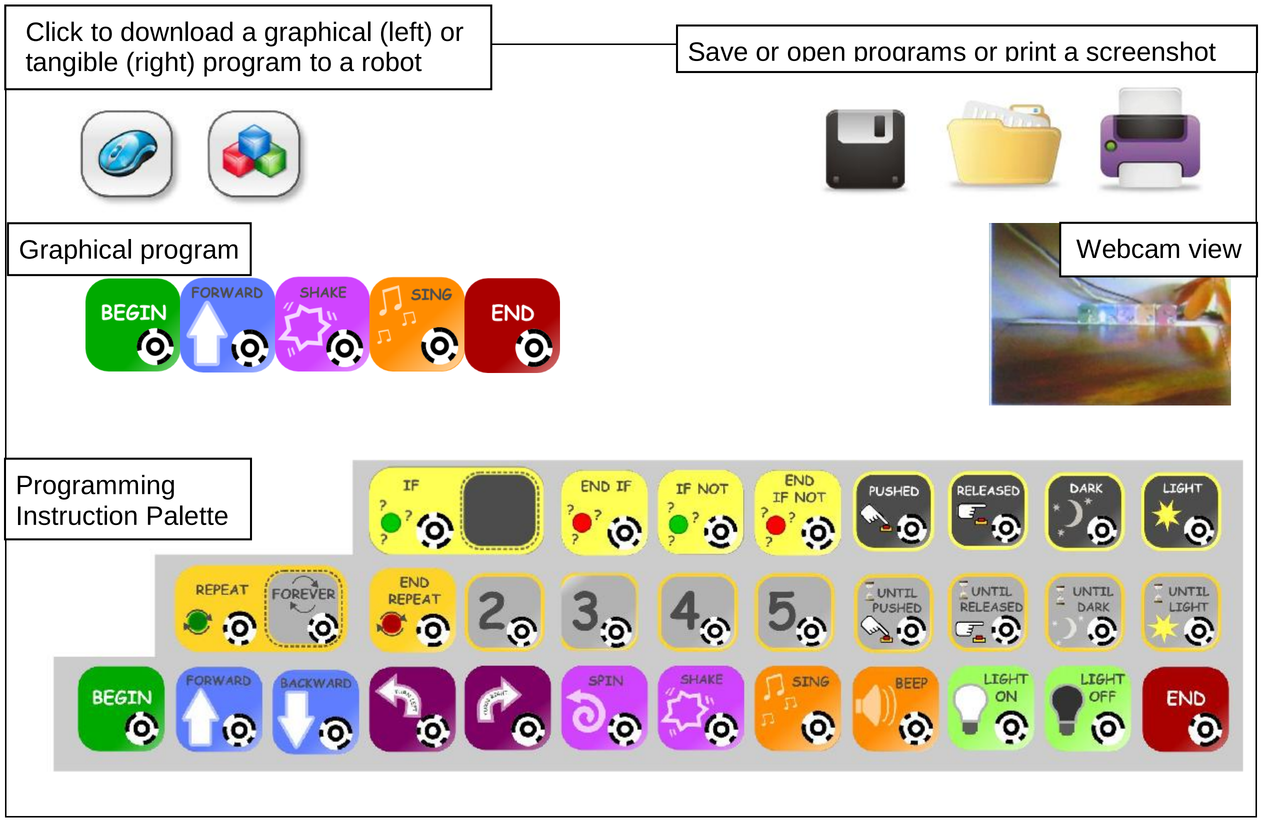


Figure D3. On-screen features of the CHERP interface.

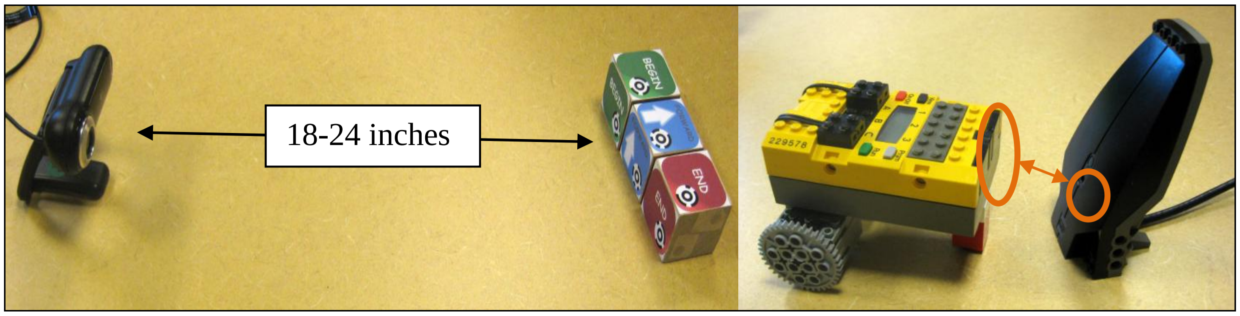


Figure D4. CHERP’s communication technology. CHERP uses a webcam to image tangible programs and a LEGO infrared transmitter ("tower") to communicate programs from the blocks to the robots through a laptop.

