

Combining Developmental Theories and Interaction Design Techniques to Inform the Design of Children's Software

Peta Wyeth
Mark Venz
University of Queensland

School of Information Technology and Electrical Engineering
University of Queensland
Brisbane, Queensland
Email: peta;mvenz@itee.uq.edu.au

Abstract

This paper describes a novel approach to developing design guidelines for educational software for children. This approach involves the application of theoretical research from the field of developmental psychology and practical techniques from human-computer interaction to guide the design process. The aim of the study was to determine the extent to which Interaction Design techniques support theories from developmental psychology in informing the design of new software systems for children. It was also intended to provide researchers and practitioners with a comparison to these two methods. The study involved a thorough review of literature on childhood development and learning as well as a user study which integrated observations of and interviews with children. The study has shown the effectiveness of each of the data gathering approaches in the development of design guidelines based on the unique needs of children. These two methods have proven to be complimentary on a number of levels, with the literature review providing important high-level, all-purpose data and the user study adding specific contextually rich detail. A combination approach which includes both methods, each with their specific strengths, was found to be successful as a first step in the design of software for children.

Keywords

Interaction Design and Children, Novice Programming Environments, Design, Developmental Psychology, Educational Applications

INTRODUCTION

This paper describes the early design processes undertaken in the development of a new resource for teaching middle school aged children (aged between 10 and 15 years) computer programming concepts. Importantly, this approach involves the application of theoretical research and practical techniques to guide the design process. The development of design guidelines for a computer programming environment, which is developmentally appropriate as well as engaging and meaningful, has been based on a mixing of old and new theories – the application of theories from developmental psychology and the use of techniques from the field of Human-Computer Interaction (HCI).

The key lesson from the design process was the importance of utilizing research from the fields of developmental psychology and HCI. There are many methods researchers and partitioners can employ to guide the design of suitable technology for children, from participatory design techniques to more traditional iterative design processes (for an overview see Druin, 2002), and using these, as well as drawing on a knowledge-base developed over the past 50 years in developmental psychology has been invaluable in the design process. By grounding design in what is known about how children think and learn designers of technology for children are able to utilize a long tradition of well established guidelines. The application of Interaction Design techniques which involve the inclusion of children in the design process to an understanding of children from a developmental perspective is a powerful design strategy. The development of the design guidelines for this new system is an example of how this strategy can be successfully incorporated into the design process.

The aims in this early design phase are centred on the development of suitable design guidelines which can be effectively used in the development of a new programming environment for children. Specifically, the paper reports on the effectiveness of the design approach. The approach was employed to:

- determine the extent to which Interaction Design techniques support theories from developmental psychology in informing the design of new software systems for children; and
- compare the information – taking into consideration issues of depth, breadth, relevance and applicability to software design – which can be gathered from these two alternate methods.

This paper describes the process undertaken to achieve these two aims. Researchers initially completed a review of Developmental Psychology literature, with a particular emphasis on cognitive development. Researchers are interested in exploring the extent to which such theoretical knowledge can be used to guide the development software environment for children. This review resulted in the development of a number of design guidelines.

In addition to a review of literature on childhood development and learning, a user study which incorporated observations of and interviews with children was undertaken. The motivation for this user study was twofold. The School of Information Technology and Electrical Engineering at the University of Queensland currently conducts robot building and programming workshops for middle school aged children. These students use a visual programming language – ROBOLAB™ to program robots. Casual observations of the students indicated that there were shortcomings with current graphical programming systems and these observations cast doubt on the power of graphical languages as a suitable introduction to programming. Subsequently, an objective of the user study was to determine the effectiveness of the ROBOLAB visual programming environment to teach children basic programming concepts. Additionally, researchers were interested in exploring the types of information obtained in the user study in relation to the developmental psychology literature. Three questions related to this issue have been posed:

1. How does the level of detail (both breadth and depth) compare to that of obtained through a literature survey?
2. Can these methods be used successfully together – does the information gathered from one process contradict the other, or are the processes complimentary?
3. How well do each of these data gathering methods translate into design guidelines?

