

Young Children Programming with Electronic Blocks

Peta Wyeth

School of Information Technology and Electrical Engineering

University of Queensland

Brisbane Queensland 4072

Australia

Tel: +61 7 33654190

Email: peta@itee.uq.edu.au

Submitted to *The Journal of the Learning Sciences*

February 2006

Abstract

Electronic Blocks are a new programming environment, designed specifically for children aged between three and eight years. They are physical, stackable blocks that include sensor blocks, action blocks and logic blocks. The Electronic Blocks can be connected to create a wide variety of dynamic structures. By connecting these blocks children can program structures that interact with each other and the environment. Electronic blocks have been designed to provide young children with opportunities to program and observe dynamic behaviour without having to acquire complex symbolic notation systems. Evaluation of the Electronic Blocks shows that the blocks' ease of use and power of engagement have created a compelling tool for the introduction of programming concepts in an early childhood setting. Young children are able easily to build and debug "program" structures using the Electronic Blocks. The syntactic and semantic problems that confront users of conventional programming languages have been reduced allowing children to independently create Electronic Block programs to achieve specific outcomes.

Introduction

This article describes the development of a new educational resource that utilises the power of computational technology, while at the same time meeting the unique requirements of young children aged between three and eight years. This resource, labelled Electronic Blocks, has been designed to incorporate the dynamic and programmable properties of a computer, while honouring the nature of young children's interactions with learning materials. The Electronic Blocks are blocks – much like children's building blocks – with electronic circuits inside them. They have inputs and outputs and, when stacked, the output of one block controls the input of another. Children are able to determine the behaviour of their creations by the arrangement of the blocks. The Electronic Blocks aim to facilitate learning by enabling children to design, construct, explore and evaluate dynamic and programmable systems.

In 1987 Sheingold (1987) posed the question “What role could or should a microcomputer possibly play in a lively environment where children are actively working with materials and inventing their own worlds?” (p. 201). She was referring to computer use in the specialist area of early childhood education. Over fifteen years have passed and there remains the challenge for researchers and practitioners to develop technology that recognises the unique educational requirements of young children. Developmentally appropriate early childhood practice supports the theory that children construct much of their knowledge through active manipulation of the environment (McInerney & McInerney, 2002; Bredecamp & Copple, 1997; Lee, 1992; Beaty, 1984). It is not clear that the computer, often seen as the primary means of introducing technology into the classroom (Plowman & Stephen, 2005; Mackay, 1991; Pea & Sheingold, 1987), offers the best opportunities for such interactions. Of concern for early childhood educators is that a young child's ability to click a mouse and manipulate computer icons is probably far in advance of his or her logical comprehension of

these actions and symbols (Elkind, 1996). The principal view is that the computer, as a screen-based medium, is not as effective as other manipulatives in developing understanding and skills in the early years (Yelland, 1999). Children within this age group learn about themselves and their world through trying everything - through touching, testing and pulling things apart (Beaty, 1984). An unfamiliar object tends to set up a chain of exploration, familiarisation and eventual understanding: an "often-repeated sequence that will eventually lead to more mature conceptions of the properties (shape, texture, size) of the physical world" (Garvey, 1990). Pre-school children are familiar with learning which involves the active manipulation and transformation of real materials (Derman-Sparks, 1992). While preschool children are designing, constructing, comparing and evaluating, they have the opportunity to reflect upon and express their thinking. The challenge is to develop new technological resources which honour this tangible, constructive and explorative approach to learning.

The stance taken in this research is that, although the computer can have many functions within an early education setting, its power as a learning tool lies in its unique programmable and dynamic properties. It is these properties which make a computer different from other media with which young children interact. This is an idea that has been advocated by many in the past 20 years (from Papert, 1980 to Martin et al., 2000). Papert, in his landmark work *Mindstorms* (Papert, 1980), recognised that computer programming has great potential as a vehicle for the acquisition of useful cognitive skills such as problem solving and reflective thinking. The power of becoming a creator and the importance of construction within a learning process may be demonstrated using a musical analogy (Resnick et al., 1996). Through comparing the notion of a child learning to play the piano to that of learning to play the stereo, it is clear that learning to play the piano is a richer educational experience.

“By learning to play the piano, you can become a creator, not just a consumer, of music, expressing yourself musically in ever-more complex ways. As a result, you can

develop a much deeper relationship with (and deeper understanding of) music.” (Resnick et al., 1996, p. 5)

This analogy can be related back to educational computer use. Through learning to program, users become creators, not just consumers, of computing activities. There are many examples of programming environments that provide rich educational experiences for children – Logo (Papert, 1980), ToonTalk (Kahn, 1996) and Squeak (Kay, n.d.) are well known examples. While these environments seek to enable children to use the dynamic and programmable properties of the computer, all rely on abstractions to and / or from the computer screen and keyboard to the physical world. In other words, none have both physical inputs and outputs for the programming environment. A solution to providing suitable programming experiences for young children, who are only just acquiring the rudiments of notational systems, is a method of programming where both program input and output are tangible.

Electronic Blocks provide such a solution. Unlike the computer, both the input to and the output from the Electronic Blocks are physical, making them well suited to sensory-dependent children. Electronic Blocks have been developed to offer opportunities for young children to program and observe dynamic behaviour without having to acquire complex symbolic notation systems. They represent a paradigm shift, away from using the computer to teach programming, towards a physically embodied system that provides programming experiences that involve active manipulation of real materials. This programming interface has been designed to take into account the special needs of children under eight and has been based on early childhood development and learning research.

In previous work, the Electronic Blocks have been evaluated to determine the extent to which they provide a developmentally appropriate introduction to programming for very young children (Wyeth & Purchase, 2002). Evaluation results demonstrated that the

Electronic Blocks were successful in addressing concerns about young children's physical and cognitive readiness to use computers by making the traditional interface disappear, replacing it with a tangible interface that is both familiar and powerful. Preschoolers are able to create simple programs readily, and learn rapidly from the physical interactions provided by the blocks (Wyeth & Wyeth, 2001). However, these previous evaluations have not explored the potential of the Electronic Blocks as a resource for purposeful, task-oriented programming, but rather explored how well the blocks allowed ad hoc programming to occur.

The Electronic Blocks' ability to provide older children with meaningful programming experiences is explored in this paper. The aim of the study is to determine whether seven and eight year old children are able to access the blocks' programmable nature in a more deliberate manner, and whether children learn aspects of programming from performing specific tasks with the blocks. The programming analysis conducted for this Electronic Blocks evaluation was based on data from video observations of the children as they performed specific programming tasks. The evaluation looked at both high level and low level programming processes:

- how well children understood Electronic Block syntax and functionality;
- whether children were able to program with the blocks to achieve specific outcomes;
- the level of debugging which occurred;
- the extent to which planning as a part of programming activities; and
- how often children built and compared alternate solutions to programming tasks.

The evaluation demonstrated that children were easily understood the syntax and functionality of the Electronic Blocks and were therefore able to create solutions to specific programming tasks. As a resource for exploring programming concepts, the blocks have proved to be effective in encouraging children to create and re-use code structures, allowing

children to simply debug their creations, facilitating their involvement in the planning of new “systems”, and offering opportunities for children to evaluate alternate programming strategies.

Background

Resnick (1995, p. 31) asserts that the “best computational tools do not simply offer the same content in new clothing; rather, they aim to recast areas of knowledge, suggesting fundamentally new ways of thinking about the concepts in that domain, allowing learners to explore concepts that were previously inaccessible.” Computer programming as a field of learning offers such opportunities and the concept of computer programming as a rich educational experience is hardly new. The development and adaptation of computing programming languages for use by children can be considered from two perspectives:

- Simplifying the grammar of the programming languages
- Changing the mechanisms for program representation

Each of the following examples may be considered in terms of how they simplify programming syntax and semantics as well as how they provide representations which are suitable for children.

Using Computers to Introduce Programming to Young Children

The well-established Logo programming language (Papert, 1980) enabled children to write procedures to manipulate a “turtle” on a computer screen. Logo was developed as an educational resource which incorporated some of the essential elements of Piaget’s constructivist theory which holds that children are builders of their own intellectual structures and that intellectual development does not always need explicit teaching (Vaidya, 1984). Logo was being introduced in many classrooms in the mid-1980’s as a result of widespread expectations of gains in higher cognitive abilities such as reasoning and planning, although there was little clear evidence of a relationship between Logo experience and such gains

(Cumming and Thomason, 1996; Hughes and Macleod, 1986). Introduced partly as a reaction against the rigidity of drill and practice programs, Logo offered an alternative which was consistent with the constructivist view of learning. The constructivist approach takes the position that knowledge is neither a copy of the external world nor a reflection of pre-formed structure in the mind. Rather it is built up over time as the result of constructive action (Lee, 1992).

Logo, although designed specifically for educational purposes, has rarely been used with children under the age of six or seven. This lack of examples is not surprising as many of the ideas involved in Logo, such as that of variables, or recursion, would be difficult for young children to grasp. Even the basic commands of Turtle Graphics, such as 'FORWARD 100', or 'RIGHT 90', involve not only relatively large numbers but also ideas about angle which young children may not yet have encountered (Hughes and Macleod, 1986). For young children, the ideas about instructions and sequence which form the core of programming are not necessarily simple. Given that preschoolers are only just acquiring the rudiments of notational systems it may be difficult for them to cope with the symbolic systems required to successfully program a computer. Such symbolic systems are too complex for young children to fully understand (Sheingold, 1987).

The Craigmillar Logo Project (Hughes and Macleod, 1986) and the use of Logo at the Drexel Early Childhood Centre (Vaidya, 1984) are two examples of Logo being used with young children. In both cases Logo was simplified to accommodate young children's developmental levels. Researcher from the Drexel Early Childhood Centre (DECC) found that young children using standard Logo had difficulties using even primitive Logo commands (Vaidya, 1984). In response to this difficulty, researchers simplified commands by constructing one-key Logo programs, which the preschoolers could then use (for example, FD 10 became F). Children involved in the Craigmillar Logo Project initially used the

Edinburgh Turtle – a floor-crawling robot – which was controlled through instructions from the computer. Towards the end of the project children did progress to controlling a simulated Turtle on the computer screen.

Both research projects found value in using simplified programming languages. The DECC research found that through using one-key commands to move the turtle children were successful in creating stories and graphics (Vaidya, 1984). Research from the Craigmillar Lego project found that when using Logo, the children demonstrated a high level of absorption, collaborative problem solving and mathematical discussion (Hughes and Macleod, 1986). In addition, the children made statistically significant gains on a standard ability test. The researchers felt that these results demonstrated that Logo has an important role to play in the education of very young children.

