Empowering Families Facing English Literacy Challenges to Jointly Engage in Computer Programming

Rahul Banerjee, Leanne Liu, Kiley Sobel, Caroline Pitt, Kung Jin Lee, Meng Wang†, Sijin Chen, Lydia Davison, Jason C Yip, Amy J Ko, and Zoran Popović
University of Washington, Seattle
†Wuhan Huada National E-Learning Technologies, Hubei, China
{banerjee, zoran}@cs.uw.edu, {liul26, ksobel, pittc, kjl26, mengw414, sijinc, lydiad6, jcyip, ajko}@uw.edu

ABSTRACT
Research suggests that parental engagement through Joint Media Engagement (JME) is an important factor in children's learning for coding and programming. Unfortunately, parents with limited technology background may have difficulty supporting their children's access to programming. English-language learning (ELL) families from marginalized communities face particular challenges in understanding and supporting programming, as code is primarily authored using English text. We present BlockStudio, a programming tool for empowering ELL families to jointly engage in introductory coding, using an environment embodying two design principles, text-free and visually concrete. We share a case study involving three community centers serving immigrant and refugee populations. Our findings show ELL families can jointly engage in programming without text, via co-creation and flexible roles, and can create a range of artifacts, indicating understanding of aspects of programming within this environment. We conclude with implications for coding together in ELL families and design ideas for text-free programming research.

Author Keywords
Coding; Programming; Joint media engagement (JME); English-language learning (ELL) families; Text-free.

ACM Classification Keywords
H.5.2. User Interfaces

INTRODUCTION

When it comes to children's resources for learning to code, there are many programming environments to choose from [29,42]. However, after a child has access to a programming environment, the contexts in which they use these environments also matter for learning. In particular, researchers have highlighted how co-engagement, or joint media engagement (JME) [49], is especially supportive of children's learning with new media and how parental engagement can be a key factor in children's learning [17,24]. Unfortunately, parents not connected to technology fields may have less programming knowledge to support their children's learning [13]. To address this, family-oriented programs using design-based activities like Family Creative Learning (FCL) [43] can empower such families lacking "preparatory privilege" [35] to get involved with their children's creative activities.

While family-oriented programs can engage diverse populations, participating in them can be difficult for people facing English literacy challenges. One reason is that FCL relies on programming tools using English text (i.e., Scratch [42]). Yet, among U.S. immigrant families from lower-SES and marginalized populations, it is common for the children to have English fluency while the adults continue to experience difficulties with English literacy [22] (we refer to these types of families as ELL families). Also, the U.S. is a multicultural society, with up to 59 different languages spoken in a single area [46]. While programming tools can be localized to multiple languages [10], it is challenging for translation to cover such a large number of languages. Even if manual or automatic translation was accurate and feasible, instructors of FCLs would need to understand multiple languages and use multiple translated versions of the same interface to adequately support family learning.

Given these challenges, we ask: *how might we empower children and parents in ELL families to jointly engage in learning to code?* We pursued this question by designing a text-free, visually concrete environment for coding, and then studying parent-child co-use of this system in multi-language ELL family-oriented sessions at community centers. We used English to instruct the bilingual children how to code in this environment, and then they taught their parents how to use the system. In this paper, we present the design principles behind this new programming interface and approach for empowering ELL families to jointly engage in learning to code. We also describe our case study comprising three family-oriented workshops held at community centers.

Through this work, we contribute a system embodying two design principles, and through studying its use, a new understanding of how such coding environments may support and empower ELL family members jointly learning to code.

CHI 2018, April 21–26, 2018, Montréal, QC, Canada.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5620-6/18/04...$15.00.
https://doi.org/10.1145/3173574.3174196

BACKGROUND

Researchers have created several systems to help children learn to code [46,29,41,42]. However, many critical sociocultural factors determine whether children successfully engage and learn. A survey of prior end-user programming environments can be found in [29], but none of these systems have focused on ELL families coding together.