The researchers were also interested in observing children as they use current graphical programming languages in an effort to understand the strength and weaknesses of such languages and subsequently using this information in the design of a new product. The user study resulted in a number of design guidelines. A comparison of the guidelines generated as a result of the literature review and of those that came out of the user study has been made. This comparison highlights the importance of both strategies in the design of products for children.

BACKGROUND

Teaching children to program has been advocated by many in the past 20 years. Papert, in his landmark work *Mindstorms* (Papert, 1980) recognized that computer programming as an educational activity had great potential as a vehicle for the acquisition of useful cognitive skills such as problem solving and reflective thinking. Other researchers have also identified the importance of allowing children to experience the unique dynamic and programming properties of computers and in doing so allowing children to become creators, not just consumers, of computing activities (Resnick et. al, 1996; Kay, 1994; Wyeth and Wyeth, 2001).

The overarching aim of this project – the creation of an environment that allows middle school aged children to become competent designers and creators of technology – follows this philosophy. However, the development of a new graphical programming tool, which takes into consideration the unique requirements of children aged between 10 and 15 year, presents many challenges. There is significant research currently being undertaken that explores the psychology of programming and the corresponding implications for educational programming environments (refer to Pane and Myers (2000) for a full review). It is becoming increasingly apparent that further effort needs to be applied in the study of current educational programming environments and the development of new ones.

As research by Wyeth and Purchase (2003) has shown, matching the programming environment to the specific developmental needs allows children to become involved in meaningful programming experiences that they would otherwise be unable to access. An important aspect of this matching process is a thorough understanding of children from a developmental perspective (Wyeth and Purchase, 2003). When considering the suitability of programming environments for children it may be that graphical programming languages – with their visual representation of functions, their use of wires to represent the flow of control and a syntax which is enforced by simple wiring rules – are most appropriate for children aged between 10 and 12. Indeed, graphical programming languages are recognised as important vehicles through which novices can learn to program (Pane and Myers, 2000). However, for older children, who are better able to think logically and abstractly, textual programming language where all programming functions are represented as text-based abstractions may be a more appropriate vehicle for investigating the concepts that form the foundations of computer programming.

In order to develop computing technologies that support the natural interactions of people, it is necessary to draw upon methodologies used to understand human activity. Observation of users is especially important during the early design stages of a product (Shneiderman, 1998). Researchers suggest that this observation process can closely follow the principles of ethnography. Blomberg (1995) describes ethnography as it relates to human-computer interaction as the process of conducting field research with the aim of developing understandings of

everyday work practices and technologies in use. By following the core principles of ethnographic study (as described in Bloomberg, 1995), researchers are able to study activities in the natural settings in which they occur and develop detailed descriptions of the work experience. However, whereas traditional ethnographers immerse themselves in cultures for long periods of time, the adoption of this technique by HCI practitioners is generally limited to days or even hours of observations (Shneiderman, 1998). Researchers from the field of Education use the term direct observation to distinguish naturalistic observation from experimental based observation (Irwin and Bushnell, 1980). During this initial stage of the project, this type of natural observation, taking place over a number of days, is most effective for capturing the interactions of children as they interact with the existing programming language.

An open-ended questionnaire and a focus group have been utilised as part of the user study. Questionnaires are a well established technique within the field of HCI for collecting users' opinions (Preece et. al, 2002). The questionnaire in this user study is open-ended as it is derived from user observations and is focused particularly on eliciting general (not specific) responses to questions about the programming process. Focus groups can provide a good forum for eliciting user requirements as they are good at gaining multiple points of view (Preece et. al, 2002). This user study utilises this technique to provide participants with the opportunity to openly express their opinions and provide suggestions in relation to the design of the new programming environment.

DEVELOPMENTAL PSYCHOLOGY RESEARCH

This section is a summary of the literature review undertaken as the first part of this research project. The literature review itself was extensive and the information represented here is a high level overview of the key elements of that review. In particular, the focus for the literature review is on cognitive development as it the way children understand new concepts and ideas that is key to learning to computer program.

Adolescents have particular cognitive needs. When they begin their adolescence they are only able to conceive and mentally operate on objects they have directly experienced. Only as they progress are they able grasp abstract concepts. An understanding of cognitive development in adolescence is an important consideration in developing systems to support learning to program. While the execution of a program can be tangible, the program itself consists of abstract symbols manipulating abstract structures. The following sections review some theories of cognitive development in adolescents.

