Interactive Design by Children: A Construct Map for Programming

Alexandria K. Hansen[1], Hilary A. Dwyer[1], Charlotte Hill[2], Ashley Iveland[1], Timothy Martinez[2], Danielle Harlow[1], Diana Franklin[2]

[1]Department of Education
Gevirtz Graduate School of Education
UC Santa Barbara
Santa Barbara, CA 93106-9490
{akillian, hdwyer, aockey, dharlow}
@education.ucsb.edu

[2]Computer Science Department
2104 Harold Frank Hall
UC Santa Barbara
Santa Barbara, CA 93106-9490
{tmartinez}@umail.ucsb.edu
{charlottehill, franklin}@cs.ucsb.edu

ABSTRACT
In this paper, we present our analysis of 92 fourth graders' digital story projects completed in LaPlaya, a Scratch-like programming environment. Projects were analyzed for the way that students programmed the start of the story, and if the program integrated user-centered design by providing instruction to the user on how to interact with the digital story. We found that fourth grade students rarely used user-centered design while creating digital stories in our block-based programming environment. Without explicit instruction, the demands of learning programming and simultaneously programming for an abstract user may be too cognitively demanding for the average fourth grader.

Categories and Subject Descriptors
D.1.7 [**Programming Techniques**]: Visual Programming; K.3.2 [**Computer and Information Science Education**]: Computer Science Education.

General Terms
Design, Human Factors, Theory

Keywords
Graphical programming; Computer science education; Elementary school, Interactive, Design, Scratch

1. INTRODUCTION
Coding has taken K-12 education by storm. The increase in popularity and prevalence of graphical, student-friendly programming environments has greatly increased the amount of students who are coding. A 2014 New York Times article [1] claimed that 20,000 K-12 teachers nationwide have introduced coding in their classrooms, and 30 school districts plan to add coding in the fall of 2015. Over 4.5 million 4th-6th grade students participated in this fall's Hour of Code, hosted by Code.org [2]. As coding becomes integrated into the traditional school day, research-based findings need to inform the instructional design of curriculum and accompanying resources for students and teachers.

IDC '15, June 21 - 25, 2015, Medford, MA, USA

ACM 978-1-4503-3590-4/15/06···$15.00
DOI: http://dx.doi.org/10.1145/2771839.2771893

The popularity of block-based programming environments is no surprise; these interfaces reduce the cognitive load required of novice student programmers. Block-based programming environments such as Scratch [3] and LaPlaya [4] reduce typing requirements, remove potential for syntax errors, and provide visual cues such as block color and shape. In these environments, students drag and drop *blocks* (representing commands) to create code (*scripts*). Each script begins with an event (e.g., "when space bar is pressed") and follows with a sequence of action blocks (e.g., "move 10 steps," "turn right"). The scripts are organized by *sprite* (a programmable agent which is often an image of a person, animal, or object), and each sprite's code is shown next to a *stage* that displays the visual output of the program. Block-based programming environments are designed for creating interactive projects that engage users, providing an opportunity to explore novice programming along with novice interaction design.

In this paper, we present our analysis of 92 fourth grade students' digital story projects. These fourth graders (aged 9-10) had recently completed a 16-hour curricular module designed to teach computational thinking concepts and computer programming skills. The digital story was the final, culminating project for the first module. Digital storytelling "allows computer users to become creative storytellers" [5] by using their knowledge of a variety of forms of technology, or, in our case, those created by programming in the interface LaPlaya, a modified version of Scratch. Digital stories consist of multiple characters (*sprites*) and scenes. The characters can be programmed to interact by dialog or actions triggered by motions of sprites or other actions on the interface. Triggers include pressing keys, clicking on sprites, timing, or messages passed between the characters.

We focus our analysis on two aspects of the interactive design. First, we analyzed the sophistication of programming implemented by children (e.g., key clicks, broadcasting messages). Second, we analyzed children's use of user-centered design by examining how those control choices impacted the user. User-centered design is the "design processes in which end-users influence how a design takes place" [6]. In the context of digital stories, we consider user-centered design as providing an interactive experience with explicit instruction to the user about how to progress through the story.

While extensive user-centered design has informed the development of many block-based programming environments to ensure the design and available tools are developmentally

appropriate for a target age group [3, 4, 7, 8], scant research exists on how young students integrate user-centered design in their own block-based programming. We found no existing elementary school curricula that included explicit lessons or activities to teach students this distinction when programming. In this paper, we present an initial categorization of the ways that children controlled their digital stories and the ways in which they integrated user-centered design *without explicit, prior instruction*.