In more recent years, a number of visual programming environments have been developed for use by children. These environments typically use direct manipulation of iconic programming elements. Examples include:

- Toon Talk, a video game-like environment which aimed, through animation, to assist children in understanding code execution (Kahn, 1996);
- KidSim (later called Stagecast), a ‘program by demonstration’ environment which allowed children to construct and modify simulations (Cypher & Smith, 1995; Smith et al., 1996);
- Leogo, which allowed children to become programmers through one of three paradigms: by direct manipulation using ‘programming by demonstration’, by clicking buttons and dragging sliders in an iconic language and by typing commands using a text-based language (Cockburn & Bryant, 1997); and
- Squeak, a highly interactive and constructive authoring environment that children can use to create objects and subsequently script their behaviour (Kay, n.d.).

These programming environments have primarily been designed for children over the age of eight. While evidence from studies of Toon Talk (Kahn, 1999), Cocoa (Gilmore et al., 1995) and Leogo (Cockburn & Bryant, 1997) suggests that children over the age of eight found these systems effective and appealing introductions to programming, there is no evidence to indicate that they are useful or usable for younger children. My Make Believe Castle developed by LCSJ (My Make Believe Castle, 1995), an iconic, animation-based environment which allows children aged four to seven to control the actions, mood and speech of castle inhabitants, is one of the few visual programming environments designed for young children. While all of these environments seek to enable children to use the dynamic and programmable properties of the computer, they rely on abstractions to and from the computer screen and keyboard to the physical world.

Young children and computer programming – A cautionary note.

Although there have been studies carried out in which young children appear to be programming computers with competence (Vaidya, 1984; Lawler, 1985; Lawler et al., 1986; Hughes and Macleod, 1986), Elkind (1996), a key figure in early childhood education issues this caution:

“Using a young child's computer competence to gauge his or her intellectual abilities can be very misleading. The risk is similar to that of assessing a young child's cognitive abilities from his or her use of grammar and syntax. We all have heard young children use grammar and syntax far ahead of their level of comprehension. They can use "if", "but" and "because" correctly without any real understanding of propositional and conditional thinking or of psychological causality. In the same way, I believe that children often can display a level of computer competence that masks a much lower level of cognitive understanding.” (p. 22)

Consequently, although there is little doubt that young children find it easy and interesting to give instructions to a computer that result in events occurring on the screen, there is a problem of what children understand. A young child's ability to click a mouse and manipulate computer icons is probably far in advance of logical comprehension of these actions and symbols (Elkind, 1996). There may be a disparity between the young children's skills with computers and their thought processes.

Tangible Programming Environments

In addition to the programming environments described above, researchers are currently exploring new technology which allows children to create computer programs using tangible components. These ideas have been instantiated in three ways: programming which results in a tangible output; programming which requires tangible input; and programming in which both input and output are tangible.

Producing Tangible Output.

The development of classroom technologies that enable children to learn from construction is the key idea which underpins much of the research in this area. Many researchers have identified that technology which supports children becoming involved in design projects provides rich opportunities for learning (Harel, 1991; Kafai, 1996; Lehrer, 1993; Resnick et al., 1999). Design activities are well documented in engaging learners as active participants in the learning process.

Following on from the development of Logo, were the “floor turtles” which could be used in conjunction with it (Martin et al., 2000). A pen attached to a robotic turtle created drawings on paper in response to Logo programs created by children. In the early 1990's LEGO/Logo (Resnick & Ocko, 1991) combined the LEGO Technic product with the Logo programming language. It was the first robotic construction kit ever made widely available (Martin et al., 2000). The Programmable Brick is a successor to this research. The

Programmable Brick is a tiny computer embedded inside a LEGO brick that children use to build systems that behave and respond to their environment (Resnick et al., 1998). Children included the Programmable Brick into their regular LEGO constructions and then wrote Logo computer programs to make their creations react and behave. While the output was tangible, input required the use of a desktop computer.

A construction kit called Cricket was developed as a successor to the Programmable Brick. Crickets are small Programmable Bricks that, in addition to connecting to motors and sensors, can communicate with each other via infrared light (Resnick et al., 1998). The communication ability of Crickets allows children to think about systems of communicating entities and explore the behaviours that arise from Cricket interactions. Like the Programmable Brick, Crickets are fully programmable with children being able to write and download computer programs into the Crickets from a desktop computer (Resnick et al., 2000).

Programming with Tangible Input.

AlgoBlock (Suzuki & Kato, 1995) allowed children to build programs by connecting electronic building blocks to form a sequence of program steps. Each block embodied a Logo primitive thereby providing children with a unique way to provide programming input. The output of AlgoBlock programs was displayed on a computer screen. While a light on each of the blocks was illuminated as its corresponding statement was executed, the blocks provided no meaningful tangible output which represented the result of code execution.

Tangible Input and Output.

The development of *curlybot* has occurred in parallel to the development of Crickets. *curlybot* is aimed at children in their early stages of development - ages four and up (Frei et al., 2000). It is an autonomous two-wheeled vehicle with embedded electronics that can record how it has been moved on any flat surface and then play back that motion accurately

and repeatedly. This is an example of ‘programming by demonstration’. *Curlybot* was developed with the view that it would allow children to develop intuitions for advanced mathematical and computational concepts, like differential geometry, through play away from a traditional computer (Frei et al., 2000). In preliminary studies conducted by the developers of *curlybot*, they found that children learned to use *curlybot* quickly.

Electronic Blocks Design

The development of the Electronic Blocks has taken place in response to concerns about young children's physical and cognitive readiness to use computers to create programs. In addressing Elkind's caution (1996) and acknowledging the view that the computer, as a screen-based medium, is not as effective as other manipulatives in developing understanding and skills in the early years (Yelland, 1999), the Electronic Blocks have been designed so both the method of programming and the outcome of the program are tangible. They are designed to provide developmentally appropriate programming experiences for sensory dependent young children.

Electronic Blocks differ from *curlybot* ((Frei et al., 2000) in that they embody a constructivist view of programming rather than use a ‘program by demonstration’ paradigm. The programming method embodied within the Electronic Blocks follows the principle “What I cannot create, I do not understand” (Gleick, 1992 in Resnick, 1994) and their design is based on the belief that one of the best ways to gain a deeper understanding of something is to create it. The Electronic Blocks incorporate the idea of allowing children to become engaged in powerful programming experiences whereby children can use simple elements to build complex systems that interact with each other and the environment in a wide variety of ways. The new technology is designed to allow children, through concrete interactions, to explore ideas about the emergence of complex behaviours, using simple, tangible components.

Physical Design

Electronic Blocks are physical building blocks. The Electronic Blocks have been made by placing electronics inside LEGO™ Duplo™ Primo™ blocks. The blocks have inputs and outputs and when connected, the output of one block controls the input of another. Electronic Blocks can be connected together to create a wide variety of dynamic structures which interact with the environment. Different behaviours may be observed by connecting two or more blocks together. There are three kinds of Electronic Blocks: sensor blocks, action blocks and logic blocks (Figure 1).



Figure 1: The complete Electronic Block family: the three sensor blocks are to the left, the four logic blocks are in the centre, and the action blocks to the right.

There are three sensor Electronic Blocks: a *seeing* block, a *hearing* block and a *touch* block. These blocks are capable of detecting light, sound and touch, respectively. Action blocks produce some kind of physical output. The *light* block lights a bright incandescent bulb, the *sound* block plays a simple children's melody, and the *movement* block is a four wheel car that drives in a straight line.

By placing a *hearing* block on a *light* block children can produce a structure which shines whenever they talk. By then removing the hearing block and placing it on a movement block they create a vehicle that moves when they talk. These are examples of simple sensor-action combinations. Given a set of three sensor blocks and three action blocks, there are a

total of nine such combinations. Figure 2 provides an example of a sensor-action combination.



Figure 2: A *touch* block attached to a *light* block will cause the light to turn on whenever the sensor plate is touched.

Logic blocks have an intermediary role. When placed between a sensor block and an action block they have the ability to alter the expected action by altering the signal passed between blocks. Logic blocks provide users with the capability to:

- produce an action if a particular stimulus is not received (*not* block),
- toggle the input so that in the first instance the stimulus from the environment will “turn the action on” and the second instance of the stimulus will “turn the action off” (*toggle* block),
- stretch a short signal so that the action will stay on for two seconds after the stimulus stops (*delay* block), and
- only produce an action if input signals are received simultaneously through both inputs (*and* block).

The addition of logic blocks to the set of Electronic Blocks opens up a wide variety of additional construction opportunities. For example Figure 3 demonstrates what happens to a *light* block when it has a *not* block and a *hearing* block stacked on top of it: the light comes

on when all is quiet. Logic blocks add to the complexity and variety of structures that may be created.

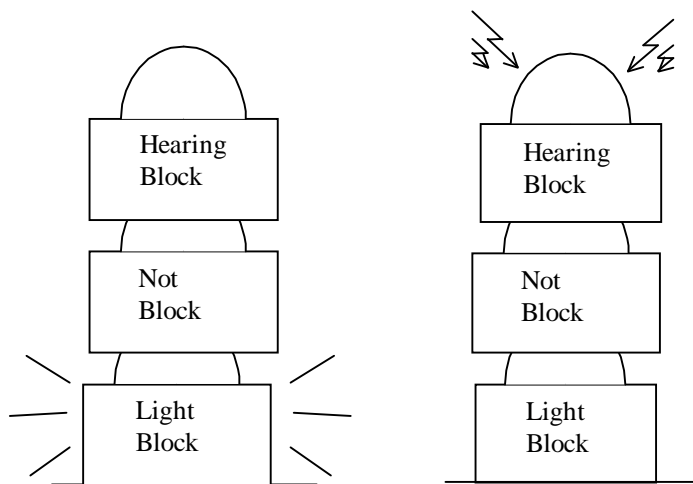


Figure 3: A *not* block placed between a *hearing* block and a *light* block performs a logical NOT operation causing the light to go on whenever there is no sound.

An additional aspect of Electronic Blocks is their ability to interact not only with the environment but also with each other (Figure 4): a *light* block in one construction can trigger a light sensor in another construction. To add further complexity logic blocks could be included in these interacting stacks. The construction opportunities offered by the full set of Electronic Blocks are wide and varied.



Figure 4: A remote control car demonstrates interaction between block stacks. The *touch* block on the *light* block forms the remote control for the car formed by placing a *seeing* block on a *movement* block.

Identifying the Blocks.

All sensor blocks are yellow. Readily understandable icons identify the different functions of the sensing blocks: for example, an eye for a seeing block. The functionality of the action blocks is somewhat self-evident from the physical structure of the blocks. The sound and light blocks are also adorned with explanatory icons. Each different logic block type has distinctive icons and colours to assist their identification. It is difficult to choose meaningful icons for these blocks. What icon explains “and” to a preschooler? The icons were chosen to have readily understood adult meanings: for example, & for “and”.