Children within this age group are moving from one stage of cognitive development to another. The younger children (aged 10 to 12) would generally be at a stage referred to by Piaget as concrete operations (Berk 2004, Peterson 2004). Although these children can understand concrete problems, Piaget would argue that they cannot yet perform on abstract problems, and that they do not consider all of the logically possible outcomes. At the age of approximately 11 or 12 children move from this stage to the formal operations stage and are capable of thinking logically and abstractly (Berk 2004, Peterson 2004). They can also reason theoretically.

Prominent in cognitive development theory are Piaget's stages of development. Two stages that are most applicable to this project are Concrete Operations and Formal Operations (Berk 2004, Peterson 2004). Piaget uses the term operation to describe logical thought (Beilman and Snowman 1993). Children aged between seven and eleven are said to be in the Concrete Operations stage. In this stage, they are capable of logical operations, but they solve problems by generalising from physical (concrete) examples (Beilman and Snowman 1993). They are not able to mentally manipulate conditions unless they have previously experienced those conditions (Bee and Boyd 2002). Children in this stage are not able to imagine permutations that are impossible or they have never experienced (Peterson 2004). Around the age of eleven, children enter the Formal Operations stage. They develop a capacity for abstract and scientific thinking which has not been fully evident prior to this age. This is the ability to set up and test hypotheses. They work through tests and hypotheses in a systematic way (McInerney and McInerney 1998). More importantly they are able to do this without physically having to test each hypothesis (Marton and Booth 1997).

While it is possible to observe and measure these stages in children and children seem to be unable to comprehend certain constructs until they've mastered earlier constructs, Piaget's theories underestimate the abilities of children (Winch 1998, Peterson 2004). The change between Concrete and Formal operations is progressive in nature as opposed to distinct change between one stage and the next (Winch 1998). A child able to do formal manipulations in one area may not yet be able to act in a formal way in another area. Children at different ages can display actions belonging to very different stages (Marton and Booth 1997). Most people in the Formal Operations stage, new to a concept, tend to need concrete examples or metaphors to understand the concept (McInerney and McInerney 1998). Physical or concrete tests and examples are still important in building an understanding of new material. They serve to reinforce the validity of the thinking behind and provide a concrete understanding of new theses (Clements and McMillen 1996).

Piaget's theorised that knowledge is constructed by individuals through their actions and interactions with the world, through accommodation (the individual's schema adjusts to the environment) and assimilation

(environment is adjusted to suit the individual's schema) (Berk 2004, Peterson 2004). A person's knowledge of the world is based on what happens to them. From a Vygotskian perspective, cognitive development is a concept of cultural transmission (Moll, 1990). Children acquire knowledge from their peers, parents and mentors. Errors are corrected and extensions to their knowledge are offered. Learners are able to achieve beyond their capabilities with assistance from a more capable person (Marton and Booth 1997, McInerney and McInerney 1998).

Cognitive development, according to Vygotski, is via the Zone of Proximal Development (ZDP). This zone is defined as the distance between what the learner knows and new knowledge (Stuyf 2002, Moll 1990). It is an area between what people can do by themselves and what they can achieve with competent assistance, a mediator.

To place learning in the ZDP an appropriate level of difficulty needs to be established. This level ... must be challenging but not too difficult. The educator then needs to provide for assisted performance. ... The adult provides guided practice to the child. ... As with scaffolding around a building, it is gradually removed so that the child can perform the task independently.

(McInerney and McInerney, 2002, p46)

While the mediator is more often than not another person, it could also be a system of tools the learner is using. Scaffolds provide the learner assistance to work beyond their ability. They simplify tasks by providing direction, support for thinking and modelling expectations (Stuyf, 2002; Jackson et al., 1998). The gradual removal of the support scaffold is also an important part of this learning process, enabling the child to work on a task independently (Jackson et al., 1998).

Design Implications

From the literature survey it is clear that the age of the student is a key factor in determining what is able to be understood, learnt and practiced. Younger students need more concrete examples and methods to learn new skills, but they are able to learn complex and abstract skills beyond their abilities through the use of mediators like scaffolds.