Parental support for children to code

One factor is joint media engagement (JME), which Takeuchi and Stevens define as "spontaneous and designed experiences of people using media together" [49]. JME between parents and children is linked to increased family connectedness [40], higher self-efficacy and expertise with computers [34], and parental involvement has a positive effect on grades [17] and academic achievement [24]. Simpkins et. al. [47] found that parent-child coactivity around computing predicts a child's interest and engagement in computing, and Armon [1] found that pairing parents and children while learning programming improved creativity and thinking skills. Thus, for children learning to code, parents can be an important source of support.

Unfortunately, adults working in areas unrelated to computing often face challenges in locating ways to support their children's coding endeavors [13]. Roque created Family Creative Learning (FCL) to empower such families to learn new technologies and design projects together, based on their interests [43]. However, Scratch [42], the system used in FCL, generally uses English text. Given that 59% of children in U.S. newcomer families live with at least one parent who is not proficient in English [23], existing family-oriented programs for coding can pose language barriers. Further, the U.S. is a multicultural country, with many areas having high language diversity [46]. Multiple languages may be used even at a single community center for immigrants, implying that programs like FCL using English-language tools could deter non-English families. Thus, it is challenging for family-oriented programs to serve such diverse ELL populations.

Native language support and removing text

Dasgupta and Hill [10] found that versions of Scratch translated into local languages were associated with a higher "growth" rate in learners' use of programming constructs. They concluded that the dominant effect of localization might be that "being able to engage in one's primary language supports users who would otherwise not learn to code at all." However, there are many challenges with simply translating a coding interface. Automated translation is not guaranteed to make reasonable choices for reserved keywords, especially with languages having gendered nouns. Manual translation to avoid these problems takes time (e.g., a year of work to translate Scratch blocks into Amharic [52]), and changing the interface forces this to be re-done. Also, unlike prior work that focused on homogeneous populations [10], diverse languages are a reality for community centers in the U.S. A family-oriented coding session could employ multiple translated versions of the system, but this would create a dependence on instructors who can teach and provide help in multiple languages.

Instead of localizing interfaces, some work has explored removing text altogether. Medhi et. al. [36] worked with lower-SES illiterate communities in Bangalore to investigate the usability of text-free interfaces for finding employment. They found that "text-free designs are strongly preferred over standard text-based interfaces" by these communities, with such interfaces "potentially able to bring even complex computer functions within the reach of users who are unable to read" [36]. Prior coding interfaces have removed text from their interface, using abstract representations to represent computation. PICT [20] programs were flowcharts with icons representing operations connected to form computations, while DataFactory [50] programs were numbers moving on conveyor belts between machines. ToonTalk [26] used metaphors connecting real-world objects (e.g. robots, birds, boxes, etc.) with programming concepts. These systems visually depicted abstract computational ideas using a graphical metaphor (e.g., icons for functions, birds or conveyor belts for data channels, etc.), exposing an animated interface for doing numeric computation. In KidSim [9], users created visual rewrite rules to implement agent-based simulations. diSessa's Boxer used Naïve Realism [14] to depict programs via graphical notation, but its notation also used text (e.g., boxes labeled with text representing variables or procedures).

Among recent systems, Scratch Jr. [19] replaced the text labels on code blocks with graphical symbols. While using graphical symbols removes the explicit need for natural language, such symbols are rarely universal across language and culture [27]. Also, Scratch Jr. is aimed at children ages 5-7, and would be less helpful for older children (ages 9-12). Moreover, graphical, symbolic notations do not necessarily support learning. According to du Boulay, a "running program is a kind of mechanism and it takes quite a long time to learn the relation between a program on the page and the mechanism it describes." ([16], p. 285) Neither textual nor graphical notations make this relation visible. Recent work has found ways to teach such relations using program visualization, but this relies heavily on natural language explanations [39].