Electronic Blocks Evaluation

Twelve children aged between 7 and 8 years participated in an Electronic Blocks evaluation. They were required to complete twenty tasks using the Electronic Blocks.

Examples of the types of tasks children were asked to complete included:

- Make a voice activated torch.
- Make a car that moves when you speak or when you touch it.

This study’s primary focus was on the extent to which children were able to effectively use and understand the Electronic Blocks and how well this understanding translated into meaningful programming activities such as creating working ‘code’ and debugging.

Study Procedure

The main aim of the evaluation was to determine the types of programming with which participants became involved while using the Electronic Blocks. In order to study this process-oriented activity it was necessary for the evaluation to be problem oriented. By observing children as they completed problem solving tasks, the evaluation was able to assess the extent to which children are able to apply programming skills while creating program structures with Electronic Blocks.

Twelve children, aged between seven and eight years, participated in this study. The average age of the students was eight years and two months – the youngest student was seven years and five months old, and the oldest student eight years and nine months old. They were all grade three students at a local primary school. The school had three classes of grade three students and four children were chosen from each class. The teachers were asked to choose average students who worked well together and to group them as pairs. Each class supplied two girls (paired) and two boys (paired). Although this was not a prerequisite of the study – the study asked for two pairs from each class – these were the pairings the teachers chose to supply.

At each session Electronic Blocks were set up in a break-out room usually used for remedial work with children. A video camera and audio equipment were set up to record the children interacting with the blocks.

The evaluation involved six sessions that took place over a two week period. The students involved in the study were required in the first instance to work as one large group to participate in an Electronic Block lesson (day 1). This session was intended as a familiarization and learning period. They then participated for one session as three groups of four students in an open free play environment (day 2). After the children had been provided with the opportunity to familiarize themselves with the Electronic Blocks – through the Electronic Blocks lesson and a free play session – they were then required to complete twenty tasks over four 20-minute sessions (days 3 to 6).

The 20 tasks were divided into five categories. These categories reflect the differences in complexity of Electronic Block solution stacks:

- *Simple sensor-action constructions.* These are tasks that needed only sensor blocks and action blocks in the solution. Task 3 – “Create a radio that plays music whenever you touch it” – is an example of such a task.

- *Interacting block stacks.* These tasks required students to create simple sensor-action stacks that interacted with each other in some way. Task 4 – “Build a light that turns on whenever it sees another light” – is an example of such a task.
- *Simple logic stacks.* These are solutions to tasks that required the used of a single logic block. Task 6 – “Make a radio with an on/off switch” – is an example of a simple logic stack.
- *Logic combination stacks.* Some tasks required the construction of stacks which used combinations of logic blocks to achieve a suitable solution. Task 8 – “Make a car that only goes when it is quiet and when you touch it” – is an example of a logic combination stack.
- *Interacting blocks stacks with logic.* The solutions to tasks required constructions that interacted and used some logic blocks as well. Task 9 – “Create this story: A car is out of control. It roams around bumping into everything. Clever scientists discover that the car stops when they shine a bright light at it” – is an example of such a task.

One the final day of the evaluation, the children were asked to complete a questionnaire which focussed on both their perceived understanding of the Electronic Blocks and their feelings towards using the blocks. The children also participated in a ten minute demonstration for the rest of their classmates. These demonstrations took place in the children’s grade 3 classroom. Each of the four children from each of the grade 3 classes prepared an Electronic Block structure to demonstrate to their classmates.


Evaluation Methodology.

The evaluation addressed the issue of whether children could access the blocks’ programmable nature, and whether children learned aspects of programming from performing specific tasks with the blocks. Programming skills were analysed across the range of task

types using a programming skills checklist. The checklist has been based on a hierarchy of programming skills (see Table 1) developed by Oakley and McDougall (1997).

Table 1: Oakley and McDougall (1997) divide their six categories of programming skills into high order skills such as planning the programming process and exploring alternate strategies, and low order skills, including knowledge of syntax and coding simple structures.

Programming Skills

Evaluation of Alternative Strategies	
Complex Planning	
Debugging	
Use of Simple Structures	
Coding Simple Structures	
Knowledge of Syntax	
	Highest order
	Lowest order

The programming skills checklist checks for the following knowledge, skills and processes:

- Knowledge of Electronic Block syntax and functionality – how well children understand the potential role of a specific Electronic Blocks within a stack;
- Ability to create structures to achieve the desired results – children’s ability to connect blocks in a specific way to produce a particular behaviour;
- Understanding of how to reuse simple structures – children’s knowledge of how to reuse simple block structures from one task in another task;
- Testing and debugging skills – children’s ability to verify the validity of their solution through a process of identifying an error and correcting the error to achieve the solution;

- Planning of solutions – how well children are able to create and discuss plans for solving task problems; and
- Use and evaluation of alternatives – children’s ability to identify and implement alternate task solutions.

Based on the video data, a programming skills checklist was completed for each task, for each of the children participating. The checklist indicated which skills were used in the process of performing each task, and the degree of competence displayed with each skill. Specific case studies, which complement the checklist analysis, provide rich snapshots that help explain the data in a qualitative sense. These case studies are included as vignettes to highlight specific points. The analysed data and the case studies provide the evaluation information required to explore the extent to which the children were involved in programming activities.

Evaluation Observations

Each group of tasks has been analysed based on the total number of solution attempts for those particular tasks. A solution attempt is defined as a single child becoming involved in a construction process to work towards a solution for a specific task. It was not necessary for the child to successfully complete the task for the attempt to be considered in the data; however a child did have to start the process. Data from children who watched their partner or copied a solution was not included in the data analysis. If more than one solution was built for a single task, it was still considered a single solution attempt. Data is represented as percentages, as it is the clearest way to express the data extracted from the evaluation checklists. Given the number of students involved it does not imply that the evaluation of the Electronic Blocks is quantitative. While the evaluation is strictly qualitative (video analysis of a small number of students in a classroom setting), the percentages are a useful expression

of the checklist data. These percentages represent the number of occurrences of a particular behaviour across solution attempts.

Syntax and Functionality

When considering children's ability to understand the syntax versus function of individual Electronic Blocks, the understanding of syntax is the easier of these two aspects to evaluate. The physical affordances of the blocks enforce the simple syntax. The children were unable to create stacks where the function was undefined or ambiguous. This does not imply that all program stacks produced the desired or even useful behaviour, only that the blocks stacks were syntactically correct. Because of the use of physical affordances in an everyday play tool (blocks) the syntax required no explanation, and was immediately understood. The blocks have no buttons to press or rituals to perform to make them work; they simply embody their function.

Given that syntax is readily understood, this still leaves the issue of whether the children understood the blocks' functions. Within a hierarchy of programming processes, the ability to understand the Electronic Block functionality would be considered as the simplest as it primarily involved determining children's ability to identify which types of blocks should be used in a particular task solution. Understanding functionality does not imply that children have the ability to apply this knowledge to the creation of a suitable structure to meet the task requirements.

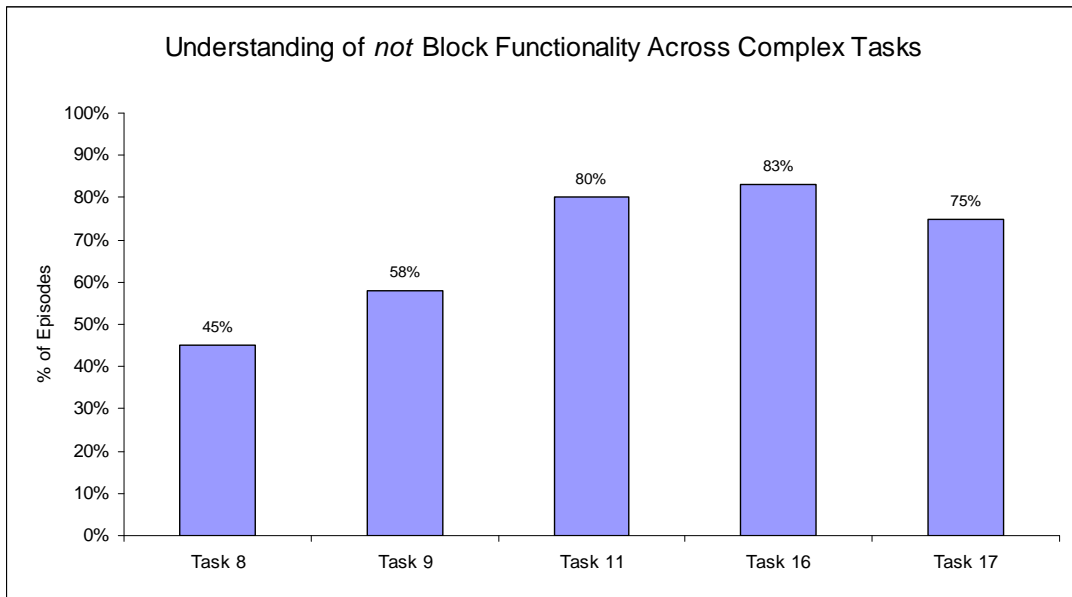
The evaluation showed that students understood the functionality of sensor blocks and action blocks. Children were generally successful at identifying suitable blocks to use in the construction tasks that only required sensor and action blocks – the simple sensor-action tasks and solutions using interaction block stacks. Children's good understanding of sensor and action blocks continued to be observed through the construction of more complex solutions (see Table 2). Children experienced varying degrees of success with respect to identifying the

logic blocks required to build solutions; analysis of the simple logic tasks shows that the children found some logic concepts more difficult than others. Consequently, the choice of logic block for the task was not always appropriate (see Table 2). This pattern continued with other tasks that required the use of logic blocks.

Table 2: Percentage of correct block choices across the twenty programming tasks.

Task type	Input blocks	Output blocks	Logic blocks
Sensor-action constructions	100%	100%	n/a
Interacting block stacks	98%	100%	n/a
Simple logic stacks	100%	100%	62%
Logic combination stacks	94%	100%	76%
Interacting block stacks with logic	96%	96%	68%

Analysis of the checklist data suggests that children found some logic concepts more difficult than others. Children were generally successful in identifying the need to use *delay* blocks (79%) and *toggle* blocks (70%) in Electronic Block solutions. However, children initially struggled to understand how to use the *not* block to reverse a behaviour. As the evaluation progressed children became more proficient at using the *not* block to allow the opposite behaviour to occur. There was evidence of improved proficiency in identifying *not* blocks in solution structures (see Graph 1). Similar improvements in understanding of function were observed while children were involved in constructing solutions with the *and* block.



Graph 1: Understanding of the *not* block increased with experience in complex tasks.