According to Piagetian theories, students in the target age range are generally moving from concrete operations to formal operations. They're more comfortable with concrete methods of problem solving, and are only just being to be able to solve problems without concrete examples. Vygotskian theories maintain that the learner is capable of solving more abstract problems, providing the learner has support. This support can come in the form of mediators, scaffolds and examples.

Based on this literature survey, researchers have proposed 4 key design guidelines. The new programming environment should:

- have the flexibility to support younger students by including concrete examples and exercises, while challenging older students to grasp more abstract constructs;
- allow younger students to manipulate virtual tangible objects;
- provide a flexible support mechanism for all learners, which assists in enabling them to fit more complex ideas into their existing knowledge schemata; and
- ensure that support provided fades as student experience grows.

These high level design guidelines will be used to inform the design of the new programming environment. While they don't provide specific details on what should and shouldn't be included in such an environment, they do highlight the importance of building a system that allows for three particular types of activities:

- direct manipulation activities which provide direct feedback and support the needs of younger children as they learn basic system functionality allowing them to gain confidence and mastery;
- instruction-based activities which allow older children the option of quick and efficient interaction to build computer programs; and
- conversation style activities which provide users with the opportunity to act as partners in conversations with the computer – a two-way communication channel which provides users with advice and help facilities.

Identifying these modes of interaction is important in the early design stage. Once a set of possible ways of interacting with a system has been identified it is possible to think about interface design solutions.

USER STUDY

The user study was conducted to identify the strengths and weaknesses of a currently used graphical programming environment in teaching middle school aged children (aged between 10 and 15) to become successful computer programmers. Researchers were also interested in evaluating the data collected in light of the design guidelines developed as a result of the literature review. The researchers have used the results of the user study to develop a set of design guidelines which will inform the design of the new programming environment. Based on this study, researchers are also able to make comparisons between the data gathered from a theoretical base and that collected during a field study.

In order to gain an understanding of the programming concepts children learn, this study has been three main sections:

- observing and collecting data of participants programming robots to complete set tasks;
- asking students to reflect on their programming experiences; and
- a focus group to gain insight into what these students would like most in new programming environment.

The study consisted of 24 secondary school students. These students, all boys, were completing their second year of secondary school. They were approximately 14 years old. All of these students were enrolled in a robotics subject at their school. Enrolment in this subject was an important prerequisite for the user study as the material covered provided the students with some prior experience with both ROBOLAB (approx 15 lessons) and using LEGO® to build robots. While the participating school was a co-educational institution, only boys had enrolled in this robotics subject.

The students worked in eight self-selected groups and these groups had between two and four members for most of the study. Due to changes in student numbers and a new activity focus, students were regrouped into four groups of five for the final session.

User Study Methodology

The first step in the User Study was to conduct a detailed, in situ observation of children as they program LEGO robots. During this initial investigation the researcher gathered data as students programmed robots. This study was performed using the existing ROBOLAB graphical language which is currently supported by LEGO Mindstorms™. The data collected was in the form of written anecdotal records.

The study was conducted at the student's school. It was conducted during class time, with the permission of the students' regular teacher. The observation of the programming process is focused particularly on how the students design and put programs together and what programming structures (if any) they use. Observations of any common interface errors made or difficulties encountered were also noted. Participants' questions were answered as they arose by prompting with questions to encourage the student's discovery of their own solution. These questions and the results of the discovered solutions were noted with the observations.

In total the user study comprised seven, 70 minute sessions held over a four week period. These were held in place of the usual robotics lessons the boys would have attended. During the first six of these sessions participants were involved in programming a pre-build robot. The final session was divided into two 35 minute sessions and involved students answering reflection questions and brainstorming ideas as part of a focus group.

Programming Observation Sessions

The first session was a review session, where the students worked through the worksheet introducing them to ROBOLAB. The following five sessions involved observing children as they completed programming tasks. The students were asked to record their programming processes, programs and resultant robot behaviour in workbooks provided for the study. Observations were taken of the students' using of the programming environment.