Children teaching parents

Since many children in immigrant families in the U.S. are fluent in English [22], an alternative to having the coding environment convey the meaning of symbols is for children to learn the programming language in a text-free system via verbal instruction in English, then teach their parents using their native language. For example, Yip et. al. [56] studied how children in ELL families often act as translators and information brokers, leveraging their linguistic capabilities, cultural familiarity, and technical skills to help their families gain access to information resources. To learn to code together, children could teach programming to their parents, allowing families to jointly engage in coding. Few other prior works have explored this possibility of supporting ELL parents, and none in the learning of a coding interface.

THE BLOCKSTUDIO SYSTEM

Prior systems have employed either textual or graphical notation that pose barriers to ELL families learning to code via joint media engagement. We present *BlockStudio*, a programming-by-demonstration [8] environment free of notation, supporting our goal through a unique combination of two design principles.

First, BlockStudio is ***text-free***, in that it avoids any use of text in the coding interface. We achieve this through the existing programming paradigm of programming-by-demonstration [8], in which users provide examples of the behavior they would like the system to perform, and then the system synthesizes a more general rule from those examples. By using demonstrations, our system supports authoring by *showing* with examples rather than by *telling* with natural language.

Second, BlockStudio is ***visually concrete***. By following a programming-by-demonstration paradigm, BlockStudio can express program state and operations via visually concrete attributes on the screen, like position, size, shape, and color. Through its visually concrete universe, BlockStudio poses a notional machine [16] where the program on the screen attempts to self-describe its execution behavior through examples, rather than keeping the machine behavior invisible, and therefore requiring natural language explanation to learn. The abstract task of reasoning through "what does this code block in the program do?" is replaced by the concrete task of thinking through, and then enacting "how should the on-screen rectangles change in this particular scenario?" The concreteness of this paradigm could allow a child to learn the language through demonstration, and potentially teach a parent the language through similar examples, all without any reliance on natural language. BlockStudio's interface is shown in Figure 1 (left), with its main parts labeled in Figure 1 (right). Figure 2 shows how BlockStudio leverages these principles to allow users to specify program behavior without dealing with abstract text or graphical symbols that represent behavior. Users place rectangular blocks on the screen (the spaceship in the figure), and then demonstrate to the system how these blocks should be modified in response to user input and collisions with other blocks. This simple set of programming-by-demonstration abstractions achieves both principles of being text-free and visually concrete. In the remainder of this section, we describe our design process and provide further details about the system.

Design Process

Previously, researchers working with *KidsTeam UW*, an intergenerational design team at a university, had iteratively refined a coding environment aimed at children ages 9 through 12 [2]. In these sessions, they used Cooperative Inquiry [15], which is a participatory design [30] method focused on developing equal and equitable partnerships

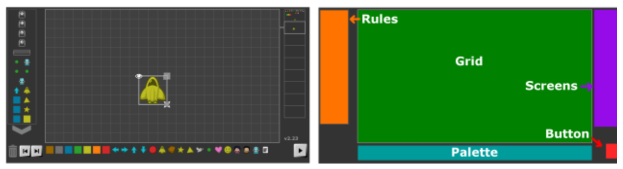

Figure 1: BlockStudio user interface (left) and main parts (right): Palette, Grid, Rules, Play button, and Screens.

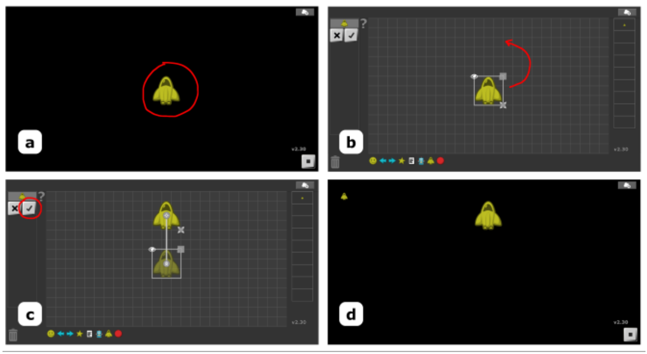

Figure 2. How to create a rule in BlockStudio: (a) Click spaceship to trigger event, (b) Drag spaceship up to demonstrate the change, (c) Click ✔ to end demonstration, (d) Clicking spaceship now moves it up. (See supplementary video for details)