The children's responses on the feedback questionnaire indicated that they were confident in their own knowledge of Electronic Block functionality. All children indicated that they understood the functionality of all the blocks. The classroom demonstrations which occurred on the final day of the evaluation demonstrated that the children were able to verbalise to their classmate the way in which the Electronic Blocks worked. They were able to describe the functionality of the sensor and action blocks and detail how the logic blocks affected the behaviours of the action blocks. The children demonstrated Electronic Block structures from the five categories of structure explored during the evaluation and these demonstrations, on the whole, were well constructed and well thought-out. One child had difficulty while attempted to describe a very complex structure to his classmates. Lucy's¹ demonstration (see Vignette 1) is an example of a child being able to describe in a competent manner the workings of Electronic Block structures.

¹ All names of children have been changed to protect children's privacy.

Lucy built a music activated remote control car. One of the stacks consisted of a *hearing* block on a *movement* block. Another stack contained a *touch* block and *toggle* block and a *music* block.

$$\frac{H}{C} \leftarrow \frac{T_o}{M}$$

“See I used the music to make it [the *movement* block] go ... and if you take it away [pointing to the *hearing* block] it will stop because it can’t hear anything. It has to hear something to go. [When asked how the *hearing* block is activated] I put a *touch* block on [the *music* block] but then instead of touching it all the time I used an on/off block so it would stay on all the time instead of keeping my fingers on it the whole time [showing an example of the music turning on and off with her touch].”

Vignette 1: Lucy’s Electronic Block demonstration to her teacher and classmates.²

Achieving a Desired Solution

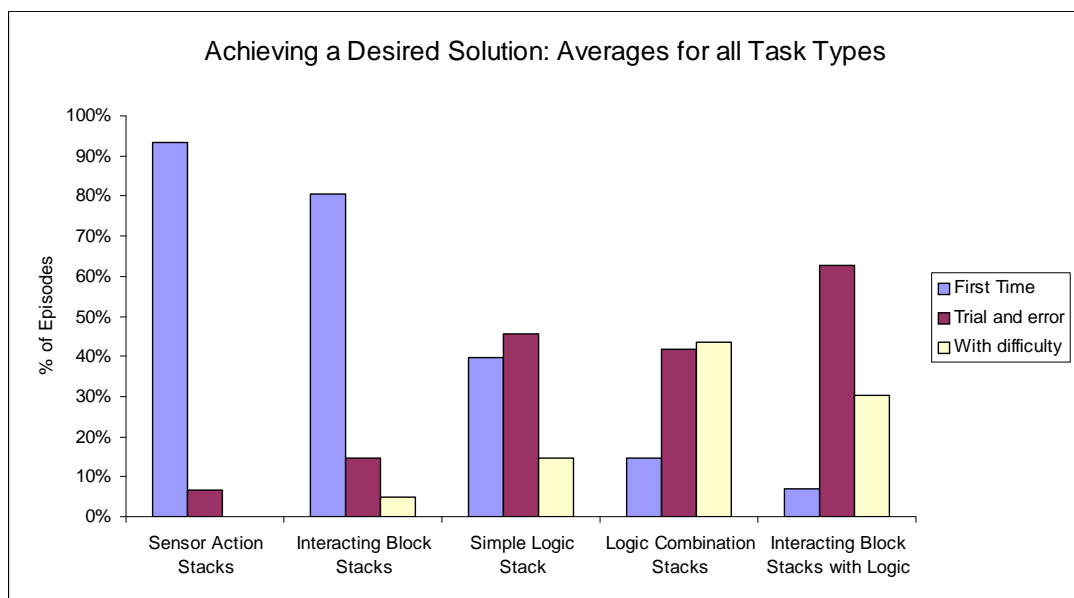
The programming process of achieving a desired solution is assessed in the evaluation by recording the children’s ability to use suitable Electronic Blocks and connect them in such a way as to achieve the behaviour described in a particular task. It requires not only a functional understanding of what each block does, but the application of this knowledge to produce a particular behaviour. The building of such a solution also requires an appreciation of how signals are passed between blocks.

In the analysis, children’s ability to achieve a desired solution is divided into three categories indicating whether the participant (a) was able to achieve a solution on the first

² Block Construction Key: Hearing Block (H), Seeing Block (S), Touch Block (T), Movement/Car Block (C), Light Block (L), Sound/Music Block (M), Not Block (N), Toggle Block (T_o), And Block (A), Delay Block (D). Stacking order is represented by the physical layout. Interacting stacks are represented with arrows.

attempt, (b) used trial and error in the construction of the solution, or (c) only achieved the solution with great difficulty. The phrase ‘trial and error’ here is used to indicate that the children needed more than one attempt at building a correct solution. However, it is not necessarily an indication of the process that they used to move to a correct solution. Some children incrementally achieved a solution using an undirected process while others used more methodical processes. A child was said to ‘achieve a solution with great difficulty’ with a task if they were unable to complete the task, or required help from the researcher to complete it.

Across all twenty tasks there was strong evidence that children were able to achieve a desired task solution. Children created appropriate program stacks on their first attempt on 47% of occasions. They used trial and error to create correctly working program stacks on 35% of occasions, and had difficulty creating a working solution on 18% of occasions. Graph 2 outlines the extent to which children achieved the desired results across all task types.



Graph 2: As the complexity of the tasks increased, the number of first time successes decreased with a corresponding increase in experimentation and difficulty.

Sensor-action tasks provided children with good opportunities to construct appropriate solutions and they experienced very little difficulty in doing so. A majority of children found

a solution to these problems on the first attempt. Construction during the creation of solutions to interacting block stack tasks had the same properties. They were an effective means by which children were able to construct structures to achieve a desired solution. With the introduction of logic blocks, the ability of the children to find a successful solution on the first attempt decreased. Solutions that needed the inclusion of a logic block were more likely to be found using trial and error.

The two most complex types of tasks – tasks that required solutions to include logic block combinations or interacting stacks with logic blocks – had many similarities with respect to the children’s ability to achieve a desired solution. Generally, children found these tasks more difficult and as a result there were a greater number of children taking opportunities to work towards a solution incrementally. A significant number of children found these tasks difficult to complete, with the children finding logic combination stack tasks the most difficult to complete successfully. For example, one of the tasks required that students create a car that only went when it was quiet and when they touched it. The children found this task difficult; only seven students from the twelve who attempted the problem used the *and* block correctly during construction and six students struggled with the idea of “quiet”. Only, five children readily understood the need for a *not* block in their Electronic Block stack.

While sensor-action and interacting block stack tasks provided children with good opportunities to construct appropriate Electronic Block solutions easily, they found the other task categories more challenging. The reasons for this are twofold. The first difficulty relates to children’s knowledge of Electronic Block functionality. This knowledge of functionality forms the basis for the successful creation of a structure to meet a particular goal. Consequently, understanding of Electronic Block functionality directly impacts on a child’s ability to create solutions to tasks. On tasks where children failed to translate a particular task

requirement to a specific Electronic Block function, children struggled to create the desired solution for a task. In parallel with the findings related to knowledge of functionality, children exhibited improved abilities to achieve a desired solution when using previously encountered logic concepts (such as the *and* concept).

The second difficulty experienced by children when constructing solutions to complex tasks was related to an understanding of Electronic Block signals. While children were good at identifying the correct blocks to use in some complex tasks, they struggled with connecting them in the correct order. For example, despite identifying the correct blocks many children struggled to successful build working solutions to task 12 (33%) and task 15 (80%). They could not work out how to stack the blocks correctly to achieve the desired solutions. There were also examples of such difficulties in more simple tasks that required the use of logic blocks. Vignette 2 of Matt and James completing task 6 – making a radio with an on/off switch – demonstrates this difficulty.

Researcher – “Can you make a radio with an on/off switch?”

Matt – “OK where’s the on/off switch (looking around) ... the radio (finds the *sound* block and picks it up) ... with an on/off (looking around and then finds the *toggle* block) on/off!” He places the *toggle* block on the *sound* block. At the same time James picked up a *sound* block – he then stops, happy to watch Matt build the solution.

$$\frac{T_o}{M}$$

Matt – “... and then I’ll get a touch.” He places the *touch* block beside the *toggle* block and touches it. The music only plays while he is touching the *touch* block. He touches the *touch* block repeatedly. “What’s going wrong?”

$$\frac{T_o T}{M}$$

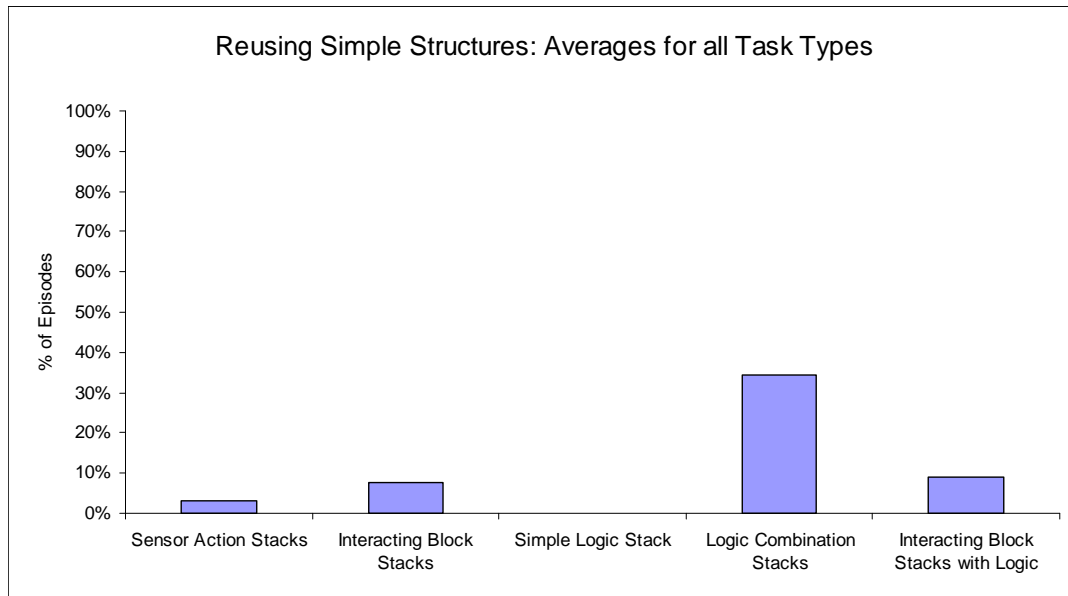
Researcher – “Well, what do you think is going wrong?”