Reflection Questions

The final session was divided into two parts. Initially students were asked to write responses to questions on how they programmed, why they used particular programming constructs, and how they went about correcting any errors. There were three questions that required a written response. These were asked in the first 35 minutes of the final session.

1. Describe the process you use to program.
2. Explain why you used the icons (programming elements) that you did. For example: Why use sub-routines? Why use loops?

3. Describe the process you used to overcome problems with the program.

Focus Group

During the second part of this final session the participants were re-grouped into four groups of five students and asked to brainstorm the following question: "If you were to build a new robot programming environment, what features would it contain?" This session in the study asked the students to brainstorm for ideas and concepts that they would include in a new ideal robot programming environment. This session was held at the end of the study so that the students were able to use their experiences programming the robot to focus their suggestions.

User Study Results

The observations, questionnaire and focus group provided researchers with rich information on the users of the ROBOLAB visual programming environment. The results from the user study have been distilled below to provide the reader with an insight into the key finding of the study.

Programming Sessions

The user study showed that students were largely able to understand the syntax of the ROBOLAB programming environment. This was evident from the children's ability to create simple programs for their robots. The most difficult syntactic knowledge required for successful programming using ROBOLAB is the placement and connection of the icons. Most students were well able to successfully build working programs based on this process. In addition, participants were observed re-using fragments of code through the effective utilisation of sub-routines. The primary syntactic problem observed was that of "bad wires" and is the result of incorrect connection of two icons. This is largely an interface issue as the icons are small and the connection points on the icons are very close together. Frequently novice programmers inadvertently connect one terminal of the icon to the other on the same icon and the wire is hidden underneath the icon.

Importantly, the observations of children's programming activities also illustrated the key difficulties that children had. The processes of systematic debugging and planning within the programming process are considered high-order programming activity (Ohlsson 1995, Marton and Booth 1997). Interestingly, there were only limited examples of students becoming involved in such activities. The user study also showed that, while participants were able to test and debug programs, this largely involved the simple correction of values and fixing problems caused by the interface. While students were able to find and correct problems in their programs, they were not observed involved in systematic debugging processes such as breaking a problem down into smaller components and testing these, isolating problem areas, or methodically outlining a list of potential alternatives or solutions. Initial code solutions were generally based on an assumed similar problem the students have solved in the past. The students then built upon the initial solution for all subsequent and familiar looking problems. If the initial solution is incompatible with the current problem students tended to "hack" solutions (in most cases an additive process) rather than discarding the initial assumption.

During the five programming sessions only one group of children were observed actively planning programming solutions before moving to a computer to enter the program. Despite three students working on paper first, most students had implicitly planned their solutions by "selecting icons necessary for the program". This "just in time" approach to solving the programming problems where students grabbed the necessary icon to solve the next section of the path became problematic when students were involved in solving more complicated the programming tasks.

Reflection Questions

On the last day of the user study, participants were asked to answer three questions on their programming activities and experiences. The students' responses are outlined here.

When asked to describe the process they used to program the robot eleven students answered this question by indicating that they used a process of trial and error, or "grabbing" icons they knew would make the robot perform particular actions. They would form a program, test the program and replace icons and adjust values if necessary. Four students stated they largely used sub routines of the previous tasks, with two students claiming to cut and paste from previously written programs. Three students claimed to have worked solutions out on paper first.

Participants were then asked to explain the use of particular icons and other control structures. Fourteen students stated that they used the icons that they thought were going to produce a particular behaviour. Seven students stated they used sub-routines because they felt that the use of these structures made programs simpler, easier to understand and used fewer icons. One student claimed novelty as his driving force for using sub-routines. Four students stated they didn't use loops or other program control structures because they didn't understand them or know how to use them.

The final reflection question asked students to describe the processes they used to overcome difficulties. A majority of students (15 of the 20 students asked) indicated that they used an ad hoc trial and error debugging process, guessing at what was wrong and changing things until the program worked. The remaining students felt that they methodically worked through the program to find the problem and correct it. One student also indicated that there was a need for outside assistance on some occasions.