with children through co-design [57]. This prior work explored the idea of minimizing text in a coding interface [2], but not joint use by families, or ELL users. During the summer of 2016, we used this system to conduct a pilot study (unpublished) with children at Springdale Middle School (pseudonym), where the school was in charge of recruitment. Here, we had ELL students (Spanish, Amharic) who wanted to participate, even though the instructor did not speak these languages. Using translators enabled these children to create some artifacts using this prior system, indicating that people facing challenges with English can learn to use a coding interface via translation.

Revised System

Given our positive experience with human translators, and guided by our focus on ELL families and our design principles derived from this objective, we modified the user interface of BlockStudio's prior version to create a revised system suitable for ELL families. To meet our text-free design principle, we removed all text from the interface (Figure 3), consisting of around a dozen words, mostly labeling parts of the interface. We replaced text with symbols only when absolutely unavoidable, ending up with three abstract symbols in our interface: ✓, ✕, and **?**, for 'confirm', 'cancel' and 'question', respectively. The first two are standard symbols in interfaces and thus familiar to non-English users who use a smartphone, while the question mark may be revised in future versions. To meet our visually concrete design principle, we rebuilt some of the interface. At Springdale, we had found that the lack of visual feedback showing block modifications during rule demonstration caused usability problems, as did the absence of a way to

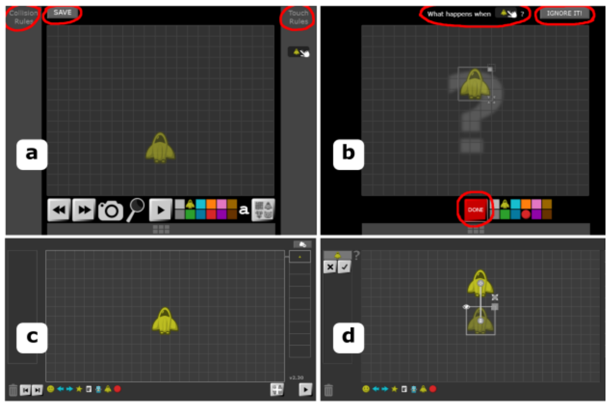

Figure 3. The top row shows the prior system, with text circled in red. The bottom shows BlockStudio, without the text. The left shows block placement and the right shows rule creation.

edit rules (in the prior version, rules had to be deleted and recreated). We addressed these issues by explicitly providing visual feedback showing changes made by the user (one example is shown in Figure 3d), and by letting users modify the demonstration for a rule after its creation.

Programming Model

Programs in BlockStudio consist of rectangles called *blocks* (which can be created, repositioned, resized, changed, or deleted) as well as *rules*, which specify *actions* (changes to blocks) in response to *events*. BlockStudio blocks should not be confused with code blocks found in block-based programming languages [6,45], which are visual representations of abstract syntax trees [25], and are labeled accordingly (e.g., “repeat”, “if – then”, etc.).

BlockStudio blocks may have a solid color appearance or display a picture (like a cat or a spaceship), but BlockStudio internally treats all blocks as rectangles to detect overlaps. Our system uses simplified notions of movement, namely linear motion and collision/overlap, both of which are visually concrete and can be communicated without having to speak English (if nothing else, one can convey direction of motion by pointing with one’s finger). This visually concrete vocabulary allows the creation of familiar 2D games (like Pong™, Space Invaders™, PacMan™, Flappy Bird™, maze games, etc.), composed of simple patterns, like moving a character, firing at objects, controlling a paddle to catch a bouncing object, eating collectibles, keeping track of lives, loading a new level, etc. Thus, BlockStudio affords a range of creative possibilities, while adhering to our two design principles. Rules are based on block types. Thus, if clicking a spaceship (“touch spaceship” event) causes the spaceship to emit a pellet moving up (“create pellet” action), then clicking any spaceship on the screen makes that particular spaceship emit a pellet with the same velocity. Similarly, if the user specifies that a pellet colliding with an asteroid causes both colliding blocks to be deleted, then all asteroids can be deleted by moving the spaceship around (using additional rules) and clicking it to fire pellets.