<p>Matt asks “It’s right?”</p>	
<p>Researcher – “Is that on and off? It’s actually only working when you touch it.”</p>	
<p>Matt places his finger on the touch block to turn the music on. James says “Turn it off”. Matt takes his finger off and the music stops.</p>	
<p>Researcher – “Is that an on/off switch do you think?”</p>	
<p>Matt – “No.”</p>	
<p>James – “I know, I know ... put the on/off ...” He takes the <i>touch</i> block off and replaces it with a <i>not</i> block. They both look at the stack as it continually plays music.</p>	$\frac{T_o N}{M}$
<p>Matt – “I’ve got an idea”. He removes the <i>not</i> block. “Where’s um ... where’s um ... where’s um ... where’s um?” James has picked up an <i>and</i> block and says “an <i>and</i> block.” Matt says “Yeah, let’s try an <i>and</i> block with this.” He builds the structure with an <i>and</i> block and the <i>touch</i> block. When he touches the <i>touch</i> block nothing happens. He touches it a few times and then says “Oh yeah” as he begins pulling it apart. James says “It hasn’t got an <i>and</i> block on it.”</p>	$\frac{T_o T}{A M}$
<p>Matt returns to the original construction of a <i>toggle</i> and <i>touch</i> block on the <i>sound</i> block.</p>	$\frac{T_o T}{M}$
<p>Researcher says “Do you remember about the signals ... how they go from the top?” gesturing to show a flow down through the blocks.</p>	
<p>Matt exclaims “Ohhh!” and immediately picks up the <i>touch</i> block and puts it on top of the <i>toggle</i> block. He then touches it to test the validity of the solution</p>	$\frac{T}{T_o M}$

Vignette 2: James and Matt creating a solution to Task 6 – make a radio with an on/off switch.

Reusing Simple Structures

Reusing previously encountered structures is an important programming skill. It requires the ability to recognise that programs may contain components from a previously encountered program. These components may not be immediately recognisable and successful interpretation of a task is important in this process.



Graph 3: The reuse of simple structures was most evident in the logic combination stacks, and limited by the opportunities given the nature of the tasks in other task types.

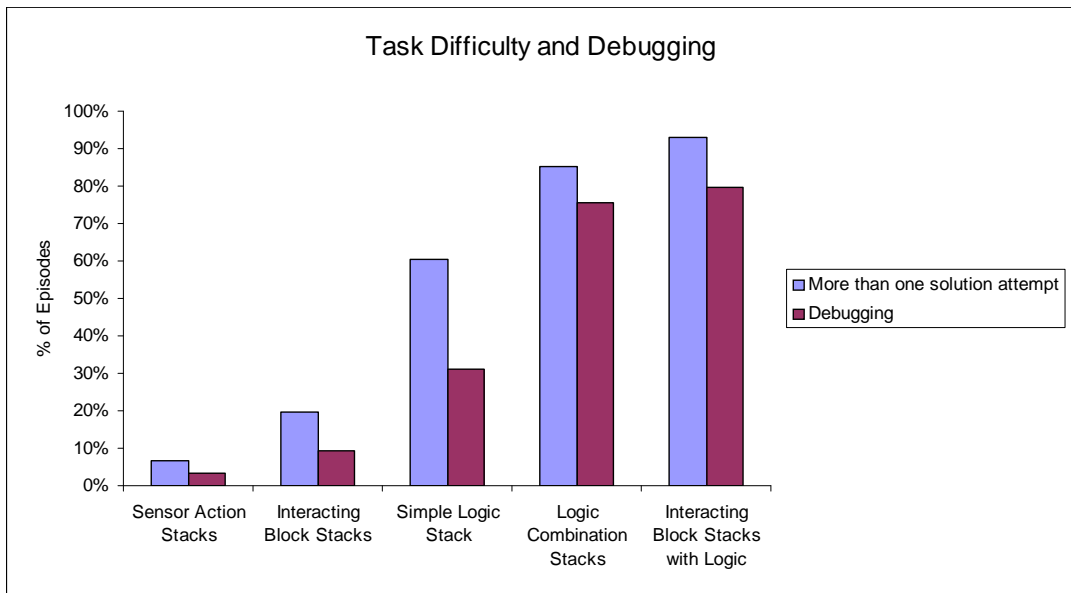
There were examples of children reusing Electronic Block concepts they had previously encountered (see Graph 3). There were only limited examples on this type of reuse when children were constructing simple stacks – mostly because of the limited opportunities for concept reuse in the assigned tasks. However, concept reuse was evident during the construction of interacting stacks, with the often required “remote control” structure being successfully repeated by many children. Children were also confident in creating “light activated” stacks. Concept reuse was also apparent when children were constructing solutions requiring combinations of logic blocks. Children were observed reusing concepts they had initially encountered while constructing simple logic stacks, for example, reusing the *and* block successfully on later attempts. Concepts were also reused from earlier logic

combination solutions; for example, the combination of blocks required to construct a system that reacts when it is “quiet”. Children used previously encountered concepts from the creation of interacting block stacks, when they were required to do so for more complex interacting tasks that also required the use of logic. The similarity of the two tasks that required two-way interaction allowed the concept to be effectively reused on the second occasion. The data demonstrated that five of the twelve children recognised the two-way interaction concept on the second viewing and as a result were better able to effectively construct an Electronic Block solution.

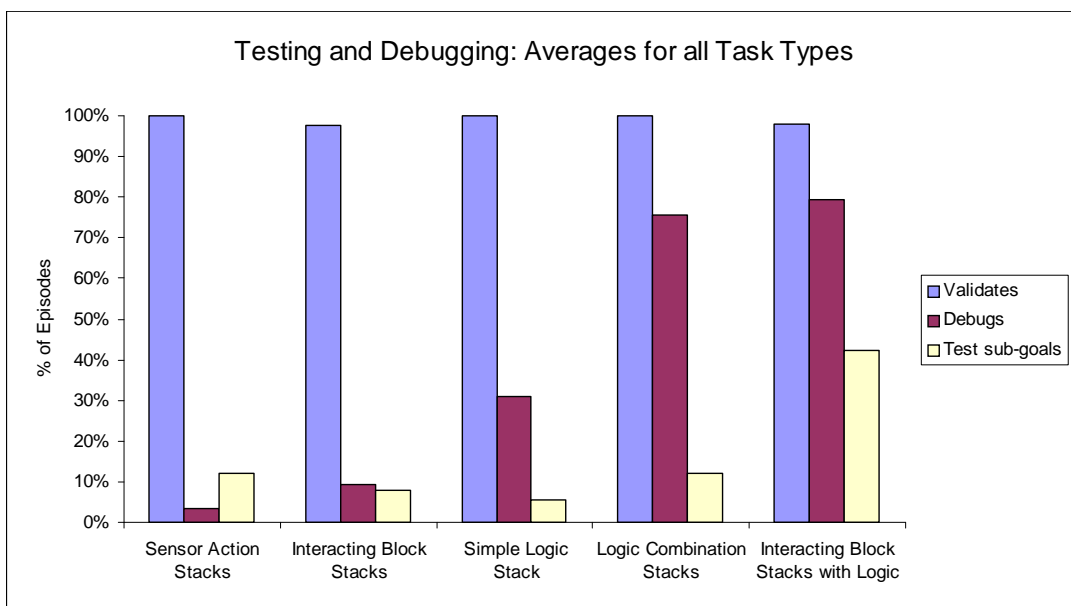
Testing and Debugging

Programming is a task that is rarely completed correctly on the first attempt. Programmers are required to test their programs. When the programs are found to be faulty, programmers need to track down the source of the error; they need to debug their programs. These processes can consume considerable amounts of a programmer’s time. Effective testing and debugging skills are essential to the process of creating successful programs.

There was strong evidence of children testing and debugging Electronic Block stacks across tasks. Across all tasks, there was an average of 40% of episodes where children were involved in debugging Electronic Block stacks. Graph 4 shows the number of children who required more than one attempt to achieve a working solution to these tasks as well as the percentage of debugging opportunities taken. Children had more opportunities to demonstrate debugging skills during the construction of solutions to the more complex tasks.



Graph 4: The percentage for multiple attempts at a solution in relation to the observed occurrences of program debugging.



Graph 5: The participants used testing and debugging processes in all task types. The use of debugging processes increased for more complex task types.

Children consistently attempted to validate their solutions to tasks (see Graph 5). The general difficulty that children exhibited in the validating of more complex stacks was the ability to establish criteria to judge the adequacy of solutions. In some instances children would think a solution was correct, when it still contained errors. Vignette 2 is a point in case. Initially, it appears that James has not established criteria to judge the adequacy of his

solution. He asks “What’s going wrong” and then when asked by the researcher what he thinks is wrong he seems unsure and asks “It’s right?” He needs some clarification from the researcher to determine that he has not created a radio with an on/off switch; that he has instead created a radio that plays when you touch it. The difference is subtle. Following this clarification, both students appear to have a better understanding of how to evaluate their solutions and identify errors. During the construction process, when they try the *and* block, they quickly determine that this solution is not correct with Matt dismantling the construction and James stating “It hasn’t got an *and* block on it.” Both students were involved in investigating the properties of a solution to verify that it fulfilled the task requirements. They did, however, initially have difficulty in identifying why a particular solution was invalid. After this initial problem was clarified both James and Matt were able to identify errors in the task solutions they created.

While these verification attempts were not always successful – primarily because children failed to identify existing errors – children did attempt to investigate the properties of their solutions to test solution validity on a majority of occasions. However, not all attempts to debug an Electronic Block stack resulted in the stack meeting the task specification. There were examples where children were able to identify an error but unable to identify the way in which to fix the problem: successful testing, but unsuccessful debugging. Children’s ability to test sub-goals by testing small pieces of “code” was analysed (see Test sub-goals category in Graph 5). Children were most likely to test fragments of “code” when they were constructing interacting stacks that used logic. They were less likely to test sub-goal elements of solutions when creating sensor-action solutions and logic combination solutions (12% for each) and there was very little evidence of such iterative debugging skills being used in the construction of interacting block stacks (8%) or simple

logic structures (6%). Vignette 3 below demonstrates children’s commitment to testing and the problems they had with debugging Electronic Block stacks.

<p>Researcher: “Create this story using Electronic Blocks. A car is out of control. It’s roaming around bumping into everything. Clever scientists discover it stops when they shine a light at it.”</p>	
<p>Elliot picks up a car and brings it close to himself. He then picks up a <i>light</i> block and puts a <i>not</i> block on it. He points it at his car. “How does the car go?”</p>	$\frac{N}{L} C$
<p>Researcher – “Well it’s roaming around ... you have to build the story, how do you think you would make it roam around?”</p>	
<p>Elliot picks up a <i>not</i> block and places it on his car. He points his torch at it again. “I need an eye. Where’s an eye?”</p>	$\frac{N}{L} \rightarrow \frac{N}{C}$
<p>He places a <i>seeing</i> block beside the <i>not</i> block and points his light at it. It continues to roam around. “We need an <i>and</i>. Is an and right?” as he pull apart his construction.</p>	$\frac{N}{L} \rightarrow \frac{SN}{C}$
<p>Researcher “Well what have you got there?” He rebuilds the car with the <i>seeing</i> block and the <i>not</i> block.</p>	
<p>Researcher “Now is that working? What happens when you shine the light at it? Does it stop?”</p>	
<p>Elliot “No ... we need an <i>and</i>.” He changes his structure to include an <i>and</i> block.</p>	$\frac{N}{L} \rightarrow \frac{A}{\frac{SN}{C}}$
<p>He puts another <i>seeing</i> block on top of the <i>and</i> block. He</p>	

creates another *not* block activated torch and then points both torches at the moving car. The car continues to roam. “It’s not working.”

$$\frac{N}{L} \rightarrow \frac{\frac{S}{A}}{C} \leftarrow \frac{N}{L}$$

He then places a *touch* block and a *delay* block on top of the moving structure.

$$\frac{D}{TS} \frac{A}{SN} C$$

Researcher – “OK, we might take some of it off; I think we are getting confused.” The *and* block and everything above it are removed from the structure. “Let’s start from this point.”