2. RELATED WORK

Seymour Papert's learning theory of Constructionism [9] motivates this work. Like Papert, we believe that individuals learn best when they are actively constructing an entity for public consumption; in our case, students created digital stories. Papert's theory shares roots with Piaget's theory of constructivism. Piaget believed that individuals construct knowledge through experiences, and those experiences are then sorted into cognitive schemes [10]. As more complex experiences occur, new information is integrated into pre-existing cognitive schemes.

Some researchers have recently questioned when students should begin programming [11]. Piaget's developmental stages can help answer this question. While many scholars no longer view Piaget's stages as fixed, they still provide a basic model for what children are able to accomplish, and at what age. According to Piaget's model, children from approximately 7 to 11 years old are in the concrete operational period and are acquiring increased physical dexterity. We contend that students at this age are able to interact with computers through actions such as clicking and dragging, which have proven to be a limiting factor for younger children [12]. Additionally, children at this age are starting to be able to "represent transformations as well as static states,"- an important component of programming [13]. However, children in the concrete operational period still struggle with abstract reasoning. It is not until the formal operational stage is reached at age 11-12 that children are capable of using abstract and systematic thinking, necessary skills for computer science.

Following Piaget's model, our participants (aged 9-10) were approaching the formal operational period. We believe this was an ideal time to begin teaching programming, with age-appropriate instruction. Similarly, Duncan, Bell and Tanimoto [11] concluded that while "there might be a limit on the level of abstraction that students of this age can naturally work with…a different pedagogical approach" can still support their learning.

User-centered design involves "understanding the user and his or her experiences" - an abstract idea. Some claim that this can be achieved through engaging with empathetic design [14]. *Cognitive* empathy is described as "intellectually taking the role or perspective of another person" [15]. Integrating cognitive empathy as a pedagogical tool may support children's use of user-centered design while programming.

3. RESEARCH METHODS

This study is part of a larger study in which we are developing a computational thinking curriculum and researching how 4th-6th grade students learn programming. As stated above, we use a modified version of Scratch, called LaPlaya, which was developed to be user-friendly and age-appropriate for upper elementary school students in classroom settings. For more information about our modifications to Scratch, see [4], and for more information regarding our curriculum, see [16].

This larger study was informed by design-based research methods and used both qualitative and quantitative data analysis. Design-based research studies [17] simultaneously inform the development of curriculum, research, and practice [18], allowing improvements for curriculum and practice as more is learned about student learning.

3.1 Data Collection

In the 2013-2014 school year, we piloted our curriculum in fifteen 4th-6th-grade classrooms at five schools across California. In two that were located furthest away (nine classrooms), we collected only student projects. In the remaining three schools (six classrooms), we filmed classroom instruction and interviewed teachers and students to iteratively inform the curriculum and programming environment. The schools had varying numbers of classrooms, grades participating, start dates, and order of curriculum. Participating schools ranged from 2%-82% designated English language learners, and 4%-100% of students qualifying for free or reduced lunch. Schools generally had equal numbers of female and male students. For this study, we analyzed only the final projects. Students were allowed to select the topic for their final, digital stories.

Overall we collected 103 digital stories from fourth grade students. We omitted eleven stories because they either did not have human subjects permission or children did not write any code. Thus our data set consisted of 92 digital stories. We reviewed a subset of these by watching videos of children presenting their stories and inspecting the children's projects to create an initial construct map and related coding scheme. We used this to create an automated analysis of all 92 projects [19]. This was augmented by visual inspection of each project to determine if children provided directions to the user.

3.2 Developing the Construct Map

We identified two sub-constructs related to interactive design by children: 1) User-Centered Design and 2) Sophistication of Programming. We assigned levels by letters for user-centered and numbers for sophistication of programming.

We defined "user-centered" to mean that the student provided a mechanism for a user to control the actions of the program (e.g., by clicking on sprites or keys) and they provided instruction for doing so. Instruction could be provided by typing instructions onto a background scene (e.g., "Click cat to begin program!") or by creating buttons or sprites that say, "Click me". Our qualitative analysis revealed three levels: (A) Programmer-controlled, (B) Non-Interactive, and (C) Interactive. In programmer-controlled programs, multiple sprites were controlled in different ways, but the control was unintuitive and, as such, the programmer needed to be present to run the program. Non-interactive programs had a single, clear mechanism for running the program (e.g., clicking the green flag) and included multiple events, but no input was required from either the user or the programmer after the initial event. Finally, interactive programs prompted the user to trigger multiple events and provided clear instructions to do so.