Focus Group

During the focus group participants were asked to offer suggestions on the best way to program robots, and what would aid them in doing this. The students were re-organised into four groups for this session. The groups recorded ideas on large sheets of paper. Initial suggestions to initiate the discussion were voice and haptic controlled robots. The supervising teacher helped keep the boys on task. The data has been influenced by the preceding programming sessions.

The data from the focus groups has been broken into three groups:

- Programming
- the Interface
- Programming Assistance

From a programming perspective students requested it be easier to program than "C". They indicated that they wanted better control structures, programs to be less reliant on jumps or gotos and easier ways to create and view groupings of icons. They added to this last point by emphasising that making new icons from groups of other icons would make programs smaller and easier to understand.

The students also provided ideas about interface improvements. Most of their suggestions revolved around the use of icons as programming elements. Students wanted more information on icons and felt that icons should be self-explanatory, or show pop-ups describing their functionality. They wanted better graphics and ways to make or modify existing graphics and create their own icons. In addition to the icons, students also wanted improved methods of connecting icons. They offered suggestions on the layout of icons and how this would create programs. Many students also indicated that they wanted the ability to use text in the creation of programs. Their suggestions included typing keywords and typing in "what you want and it produces the icons".

All of the students indicated that they would appreciate some form of programming assistance. This assistant would have three primary capabilities.

1. Finding errors

Students requested an environment that highlights errors as they are made, provides examples of how to fix errors, and supports their programming efforts by showing how the program could be better.

2. The provision of tutorials and examples

Students wanted examples and tutorials that related to the types of programming activities in which they were involved. They requested that these examples be closely related to the programs functionality. The desired system would be able to get some understanding of the type of program that is being written and provide appropriate support in the form of similar examples and relevant suggestions.

3. A simulation environment for code execution

The requested simulator has at least two functions; showing the results of the program before it is downloaded to the robot, and support for code writing. Students indicated that they wanted the ability to test small sections of code, visual program walk-throughs and a description of how the program would work. The second function of the simulator was producing code or icons. The students liked the idea of this simulation being a place where they could code by example through moving a mouse around the screen in order to produce code.

Design Implications

The User Study has provided researchers with a great deal of information which can be used to guide the development of a new visual programming environment. While much of the information could be related to small interface issues, the User Study also identified four key areas that should be the focus of design efforts.

1. Connection methods and layout that parallel the expectations of the programmer

A new programming environment should use a graphical interface which allows students to snap blocks together as a means of connecting code elements. Placing various restrictions and allowances on the manipulation of these graphical elements will support good programming practise and reduce potential

errors. Users of the system will also be allowed to use the keyboard to select, add and position programming elements.

2. Ability to create new programming elements

The flexibility to customise the graphic elements by changing the graphical face of the element to reflect how users interpret that element's behaviour should be included into a new visual programming environment. Users should be able to edit the face of blocks as a mean of describing new user defined functions.

3. Support in finding and correcting semantic problems

Users will be provided with tutorials and exercises to support their programming efforts. This support will focus on allowing users the opportunity to easily produce semantically correct code. Given that novice programmers appreciate immediate feedback, the new programming environment will also include the capability of using test harnesses (created by the environment) to view the execution of particular code statements. Such a test suite will allow programmers to find semantic problems with sections of code as the results of the execution of small sections of a program will be made visible.

4. Provision of programming scaffolds

A new programming environment should provide varying levels of support through scaffolding, including an initial tutorial with example problems and exercises. In addition, the environment will include a simple artificial expert programming intelligence which prompts the programmer with Socratic style questions based on descriptions of the problem outlined by the programmer.

These guidelines will support the development of the new programming environment by providing an improved interface, more flexible programming execution and important support mechanisms.

DISCUSSION

The interesting outcome of this study from a Human-Computer Interaction perspective is the evaluation of two different approaches to understanding user requirements. Table 1 provides a summary of the guidelines generated by both the literature survey and the user study. In exploring the extent to Interaction Design techniques support theories from developmental psychology in informing the design of new software systems for children, this table clearly highlights the additional information the user study adds to the developmentally-based guidelines. The guidelines generated by the literature survey are more general and provide information of the types of interactions that would be most suitable. The User Study provided a greater depth of information, with specific details on how this interaction might be implemented within the context of creating a new programming environment.