Researcher – “Now we need to shine the light at it so it stops.”

Elliot points his torch at the moving car. It continues to roam.

$$\frac{N}{L} \rightarrow \frac{SN}{C}$$

Researcher provides a hint “Remember all the signals get passed from the top down to the bottom.”

Elliot includes a *toggle* block in his car structure. He points his torch at the seeing block. The car continues to roam.

$$\frac{N}{L} \rightarrow \frac{\frac{S}{T_o} N}{C}$$

Researcher says “This one, this one is making it work” pointing to the *not* block.

Elliot removes the *not* block and points the torch at the *seeing* block. The car turns off. He points the torch at the *seeing* block again. The car roams around. Elliot is looking a little dejected and seems to be losing interest in the finding a solution to the task.

$$\frac{N}{L} \rightarrow \frac{S}{T_o} C$$

Researcher – “Well it’s roaming around. Shine the light at it. Will it stop?”

Joseph encourages him – “yeah shine the light at it!” Elliot shines his torch at it – the car stops.

Researcher – “There that’s a solution.” The car has been roaming around and then when the “scientist” shines the light at it the car stops.

Vignette 3: An example of children’s debugging efforts.

The task in Vignette 3 asks children to create a story or scenario, and in doing so doesn’t provide any hints in as to how the task will be solved. This case study provides an interesting insight into the debugging processes of children as they work towards creating solutions to problems where the solution isn’t obvious. It also highlights one of the common difficulties children had in their construction processes – the addition of extra blocks to a structure to achieve a solution and the confusion which results from trying to debug a very complex Electronic Block stack.

Elliot quickly identifies the need for both a *movement* block and a *light* block in a solution to this task. He initially uses a *not* block to activate his light and also uses a *not* block on the car when he realises that it needs to be “roaming around”. Elliot has readily identified how to construct a torch to use with his car structure as well as how to create a “roaming” vehicle; these are both sub-components of the total solution. His early solution using a *seeing* and a *not* block on the car to create the story is only incorrect because he places the seeing block beside the not block, not on top of it. He struggles, not with the selection of blocks for the task, but rather a correct stacking order to produce the required response.

After this initial (almost correct) attempt, Elliot tries to build a correct solution by adding more blocks to his stack. He appears certain that his choice of blocks to that point are correct – he has the correct sensor block and action blocks and does not change them – but seems to feel that the addition of logic blocks will help. He finds it difficult to solve this

problem, but continues to work through a methodical trial and error process, adding blocks and testing solutions in his debugging attempts. While Elliot is able to recognise errors in his Electronic Block structures, but he does not seem to know how to work towards a construction that will meet the requirements of the task. His addition of logic blocks to his construction does not seem to be motivated by a particular strategy but appears random.

Vignette 2 also provides a good example of the difficulty children had identifying the main ideas and sub-goals of a task as well as the problems which occurred when children were unable to identify situations where they incorrectly validated their Electronic Block solution. This task required children to use a *toggle* block so that an action can be switched on and off. As this task has no sub-tasks (or sub-goals) it is not possible to break it into smaller tasks which can be tested. The early collection of blocks by Matt indicated that he had identified what the task required and that he recognised the elements of the solution. However, the lack of sub-goals results in the students moving away from a nearly correct solution. The children try a series of incorrect block combinations until they revert back to the initial nearly correct solution. As the only method of solving this problem is to create a single complete stack solution and test the outcome, it is impossible for the children to be involved in the type of analysis that involves breaking a problem into smaller components to make it easier to solve. It takes a hint from the researcher for the students to find the correct solution.

Conversely, Vignette 4 describes Amy and Kate building a solution to Task 4 and demonstrates the kind of task that can be broken into sub-components. This example of a simple interacting block stack asks children to create a light that turns on when it sees another light. The solution requires the children to identify the need for a light block that can see (using a seeing block) and another light block that can act as the trigger to test the solution. There are a number of ways to activate the second light block.

Researcher: “Build a light that turns on when it sees another light.”

As the researcher is saying this, Kate puts a *hearing* block onto a *light* block. $\frac{H}{L}$

Amy picks up a *seeing* block and turns it around and checks the icon on it.

Amy leans over and looks at Kate’s creation. She removes the *hearing* block already on the *light* block and replaces it with the *seeing* block. $\frac{S}{L}$

Kate leans over the group of blocks, picks up one *seeing* block and puts it down, then picks up another *touch* block, quickly drops it, and then picks a *seeing* block again and keeps it in her hand.

Amy – “We need another *light* block”.

Amy picks it up and puts it beside the other *light* block. Kate places her *seeing* block on it. $\frac{S}{L} \frac{S}{L}$

Kate then picks up two *not* blocks and puts one of them on her light stack. She is about to put the other *not* block on the other light stack when she notices that the light is on. $\frac{N}{L} \frac{S}{L}$

She discards the second *not* block and removes the *not* block from the first stack. She also pulls off the *seeing* block which is on this stack. $L \frac{S}{L}$

Researcher: “So you need this one to turn on (pointing to the second stack) when it sees the light.”

Kate picks up a *touch* block and puts it on the first *light* block. She touches it to turn the light on and points it at the *seeing* block. The *light* block below the *seeing* block turns on. $\frac{T}{L} \rightarrow \frac{S}{L}$

At the same time Amy has picked up a *not* block. She has identified

that it would also work as a trigger for the first *light* block. She doesn't complete that construction as Kate has already placed the *touch* block on the *light* block. Amy watches as Kate verifies the validity of the solution.

Vignette 4: Amy and Kate building a solution to Task 4 - build a light that turns on when it sees another light.

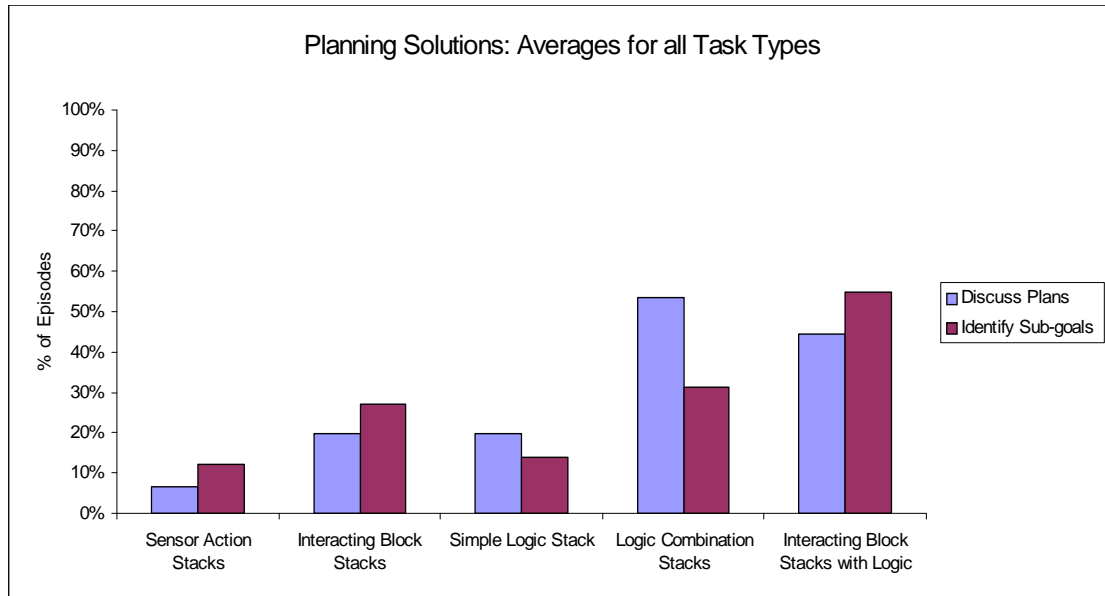
Both Amy and Kate identified what the task required them to do and decomposed the task into less complex elements. It appears that both children have identified part of the problem – needing to create a light that turns on when it sees another light – and together they have identified the need for a seeing block and a light block. They have identified the main ideas of the problem and analysed the components of the solution.

Planning Solutions

This section explores the extent to which children are involved in a top-down approach to programming rather than a bottom-up approach. Programmers tend to use a combination of these two approaches in constructing working computer programs. The programmer will have a good idea of the kind of code to be written, and some of the key code structures that will be used to solve a problem. This is bottom-up information to help the design process. The programmer also needs some top-down information: a plan for how the code structures will come together to solve the programming problem. Previous sections have shown the student's strong bottom-up skills in their ability to identify suitable blocks to complete the task. This section now looks for the complementary top-down planning skills to back up their bottom-up strengths.

The evaluation data was analysed to determine the extent to which children were involved in planning solutions to the twenty Electronic Block programming tasks. Accordingly, instances of children discussing their methods and plans for solving task problems and identifying sub-goals in the programming process were recorded (see Graph 6).

Across all tasks, an average of 28% of potential opportunities for discussing methods and plans were taken by children. Children were involved in identifying sub-goals in the programming process an average of 26% of occasions.



Graph 6: The more complex tasks encouraged the use of discussion and sub-goal identification in task planning.

Children were more likely to discuss plans for solving a task if that task was relatively complex and divisible in some way. Sixty-seven percent of the planning discussions occurred during the construction of solutions to logic combination tasks and interacting block tasks that required the use of logic blocks (see Graph 6). Children were more likely to identify sub-goals in the programming process when creating solutions to tasks that required the use of interacting block stacks (both simple and those that included logic) and during the construction of solutions to logic combination tasks.