Event Type	How the Event is Triggered	Default Action
Touch	By clicking (mouse) or tapping (touchscreen) a block	Do nothing
Key	By pressing a key on the keyboard	Do nothing
Collision	When blocks with motion collide	Bounce

Action	Example
Create block	A spaceship “fires” a pellet
Move block	A spaceship moves right
Resize block	Eating a dot makes a character “grow”
Edit block type	Eating power pellet makes PacMan™ invincible
Delete block	A fired pellet destroys an asteroid
Bounce	A ball deflects off a “Pong” paddle
Load screen	Go to saved block arrangement (e.g. “You Win”)

Table 1. All Events and Actions possible in BlockStudio.

Creating blocks and rules

Users create blocks by dragging from the palette (Figure 1, blue) onto the grid (Figure 1, green). Users can reposition, resize, replace, or delete blocks to set up the “starting appearance” of a game or an animation. If a user changes their mind, they can stop the program and change the blocks again. Figure 2 illustrates BlockStudio’s rule-authoring process. Children create and modify rules through programming-by-demonstration [8]. When an event occurs (e.g., clicking a block), BlockStudio allows the user to demonstrate the response for that event, thus creating a rule. These demonstrations are concrete, showing which block(s) need to change, and in what way(s). By comparing the on-screen blocks before and after this demonstration, BlockStudio infers a generalized state change, and internally synthesizes code to accomplish this change. For instance, if the user presses spacebar and moves a block from the position (4,10) to a new position (6,10), then BlockStudio infers that every time spacebar is pressed, that block’s x-coordinate should be increased by 2. Besides changing a block’s position, a user can demonstrate other concrete changes. Table 1 shows BlockStudio’s events and actions.

CASE STUDY METHOD

To investigate how well ELL families could jointly engage in learning BlockStudio, we conducted a case study [37] involving three different community centers. Every site included a workshop of either one or two sessions at a single community center. Each of the sites used the BlockStudio system, had the same instructor (the 1st author), and had ELL families as participants.

Context

We organized “family night” workshops at three different family-based community centers (Table 2) serving African immigrants and refugees serving marginalized communities. We refer to these locations using the pseudonyms Bulsho, Rajo and Farxad centers. Each of the community centers served a high population of ELL immigrants and refugees from Ethiopia, Somalia, Sudan, and other African communities. A community center context provided advantages over individual usability testing with multiple families. For instance, certain families needed child care services, because some of their children were too young to participate. Therefore, community center employees helped

Venue (Comm. Center)	Workshop Format and People	Audio, Video and Field Notes	Screen Capture with Cursor	Copies of Created Artifacts
Bulsho 11/2016	Day 1: children Day 2: families	☑	☑	Not Saved
Rajo 5/2017	Days 1 and 2: families	☑	Slow Internet	☑
Farxad 6/2017	Only 1 Day: families	☑	☑	☑

Table 2. An overview of the workshops showing their formats and listing what data was gathered at each location.

set up childcare on the premises, in adjoining rooms. Hosting our workshops at community centers was also important because we could not assume that all families would have an Internet connection at home, or an appropriate space to set up a computer and a video camera. Every community center had functional Internet connectivity, even if it was slow. Also, a family night setting allowed parents to come pick up their children from routine after-school activities, but spend extra time with their child (based on their schedule), thus jointly participating in our coding workshops. Takeuchi and Stevens [49] would define this as *fit*: for families "to use a new platform with any regularity, it should easily slot into existing routines, parent work schedules, and classroom practices."