The second construct, sophistication of programming, related to the types of programs that students created: Level 1: Simple, Level 2: When key pressed/sprite clicked, Level 3: Timing, and Level 4: Message Passing. Level 1 programs had two attributes: students used only one event (controlled by clicking on green flag) and the scripts contained less than two blocks. These projects were all non-interactive; events were triggered on the green flag and required no further input from the user or

programmer. Level 2 programs included multiple sprites that were controlled completely independently either by pressing keys or by clicking on sprites. Level 3 programs included multiple sprites that appeared to be coordinated because the timing had been manipulated. These programs hard-coded the timing through wait blocks. Lastly, in Level 4 programs sprites interacted by passing messages using the broadcast and receive blocks. This mechanism guaranteed the order in which sprites would run, representing the highest sophistication of programming coordination.

4. FINDINGS

In this section, we describe the ways that students combined user-centered design and programming sophistication to create their digital stories. All combinations are shown in Table 1, with percentages of student projects that fit each category.

4.1 Mechanisms of Control in Digital Stories

Although there were three user-centered design categories and four programming sophistication levels in our construct map, we could not distinguish all twelve possible combinations in this study (these are marked with "n/a" in Table 1). Simple projects (Level 1) ran when the green flag was clicked, the default in LaPlaya. A user could click the green flag without the programmer being present, but we could not determine if this was intended with so few blocks present. Thus, we could not classify a Level 1 project as programmer-controlled though this may have been possible. Students who started their programs on a pressed key or clicked sprite (Level 2) already engaged the user through their event choices so we could not identify non-interactive attributes. Lastly, students who implemented timing and broadcast/receive blocks (Level 3 and 4) used coordinated action without the input of the programmer. By definition, these programs could not be programmer-controlled.

4.1.1 Level 1B: Simple, Non-Interactive

In this level, action among multiple sprites occurred when the green flag was clicked. This was the standard way that programs were run in LaPlaya, but our curriculum focused on teaching a variety of ways that students could start programs. Level 1 projects used only one or two sprites with few completed scripts. All observed Level 1 projects were characterized as Level 1B (12%) because the projects had a clear mechanism for running the program (the green flag), but no additional input was required from either the user or programmer after the initial event.

4.1.2 Level 2A: Multiple Independent, Programmer-controlled Events

In this level, the digital story was programmed so that actions occurred when specific keys or sprites were clicked, but such actions were not made explicit to the user. For example when the letter "A" was pressed, the first character said "Hello", and when "B" was pressed, a second character responded. Unless explicit instruction was provided, the user would not know to press "A" and "B" on the keyboard. We found 50% of projects fell into this category.

4.1.3 Level 2C: Multiple Independent, User-controlled Events

In this level, the digital story was programmed so that actions occurred when specific keys or sprites were clicked, and instructions or other visual cues were provided to the user so s/he could control the program. For example, a student could program a car to move each direction when the corresponding arrow keys are pressed, and provide instruction to the user about how to control the car. We did not see any student projects that fit this description (0%).

4.1.4 Level 3B: Non-Interactive with Timed Coordination

In this level, the story was triggered by a single event (such as clicking the green flag, a sprite, or a key), but, unlike Level 1B, multiple sprites performed actions. They were coordinated with a set of "wait" blocks or "say __ for __ seconds" blocks. Students may have programmed two characters to converse by creating scripts for each character's talk using say blocks and wait blocks to control the timing of when words appeared. This level of control required the student to pre-plan what would be said, but the actual timing was a process of guess and check to see if the story played correctly. Other than the initial command to start the program, this level was non-interactive for the user. We found 5% of projects fell into this category.

A program could be categorized as Level 3C if direction was provided to the user about which sprites or keys to press to progress through the story, but we did not observe any student projects that did this (0%).

4.1.5 Level 4B: Non-Interactive with Message Coordination

In this level, the story was controlled by sprites sending and receiving messages. The visual effect of Level 3B and Level 4B was the same; only the programming distinguished them. Messages replaced the hard-coded timing blocks. When sprites completed a designated action, they broadcasted an invisible message, which triggered actions in other sprites. This required the student to plan and coordinate actions among multiple sprites. The story may have started by clicking the green flag, key, or sprite. Other than beginning the story, this mode was non-interactive for the user. We found 32% of projects fell into this category.

4.1.6 Level 4C: Interactive and Coordinated

In this level, the student included explicit instruction to the user about how to interact with the program. The story could be initiated in a variety of ways, such as clicking the green flag, sprite, or key. Additional instruction was provided to the user about how to progress through the story. The program may have

Table 1. Interactive Design Control Construct Map for Programming

	User-Centered Design		
Sophistication of Programming	**Programmer Controlled**	**User: Non-Interactive**	**User: Interactive**
Simple	N/A	Level 1B (12%)	Level 1C (0%)
When Key Pressed or Sprite Clicked	Level 2A (50%)	N/A	Level 2C (0%)
Timing Blocks	N/A	Level 3B (5%)	Level 3C (0%)
Broadcast/Receive Messages	N/A	Level 4B (32%)	Level 4C (1%)

provided instructions such as "Click on the sprites 1, 2, and 3 to move on," or "Press keys 1, 2 and 3 to see each scene of the story." Without explicit instruction, only 1% of students fell into this category (n=1). Figure 1 shows an example of a digital story that involved the user. None of the other student projects analyzed included explicit instructions to the user.