Planning was most clearly evident in the identification of sub-goals in the construction process. This is at the heart of a top-down programming methodology. In a top-down methodology, the program is divided into a number of components that may then be developed independently. In the Electronic Blocks tasks, the children would identify the

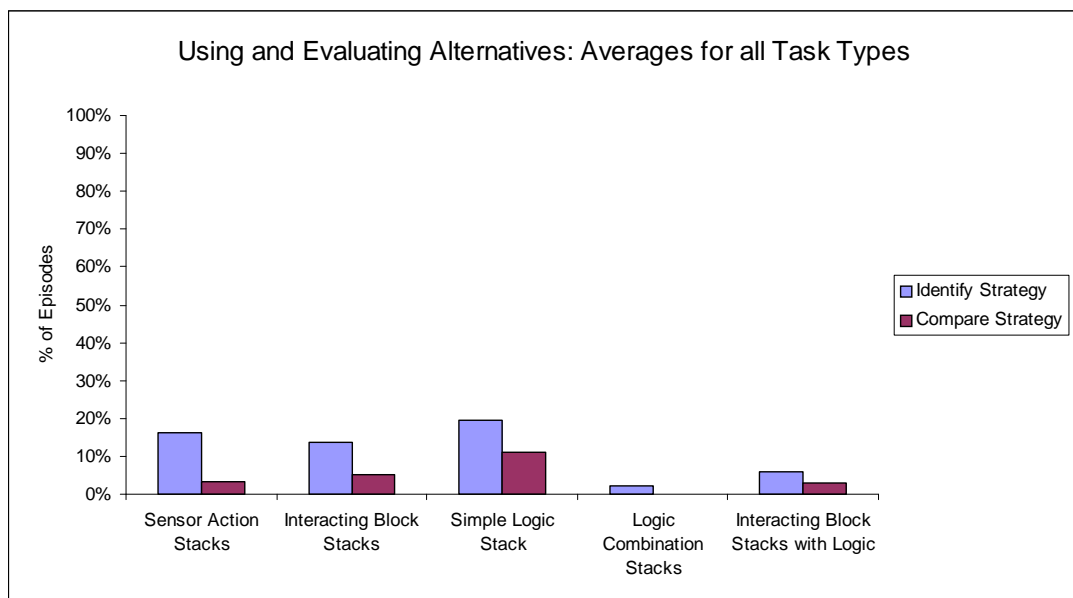
functional elements – say a torch – before exploring the details of how to construct the torch.

This identification of sub-goals is most likely in tasks which are divisible.

Using and Evaluating Alternate Strategies

Using and evaluating alternate strategies is high a level skill that programmers use to critically assess the processes undertaken to create a program. It typically involves the programmer making some judgment about a particular program, and searching for alternate, more effective methods for achieving the same solution. This process involves identifying an alternate strategy, making some comparison about the potential effectiveness of the new strategy compared to the existing one, and if worthwhile, implementing this strategy.

There was limited evidence of children using and evaluating alternate solutions to the Electronic Block tasks. The identification and construction of alternatives are closely linked: children cannot create an alternative until they have identified that one exists. Across the five task types alternate solution strategies were identified on 23 occasions. Nine of those identified alternatives were constructed and compared with initial constructions.



Graph 7: While there was limited involvement in the exploration of alternate strategies, children were most likely to identify and compare solutions during the construction of Electronic Block solutions for simple logic tasks.

In general, children were happy to create one task solution. There was some evidence of identifying and comparing alternatives during the construction of solutions to the less complex tasks. Of the 23 examples of children identifying an alternative solution strategy, 18 occurred during the construction of less complex structures – sensor-action combinations, interacting stacks, and simple logic structures. Seven of the nine examples of children creating and comparing their alternate structures occurred in these three task categories. For example in Task 1 where children were required to construct a voice activated torch, one child built a light activated music structure (using a *seeing* block, a *sound* block, a *toggle* block and a *light* block) that could be activated by a torch, but subsequently identified the simpler approach and used a *hearing* block and a *light* block.

As Graph 7 depicts, children were most likely to identify and compare solutions during the construction of Electronic Block solutions for simple logic tasks. During construction of a solution to Task 6 (a radio with an on/off switch), two students identified an alternate solution using different input sensors to activate the *sound* block. In Task 18 (a building that plays music and turns on a light whenever you touch it) five children identified alternate solutions and two of these children built their solutions and compared them. These children compared a solution that used two *touch* blocks to trigger the *and* block as opposed to a *touch* sensor and a *not* block as the trigger. One of the children commented “you only have to touch one block”.

There is some evidence of using and evaluating alternate strategies in Tasks 5 and 10. Three students while completing Task 5 (a sound activated car and a light activated car) decided they wanted remote controls for their hearing activated cars. As a result they identified an alternate strategy for activating these cars. Five of the ten students who completed Task 10 (a remote control car) either identified, or identified and then compared, alternate strategies. This primarily involved choosing different types of remote control

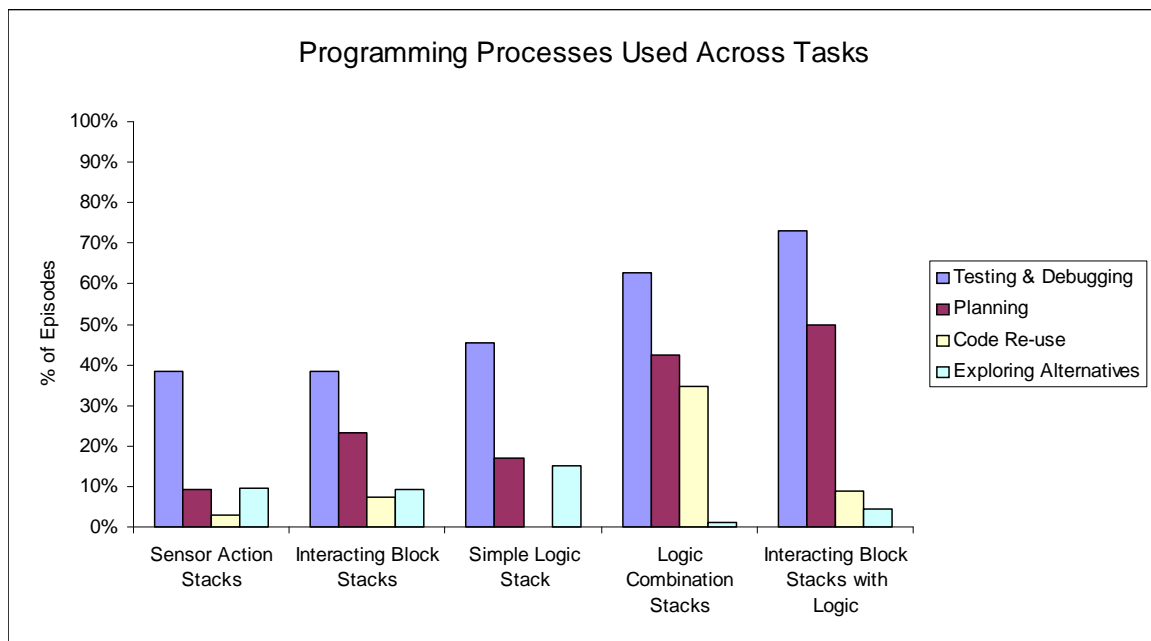
mechanisms. There were only a few examples of children identifying and comparing alternatives while creating stacks for more complex tasks. For the logic combination tasks, there was only one example of a child identifying an alternate strategy across the four tasks.

Discussion

It is clear that the children were able to access the blocks' programmable nature, and learn aspects of programming by performing specific tasks with the blocks. The physical affordances of the blocks enforce the simple syntax and as a result children were unable to create stacks where the function was undefined or ambiguous. Children demonstrated a good understanding of sensor and action blocks across all task categories and consequently were able to build solutions to sensor-action tasks and tasks involving simple interacting block stacks at consistently high levels. Children experienced greater difficulties creating solutions that included logic blocks. Difficulties are due to both problems in identifying suitable logic blocks for the tasks and understanding the nature of Electronic Block communication. As discussed, identifying suitable logic blocks improved with exposure. The results suggest that increased interactions with the Electronic Block can improve children's ability to successfully create programs to achieve solutions to complex problems.

While the blocks generally embody their function, there are difficulties with the children's understanding of the invisible communication signals passed from one block to another. Because the communication signal is invisible, it may be that children assume that there is no signal flow or there is some flow of information from the "male" connector (conventionally associated with output) to the "female" connector (conventionally associated with input). Difficulties that children experience in understanding how the invisible signals are passed between blocks may be best addressed through further discussions and illustrative examples of how the Electronic Block signals work.

Graph 8 above provides an overview of the use of these programming processes across tasks. The results show that children found simple programming tasks easy to complete with little need to use debugging or planning techniques. Children were increasingly challenged as tasks became more difficult. As a result, throughout these tasks, children used debugging and planning techniques more frequently (see Graph 8). The data shows children are capable of testing their Electronic Block structures. When their tests indicated that a solution did not meet task requirements, children would generally engage in debugging processes in an attempt to find a correct solution. This process was at times impeded by children's inability to establish validity criteria – the children would make incorrect assumptions about the nature of the task. The aforementioned improvement through exposure to Electronic Block concepts is in part driven by the children's discovery of solutions during the debugging process.



Graph 8: As the tasks increased in challenge, the children used more debugging and planning processes.

Evidence of children involved in planning processes was notable in the construction of solutions to divisible tasks. Such tasks allowed children to become involved in top-down planning as they identify sub-goals in the construction process. For example, interacting blocks stacks are clearly divisible and lend themselves readily to a combination of top-down

planning with bottom-up knowledge of block functionality. From these results it is hypothesised that further incorporation of divisible tasks within Electronic Block activities would allow children greater opportunities to engage in program planning processes.

In general, evidence of children engaged in the processes of code re-use and exploring alternative solutions was limited (see Graph 8). The results show that children were, on occasion, open to the idea of re-using code concepts. However, the nature of the evaluation was such that it did not provide specific opportunities for this skill to be used in a structured and methodical way. In order for the Electronic Blocks to engage children in the processes of code re-use, tasks could be re-designed to work through a concept, beginning with its use in simple structures and subsequently using it in increasingly complex stacks.

As with code re-use there was limited evidence of children using and evaluating alternative strategies. There are no real incentives for children to attempt alternate solutions as there are no ways to improve the output behaviour of the action blocks. Children were happy once they had achieved any result that produced the necessary output behaviour based on the input requirement. The improvement in levels of use and evaluation of alternate strategies may require the re-design of action blocks to produce improved outputs given certain conditions.

The evaluation showed that children were better able to explore alternative solutions with simple block structures, as these were the structures where children had a deep understanding of the original problem solution. As with children's ability to produce desired solutions, the use and evaluation of alternate Electronic Block strategies may also improve with increased exposure to a wide variety of programming concepts.

The evaluation has demonstrated that children are readily engaged in low level programming activities. The children mastered Electronic Block syntax and functionality with little difficulty and consequently were able to code simple structures. The simplicity of

syntax and semantics allows children to readily become engaged in high order programming activities, opening up opportunities for young children to engage in debugging, code re-use, planning and searching for alternate solutions. The typically high burden of teaching syntax and semantics to children who may not have the cognitive ability to grasp such symbolics concepts has been removed in the Electronic Block programming environment.