Recruitment

We consulted with each of our three community centers to decide the timing, number and duration of sessions, to find a good fit with the parents' schedules and constraints. The community center had families sign up in advance, but some signed-up families were unable to attend due to personal reasons, while other families that had not signed up wanted to participate on the day of a workshop. We decided to not turn away anyone who wished to participate. We gave participating families a $30 gift card per session to compensate them for their time. Our participants included first-generation immigrant adults who had various degrees of English fluency (Table 3), as well as their children (all of whom were fluent in English, except for one girl). This allowed nearly all of the children to function as translators for their parents (the roles were exchanged for the child who knew no English). Participants' schedules constrained the number of sessions we were able to organize at each location (see Table 2 for details on each workshop's format and data collected). Table 3 details our participants (13 adults, 17 children).

Procedure

Each community center was unique, but we tried to conduct each workshop using the same structure (influenced by FCL [45]): obtaining consent and assent, followed by introductory explanations encouraging the children to teach their parents how to use BlockStudio, and finally opening up to free exploration, for a total of 60-120 minutes per session. Children provided assent, while parents provided parental consent for their children and informed consent for themselves.

Center	ID	Parent, Age, & Language(s)	Parent English Fluency	Child(ren) and Age(s)	Joint Use Time (min)
Bulsho	F1	Mother (46), Somali	Low	Son (12)	38
	F2	Mother (35), Arabic, Afar	High	Daughter (12)	40
	F3	Mother (33), Somali	Low	Son (11)	39
	F4	Mother (33), Amharic	Low	Daughter (12)	41
Rajo	F5	Father (48), Amharic	None	Daughter (8) Daughter (9)	108
	F6	Father (52), Amharic	Low	Daughter (8)	140
	F7	Father (51), French	Low	Son (9) Son (13)	107
	F8	Mother (32), Amharic	Low	Daughter (10) (No English)	84
Farxad	F9	Mother (38), Somali	Low	Daughter (12) Son (9)	29
	F10	Father (53), Amharic	High	Daughter (11)	98
	F11	Mother (38), Amharic	Low	Daughter (10)	100
	F12	Father (51), Amharic	Low	Son (13)	74
	F13	Mother (36), Amharic	Low	Daughter (7) Son (9)	73

Table 3. Participant demographics at each location. We refer to participants by a pseudonym and a family ID (column 2). Joint Use refers to parent and child(ren) being at the laptop.

As there were multiple non-English languages in use at all locations, we could not provide translated versions of our consent form. However, for some English literate parents, we explained the form using basic English. For adults with no English literacy, the community center employees or participants' children helped us verbally communicate the contents of the consent form.

After obtaining consent, each family sat at a laptop connected to the Internet with a mouse, with an empty BlockStudio project on-screen. Next, the 1st author demonstrated the basic concepts of BlockStudio, like creating a rule (e.g. "When the up-arrow key is pressed, move a block up.") using English explanations, and then encouraged the children to try what they had just learned. Once the children were comfortable with the basics, we encouraged them to teach their parents the same concepts using their own language. Parents were encouraged to ask their children for help when they got stuck, and if needed, the researchers. As families got comfortable with the basics, we taught them more advanced rules involving multiple blocks, like creating blocks in response to user input, deleting blocks in response to collisions, how to create a "loop" pattern, etc. We also suggested common game design patterns, without telling children how to implement them. We took care to avoid saving rules created by us while teaching, by starting a new project without saving the current one.

Each location was set up for a duration negotiated with the community center employees (1-2 hours), and families were free to come and go as they pleased. Parents sometimes had to leave to take care of their younger children in the adjoining space (as we had arranged for childcare services). Providing this flexibility meant that the families did not start and end sessions in lockstep, leading to different session lengths. Each session had four to five researchers present as facilitators, who kept field notes and provided help when requested. Our system was instrumented to record all keyboard and mouse activity, which we were able to replay in order to generate hi-fidelity video of our participants'

Code	Explanation
Create Rule	A child or parent created a rule
Sharing w. Neighbors, Showing Off	Inviting nearby users to come see/play what they made
Taking Turns	Parent/child taking turns driving
Point / Touch Screen	Finger pointing at, or touching the screen
Success / Cheering	Moments when children/parents go "Yes!" or visibly celebrate; children turn to parent, smile ("Look! I did it!")
Parent Teaching Child	Parent explaining to or guiding child
Child Teaching Parent	Child explaining to or guiding parent
Researcher Guidance	One of us helps a child or parent

Table 4. A subset of our video analysis codes

screen activity, and the artifacts that they had created. In each session, we recorded audio and video footage of all participants [28] as well as their screens (Table 2).