Figure 1. **A Level 4C student project.**

5. DISCUSSION

Our analysis demonstrates that students used a variety of events and coordination techniques to create their stories. However, we also found that, perhaps not surprisingly, when not specifically prompted, students tended to view their project from their own perspective, omitting interactive features that could be understood by an outside user. Very few employed user-centered design in the absence of instruction.

We propose several possible explanations to explore in further research. Fourth grade students were not prompted to choose a particular user as their audience. Perhaps an abstract user was too difficult to design for, but the prompt of thinking about a particular person could have improved the designs. A second explanation relates to the programming environment. In LaPlaya, the development environment is exposed at the same time as the runtime environment. The scripts were visible when students ran their program. Students could have viewed the user as someone with access to the scripts, making explicit instructions unnecessary. This could further confuse students about the relationship between development and deployment if/when they transition to more traditional text-based languages.

Our categorization presented here suggests a preliminary learning progression for user-centered design that can be further articulated and tested. We chose to focus on fourth grade students to identify the lower anchor point of the learning progression. This analysis led to concrete revisions to our curriculum for the 2014-2015 school year. We added engineering design lessons to teach students the process of design, including thinking about the user in the program they are creating. We also created demonstrations that showed students examples of projects that did and did not include instructions to the user.

6. ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation CE21 Award CNS-1240985. We are grateful to the teachers and children who participated in this project.

7. REFERENCES

[1] Richtel, M. 2014. Reading, writing, arithmetic, and lately, coding. *The NY Times.*

[2] Alvarado, C. 2014. CS Ed Week 2013: The hour of code. *ACM SIGCSE Bull*, *46*,1, 2-4.

[3] Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. 2009. Scratch: Programming for all. *Commun ACM*, 52, 11, 60-67.

[4] Hill, C., Dwyer, H. A., Martinez, T., Harlow, D., & Franklin, D. 2015. Floors and flexibility: Designing a programming environment for 4th – 6th grade classrooms. In *SIGCSE '15.* ACM.

[5] Robin, B. 2008. Digital storytelling. *Theor Pract, 47*, 220-228.

[6] Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M. & Wiedenbeck, S. 2011. The state of the art in end-user software engineering. *ACM Comput Surv*, 43, 3, 21.

[7] Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P, & Resnick, M. 2013. Designing ScratchJr: Support for early childhood learning through computer programming. In *IDC '13*. ACM.

[8] Cooper, S. 2010. The design of Alice. *ACM T Computing Education*, 10, 4, 15.

[9] Papert, S. & Harel, I. 1991. Situating constructionsim. *Constructionism*, 36, 1-11.

[10] Driver, R., Asoko, H., Leach, J., Mortimer, E., & Scott, P. 1994. Constructing scientific knowledge in the classroom. *Edu Researcher, 23*, 7, 5-12.

[11] Duncan, C., Bell, T., & Tanimoto, S. 2014. Should your 8-year-old learn coding? In *WiPSCE '14*. ACM.

[12] Donker, A., & Reitsma, P. 2007. Young children's ability to use a computer mouse. *Comput & Edu*, 48, 4, 602-617.

[13] Siegler, R. & Alibali, M.W. 2005. *Children's thinking*. Pearson Prentice Hall, New Jersey.

[14] Kouprie, M. & Visser, F.S. 2009. A framework for empathy in design: Stepping into and out of the user's life. *J Eng Design*, 20, 5, 437-448.

[15] Mead, G.H., 1934. *Mind, self and society.* Chicago, IL: University of Chicago Press.

[16] Franklin, D., Harlow, D., Dwyer, H., Henken, J., Hill, C., Iveland, A., Killian, A., & Development Staff. 2014. *Kids Enjoying Learning Programming (KELP-CS)- Module 1 Digital Storytelling.* Available at https://discover.cs.ucsb.edu/kelpcs/educators/KELPCSIntro.pdf

[17] Barab, S. & Squire, K. 2004. Design-based research: Putting a stake in the ground. *J Learn Sci*, 13, 1, 1–14.

[18] Brown, A.L. 1992. Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *J Learn Sci*, 2, 2, 141–178.

[19] Bryce B., Hill, C., Len, M., Dreschler, G., Franklin, D., Conrad, P. 2013. Hairball: Lint-inspired static analysis of scratch projects. In SICSCE '13. ACM.