Three methods have been identified to improve levels of high order programming skills:

1. Increased exposure to a wide variety Electronic Block tasks;
2. Modifications to evaluation process to better facilitate the development of specific programming skills;
3. Modification of action blocks to produce a continuum of behaviours.

While the evaluation has shown that children are able to use the programmable properties of the Electronic Blocks to become involved in using these programming skills, the modifications suggested would result in increased usage of such skills.

Future Directions

From a functional perspective, some changes and enhancements to the Electronic Blocks family could be addressed:

- the implementation of more action blocks to provide greater variety;
- the inclusion of action blocks which exhibit changes in behaviour given certain input conditions;
- the development of blocks which allow children to explore other programming concepts such as conditionals and iteration;

Further evaluation of the Electronic Blocks would provide a greater insight into their abilities to provide children with opportunities to engage in programming processes. A longitudinal study of Electronic Block programming abilities over an extended period of time

would offer insight into children's ability to build greater complexity into their Electronic Block structures. It would be worthwhile to establish whether children were able to consistently engage in the more complex programming tasks of planning solutions to Electronic Block problems and using and evaluating alternate programming strategies, as their exposure to the Electronic Blocks increased.

Such evaluation could incorporate some of the changes suggested in the discussion section. The evaluation could include tasks that explicitly encourage children to make comparison and provide explanations. Different types of tasks – from those which are narrow, functional through to those which are loosely defined and open-ended – could be evaluated to explore how the nature of the tasks impacts on the skills children use.

Conclusion

In a tradition which began with Logo, Electronic Blocks put young children in the “programming driver's seat”, through providing a developmentally appropriate programming environment. Children are free to autonomously explore the functionality of the blocks as the complexity of the implementation is hidden from them. The syntactic and semantic problems that confront users of conventional programming languages have been reduced and consequently children are able to successfully create Electronic Block programs.

Programming ability was demonstrated by the construction of working stacks and the ability to debug non-working stacks. The data from the evaluation demonstrated that the Electronic Blocks children the freedom and flexibility to create their own working structures as well as work towards specified program behaviours. The task-oriented evaluation established that across all tasks there were children who were able to understand Electronic Block functionality to a level where they were able to solve even the most complex tasks. However, it was generally found that children of this age had most independent success in the construction of less complex tasks such as sensor-action block structures, simple interacting

block stacks and to some extent solution stacks that used a single logic block. While the evaluation demonstrates that the Electronic Blocks provide extensive opportunities for children to build programs autonomously in an open-ended environment, it also shows that in the construction of complex Electronic Block stacks children may benefit from the scaffolding offered in a guided learning environment. The Electronic Blocks shift the focus of programming from learning syntax and semantics to creating working programs to meet specific requirements. As such the blocks open up previously unavailable opportunities for young children to experience programming in terms of planning, building, debugging, re-using, and comparing.

Acknowledgements

The University of Queensland, through a post graduate research scholarship, have financially supported the completion of this research. I gratefully acknowledge this support. I also thank the School of Information Technology and Electrical Engineering for their generous financial support. I would like to thank all of the teachers and children who participated in the evaluation of the Electronic Blocks. Their willingness to participate and enthusiasm throughout made the time spent evaluating the blocks truly enjoyable.

References

- Beaty, J. J. (1984). *Skills for preschool teachers*. Columbus, OH: Charles E. Merrill Publishing Company.
- Bredekamp, S., & Copple, C. (Eds.). (1997). *Developmentally appropriate practice in early childhood education* (Rev. ed.). Washington, D.C.: National Association for the Education of Young Children.
- Cockburn, A., & Bryant, A. (1997). Leogo: An equal opportunity user interface for programming. *Journal of Visual Languages and Computing*, 8 (5-6), 601–619.
- Cumming, G., & Thomason, N. (1996). Educational strategy and cognitive change: From Prolog to Stat Play. *Australian Educational Computing*, 11 (1), 22-27.
- Cypher, A., & Smith, D. C. (1995). KidSim: End User Programming of Simulations. In I. R. Katz, R. Mack, L. Marks, M. B. Rossen, & J. Nielsen (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 27-34), New York: ACM Press.
- Derman-Sparks, L. (1992). Development and characteristics of primary school children. In L. R. Williams, & D. P. Fromberg (Eds.), *Encyclopedia of early childhood education* (pp. 238-240). New York: Garland Publishing Inc.
- Elkind, D. (1996). Young children and technology: A cautionary note. *Young Children*, 51 (6), 22-23.
- Frei, P., Su, V., Mikhak, B., & Ishii, H. (2000). curlybot: Designing a New Class of Computational Toys. In T. Turner, & G. Szwillus (Eds.) *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 129-136), New York: ACM Press.
- Garvey, C. (1990). *Play*. (Enlarged Ed.) Cambridge, MA: Harvard University Press.
- Gilmore, D., Pheasey, K., Underwood, J., & Underwood, G. (1995). Learning Graphical Programming: An Evaluation of KidSim. In K. Nordby, P. H. Helmersen, D. J.

Gilmore, & S. A. Arnesen (Eds.), In *Proceedings of IFIP TC13 International Conference on Human-Computer Interaction* (pp. 145-150), London: Chapman and Hall.

Harel, I. (1991). *Children Designers*. Norwood, NJ: Ablex Publishing Company.

Hughes, M., & Macleod, H. (1986). Using Logo with very young children. In R. W. Lawler, B. du Boulay, M. Hughes, & H. Macleod (Eds.) *Cognition and computers: Studies in learning* (pp. 179-219). Chichester, UK: Ellis Horwood.

Kafai, Y. B. (1996). Learning Design by Making Children's Games: Children's development of design strategies in the creation of a complex computational artefact. In Y. Kafai, & M. Resnick (Eds.), *Constructionism in Practice: Designing, thinking and learning in a digital world* (pp. 71-96). Mahwah, NJ: Lawrence Erlbaum Associates.

Kahn, K. (1996). ToonTalk™ – An animated programming environment for children. *Journal of Visual Languages and Computing*, 7, 197-217.

Kay, A. (n.d.). *Squeak Etoys, Children and Learning* [pdf file]. Available URL http://www.squeakland.org/pdf/etoys_n_learning.pdf.

Lawler, R. W. (1985). *Computer experience and cognitive development: A child's learning in a computer culture*. Chichester, UK: Ellis Horwood.

Lawler, R. W. du Boulay, B., Hughes, M. & Macleod H. (Eds.). (1986). *Cognition and Computers: Studies in learning*. Chichester, UK: Ellis Horwood.

Lee, P. C. (1992). Constructivist. In L. R. Williams, & D. P. Fromberg (Eds.), *Encyclopedia of early childhood education* (pp. 206-207). New York: Garland Publishing Inc.

Lehrer, R. (1993). Authors of knowledge: Patterns of hypermedia design. In S. Lajoie & S. Derry (Eds.), *Computers as cognitive tools* (pp. 197-227). Hillsdale, NJ: Lawrence Erlbaum Associates.

Mackay, H. (1991). Technology as an Educational Issue: Social and political perspectives. In H. Mackay, M. Young & J. Beynon (Eds.), *Understanding Technology in Education* (pp. 1-12), London: Falmer Press.

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. Hendler (Eds.), *Robots for Kids: Exploring New Technologies for Learning Experiences* (pp. 9-33). San Francisco: Morgan Kaufman.

McInerney, D. & McInerney V. (2002). *Educational Psychology: Constructing learning*. Sydney, NSW: Prentice Hall.

My Make Believe Castle [Computer Software]. (1995). Montreal, QC: Logo Computer Systems, Inc.

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

Pea, R. D., & Sheingold, K. (Eds.). (1987). *Mirrors of minds: Patterns of experience in educational computing*. Norwood, NJ: Ablex Publishing Corporation.

Plowman, L. & Stephen, C. (2005). Children, Play and Computers in Pre-school Education. *British Journal of Educational Technology* 36 (2), 145-157.

Resnick, M. (1994). Learning About Life. *Artificial Life 1* (1-2), 229-241.

Resnick, M. (1995). New Paradigms for Computing, New Paradigms for Thinking. In A. diSessa, C. Hoyles, & R. Noss (Eds.), *Computers and Exploratory Learning* (pp. 31-43). Berlin: Springer-Verlag.

Resnick, M., Berg, R., & Eisenberg, M. (2000). Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation. *Journal of the Learning Sciences*, 9 (1), 7-30.

Resnick, M., Bruckman, A., & Martin, F. (1996). Pianos not stereos: Creating computational construction kits. *Interactions* 3 (5), 41-50.

Resnick, M., Bruckman, A., & Martin, F. (1999). Constructional Design: Creating new construction kits for kids. In A. Druin (Ed.), *The Design of Children's Technology*, (pp. 201-222). San Francisco: Morgan Kaufmann.

Resnick, M., Martin, F., Berg, R., Borovoy, R., Colella, V., Kramer, K., & Silverman, B. (1998). Digital Manipulatives: New toys to think with. In M. E. Atwood, C. Karat, A. Lund, J. Coutaz, & J. Karat (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 281-287), New York: ACM Press.

Resnick, M., & Ocko, S. (1991). LEGO/Logo: Learning Through and About Design. In I. Harel, & S. Papert (Eds.), *Constructionism: Research Reports and Essays, 1985-1990*, (pp. 141-150). Norwood, NJ: Ablex Publishing.

Sheingold, K. (1987). The microcomputer as a symbolic medium. In R. D. Pea, & K. Sheingold (Eds.), *Mirrors of minds: Patterns of experience in educational computing* (pp. 198-208). Norwood, NJ: Ablex Publishing Corporation.

Smith, D. C., Cypher, A., & Schmucker, K. (1996). Making programming easier for children. *Interactions*, September – October, 59-67.

Suzuki, H. and Kato, H. (1995). Interaction-level support for collaborative learning: AlgoBlock—An open programming language. In J. L. Schnase & E. L. Cunnius (Eds.), *Proceedings of Computer Supported Collaborative Learning* (pp. 349–355), Mahwah, NJ: Lawrence Erlbaum Associates.

Wyeth, P. & Purchase, H. (2002). Designing Technology for Children: Moving from the Computer into the Physical World with Electronic Blocks. *Information Technology in Childhood Education Annual 2002* (1), 219-244.

Wyeth, P., & Wyeth, G. (2001). Electronic Blocks: Tangible Programming Elements for Preschoolers. In M. Hirose (Ed.), *Proceedings of the Eighth IFIP TC13 Conference on Human-Computer Interaction* (pp. 496-503), Amsterdam: IOS Press.

Vaidya, S. R. (1984). Making LOGO accessible to preschool children. *Educational Technology*, 24 (7), 30-31.

Yelland, N. (1997). Technology: changing the way we think and learn or maintaining the status quo. *Australian Educational Computing*, 12 (1), 3-8.