The notion of context is an important differentiator between these two approaches to understanding the needs of users. The guidelines generated by the literature review could be applicable across a range of domains. Indeed, it is this generality that could be highlighted as a strength of these guidelines. The guidelines that were generated as a result of the User Study were more specific in nature. They relate to how children can build computer programs more effectively using software that has an improved user interface and additional support mechanisms. These specifics will play a key role in the development of the new programming environment.

It is interesting to note the match between guidelines developed as a result of the literature review and those that resulted from the User Study. For each of these needs assessment approaches four key guidelines were generated. As highlighted in Table 1, three of the four guidelines developed using the data from the literature relate to all of the User Study guidelines. For example, where the literature outlined the importance of providing younger students with concrete examples and allowing older students a space for more challenging activities, the User Study found that younger children needed more concrete support in improving their knowledge of programming semantics while more experienced programmers were asking for the flexibility to create new programming elements.

The literature review produced a guideline that was not identified through the User Study. This guideline – to ensure that support provided fades as student experience grows – is important in the context of learning. As the User Study primarily focussed on programming *now* (and not at any time in the future), perhaps it is understandable that this type of fading scaffold was not identified. While all students indicated that they needed more support for their programming activities, they did not explore the idea of whether they would (or indeed should) need such support as they gained programming experience. There is also an interesting tension between the best educational solution and the best “getting the coding done” solution. While a fading scaffold is a useful educational mechanism, it could be argued by professional programmers that any intelligent support provided by a software environment would be useful for both the novice and experienced programmer. Improvements to

professional programming environments, in supporting debugging processes and the structuring of code, highlight this point. However, for the purposes of developing an *educational* programming environment, the gradual reduction of support is important.

Developmental Psychology Guidelines	User Study Guidelines
Have the flexibility to support younger students by including concrete examples and exercises, while challenging older students to grasp more abstract constructs.	Support in finding and correcting semantic problems. Ability to create new programming elements.
Allow younger students to manipulate virtual tangible objects.	Connection methods and layout that parallel the expectations of the programmer
Provide a flexible support mechanism for all learners, which assists in enabling them to fit more complex ideas into their existing knowledge schemata.	Provision of programming scaffolds.
Ensure that support provided fades as student experience grows.	

Table 1: A comparison of data gathering techniques in the production of software design guidelines.

FUTURE WORK

Further research is being conducted into the use of the RoboLab visual programming language by both middle school aged boys and girls. While research conducted to date provides a good understanding of the ways in which boys interact with the programming environment, researchers are continuing to gather data which will offer further insights based on observations of both male and female participants.

In the coming year, project researchers will begin work on design and development of a fading scaffold which supports the programming efforts of children within the target age group. This system aims to provide children with a helpful two-way communication channel which enables them to better understand the complexities of computing programming. Children will continue to be a part of the design process, as both informants during the design and testers of system prototypes.

CONCLUSION

This paper reported on two alternate approaches – utilising theoretically-based literature and conducting a field study – to developing software design guidelines. The guidelines generated show the importance of both approaches in the design process. While the literature review was effective for producing general guidelines, the addition of contextual information from the User Study provided important guidelines which will be utilised in the design of the new programming environment. The overlap of the guidelines, with general developmentally-based guidelines mapping directly to context-rich guidelines, demonstrates that these two approaches are complimentary and can be successfully used together. The addition developmentally-based guideline illustrates the importance of understanding children from a developmental perspective when designing educational software. Research from the field of developmental psychology may provide design information that could not be gathered during a field study.

This research study has shown the effectiveness of each of the data gathering methods in the development of design guidelines. They have proven to be complimentary on a number of levels – one approach provides general data and other adds specific contextual detail. Both are important in the successful design of software for children. In the newly emerging field of Interaction Design and Children this study represents a positive step towards understanding how to design technology that supports the unique needs of children.