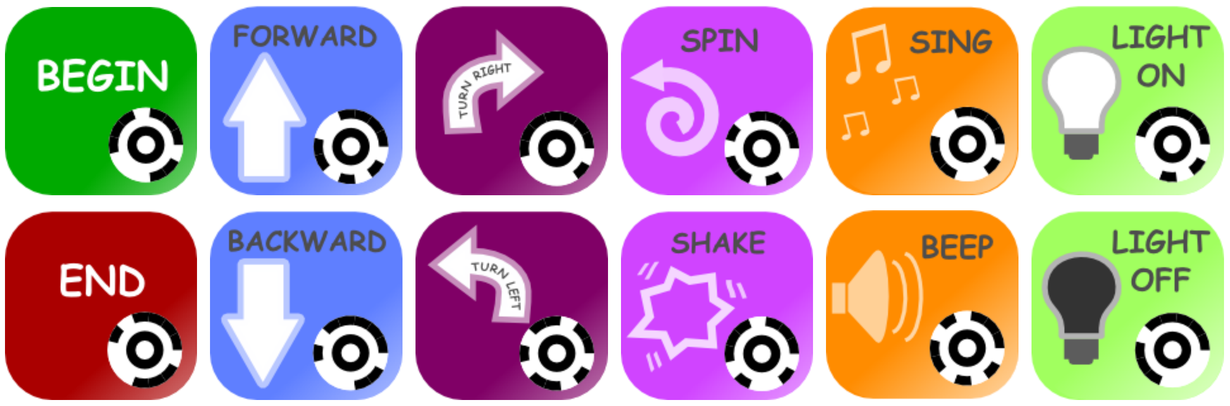


Figure D5. CHERP programming instructions available during the Hokey-Pokey challenge.

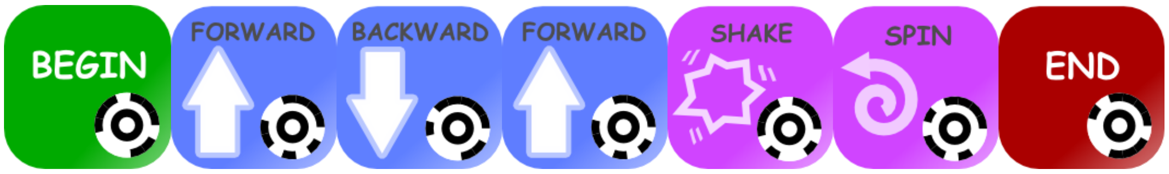
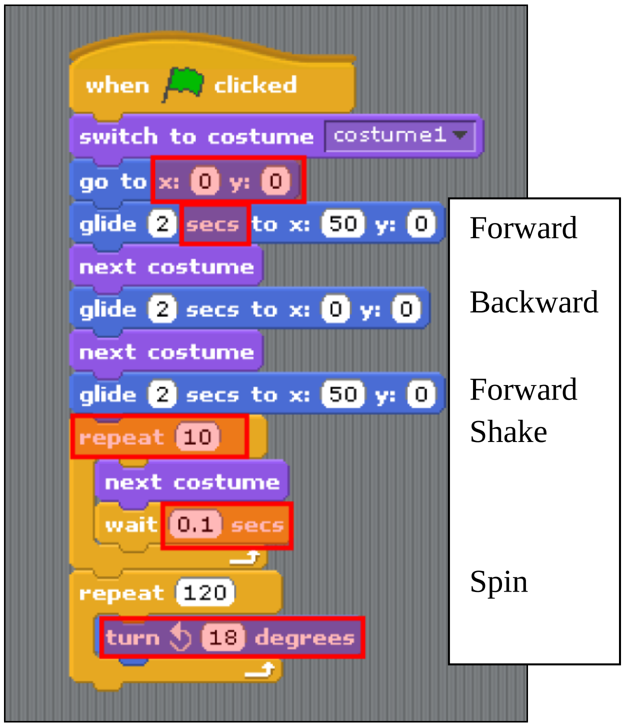


Figure D6. The expected solution program for the Hokey-Pokey challenge.



```
when [flag] clicked
switch to costume costume1
go to x: 0 y: 0
glide 2 secs to x: 50 y: 0
next costume
glide 2 secs to x: 0 y: 0
next costume
glide 2 secs to x: 50 y: 0
repeat 10
    next costume
    wait 0.1 secs
repeat 120
    turn 18 degrees
```

Figure D7. A Hokey-Pokey program made in the Scratch programming language. This program makes the Scratch cat dance the Hokey-Pokey. The CHERP instruction for each line of the song is marked where it begins for comparison. The "next costume" instruction makes the cat look like it has taken a step. Highlighted challenges, besides building up actions from smaller components include: using the Cartesian coordinate system, including possible negative values, control flow structures, decimals, and degrees.

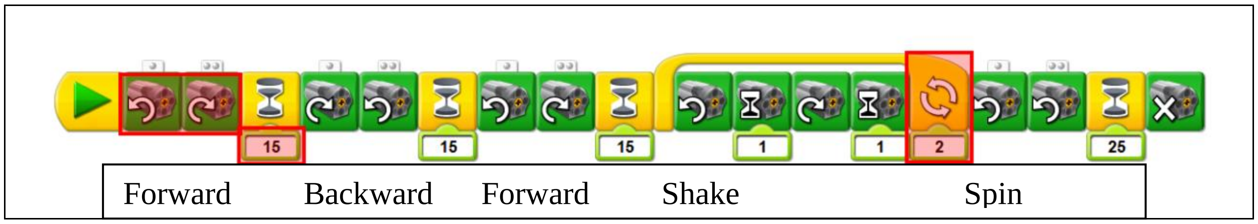


Figure D8. A Hokey-Pokey program made in the WeDo™ programming language. This program makes a robotic car dance the Hokey-Pokey. The CHERP instruction for each line of the song is marked where it begins for comparison. Highlighted challenges, besides building up actions from smaller components include: coordinating multiple motors, counting in decimals (15 represents 15 tenths of a second here), and using control flow structures.

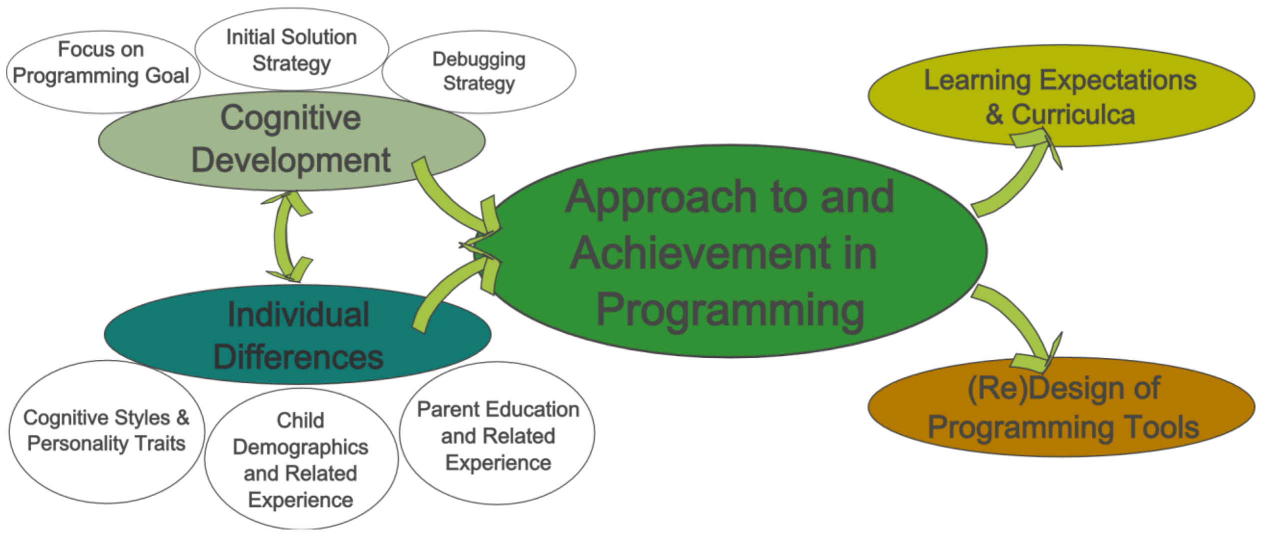

Focus on Programming Goal
Initial Solution Strategy
Debugging Strategy
Cognitive Development
Approach to and Achievement in Programming
Learning Expectations & Curriculca
Individual Differences
(Re)Design of Programming Tools
Cognitive Styles & Personality Traits
Child Demographics and Related Experience
Parent Education and Related Experience

Figure D9. Hypothesized relationships among the predictor and outcome variables (left and center) and the areas impacted by implications of the results (right).