REFERENCES

- Bee, H. and Boyd, D. (2002). *Lifespan Development*, 3rd edition. Allyn and Bacon, Boston.
- Beilman, R. and Snowman, J. (1993). *Psychology Applied to Teaching*. Houghton Mifflin.
- Berk, L.E. (2004). *Development through the Lifespan*, 3rd edition. Allyn and Bacon, Boston.
- Blomberg, J.L. (1995) Ethnography: aligning field studies of work and system design. In A.F. Monk and G.N. Gilbert (eds.) *Perspectives on HCI: Diverse Approaches*, Academic Press, London.

- Clements, D.H. and McMillen, S. (1996). Rethinking "concrete" manipulatives. *Teaching Children Mathematics*, 2 (5), 270-279.
- Druin, A. (2002) The Role of Children in the Design of New Technology. *Behaviour and Information Technology*, 21 (1), 1-25.
- Irwin, D.M., and Bushnell, M.M. (1980) *Observational strategies for child study*. Holt, Rinehart and Winston, New York.
- Jackson, S.L., Krajcik, J., and Soloway, E. (1998) The design of guided learner-adaptable scaffolding in interactive learning environments. In *Proceedings of the SIGCHI conference on Human factors in Computing Systems*, Los Angeles USA, ACM Press.187-194.
- Kay, A. (1994) Observations about children and computers. Advanced Technology Group, Learning Concepts Group, Apple Research Laboratory Research Note No. 31. [Online]. Available: <http://www.atg.apple.com/technology/reports/RN31.html>.
- Marton, F. and Booth, S. (1997) *Learning and awareness*. Lawrence Erlbaum Associates, New Jersey.
- McInerney, D.M. and McInerney, V. (1998) *Education Psychology, Constructing Learning*, 2nd edition. Prentice Hall, Sydney.
- Moll, L. C., editor (1990) *Vygotski and education: instructional implications and applications of sociohistorical psychology*. Cambridge University Press, Cambridge.
- Ohlsson, S. (1995) Learning to do and learning to understand: A lesson and a challenge for cognitive modelling. In P. Reimann, and H. Spada, (eds.) *Learning in Humans and Machines*, Pergamon, Oxford
- Pane, J. F. and Myers, B.A. (2000) The influence of the psychology of programming on language design: Project status report, in A.F. Blackwell and E. Bilotta (eds.) *Proceedings of PPIG 12*, Cozenza Italy, 193-208.
- Papert, S. (1980) *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York.
- Peterson, C. (2004) *Looking forward through the lifespan*, 4th edition. Pearson Education, Sydney.
- Preece, J., Rogers, Y., and Sharp, H. (2002) *Interaction Design: beyond human-computer interaction*. John Wiley & Sons, New York.
- Resnick, M., Bruckman, A., and Martin, F. (1996) Planos not stereos: Creating computational construction kits. *Interactions*, 3 (5) , 41-50.
- Shneiderman, B. (1998) *Designing the User Interface: strategies for effective human-computer interaction*, 3rd edition. Addison Wesley Longman, Inc., Reading MA.
- Stuyf, R. R. V. D. (2002) *Scaffolding as a teaching strategy*. (from <http://condor.admin.cny.cuny.edu/group4/>).
- Winch, C. (1998) *The philosophy of human learning*. Routledge, London.
- Wyeth, P., and Wyeth, G. (2001). Electronic Blocks: Tangible Programming Elements for Preschoolers. *Proceedings of the Eighth IFIP TC13 Conference on Human-Computer Interaction (Interact 2001)*, July 9-13 2001, Tokyo, Japan, 496-503.
- Wyeth P. and Purchase H. C. (2003). Using Developmental Theories to Inform the Design of Technology for Children. *Small Users - Big Ideas: Proceedings of Interaction Design and Children*, July 1-3 2003, Preston, England, 93-100.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the teachers and students from Pine Rivers State High School who generously gave their time to participate in the RoboLab user study. Acknowledgment is also given to the School of ITEE at the University of Queensland for the generous financial support provided to the second author.

COPYRIGHT

[Wyeth and Venz] © 2004. The authors assign to OZCHI and educational and non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The authors also grant a non-exclusive licence to OZCHI to publish this document in full in the Conference Papers and Proceedings. Those documents may be published on the World Wide Web, CD-ROM, in printed form, and on mirror sites on the World Wide Web. Any other usage is prohibited without the express permission of the authors.