Though we recorded audio of our participants, people at these community centers spoke Amharic, Afar, Arabic, Somali and French, and we did not have translators for all five languages. While the children were able to convey visually concrete instructions to their parents, they could not have translated nuanced interview questions and answers. This meant that we did not do follow-up interviews or transcriptions of the audio recordings of our participants. However, Tucker et. al. [51] showed that exclusively visual analysis of interactions has high inter-rater reliability with audio-visual analysis of the same for coding participant engagement. Based on their findings, we were able to visually analyze our ELL families' interactions to code for engagement, without translating their non-English dialogue.

Data Analysis

Our analysis examined inter-participant interactions for JME and their created artifacts for evidence of learning. We aimed to characterize the kinds of JME, if any, that were in these sessions, and also build a nuanced understanding of computational complexity in artifacts that families created.

JME analysis

To identify JME, we examined the video recordings and the available screen captures through thematic analysis [7]. First, researchers (two per video) open coded the data to produce a set of initial codes. They then iteratively reviewed the data, combining initial codes and adding new codes discovered in the process to produce the final codebook (Table 4 shows examples). Next, the 1st and 2nd authors utilized affinity diagramming to collaboratively group all codes into consistent themes, using Takeuchi and Stevens' [49] work as a reference for JME aspects.

Computational analysis

Prior work has characterized the learning of coding concepts by examining the scope of learners' use of programming constructs [11,32,54]. To assess the extent to which children and parents co-learned BlockStudio, we followed a similar strategy and analyzed their BlockStudio artifacts in three different ways. First, we analyzed Rule Types, aiming to provide a low-level picture of the *kinds of rules* within an artifact. Second, we analyzed Artifact Types to broadly *characterize the genre of the entire artifact*. Third, we analyzed Pattern Types to highlight the *game mechanics and computational patterns* (if any) in their work. In each of these categories, we report the presence or absence of a category, instead of counts. Since our participants used BlockStudio for differing amounts of time, actual counts would not be comparable across participants. However, creating a certain type of rule, artifact or pattern at all is categorical evidence of engaging with computation in BlockStudio, particularly because there were many ways to use BlockStudio without creating rules at all (e.g., creating a static arrangement of blocks). The 1st author and 2nd authors independently performed these three analyses on all the artifacts from these workshops, and then later compared their results. Any mismatches were resolved by referring to the video and the saved artifacts (where available). In theory, this analysis can be automated. However, since the artifacts from Bulsho center were not saved, we had to watch the screen capture videos to see what kinds of rules, artifacts and mechanics each family did (or did not) create.

Rule Types: We looked for touch, keypress, and collision rules created by each family. We ignored rules with default actions (see Table 1) and rules created with direct help from or by a facilitator while teaching. Rules were considered "complex" if they involved multiple kinds of blocks; otherwise, they were called "simple." An example of a simple rule is "when the smiley face is clicked, move that smiley face left," while a complex rule example is "when the smiley face is clicked, make it emit a bird flying left." The latter rule involves multiple block types because it is triggered by clicking the smiley face and it leads to the creation of a bird. We classified all non-default collision rules as "complex" because such rules are similar to an "if then" or a "switch case," selecting one of multiple execution paths depending on the colliding blocks. Other complex rule examples include deleting one of the colliders, generating blocks upon collision, and loading a different screen of blocks. For every family, we tracked whether they had (or had not) created simple or complex rules of each of the three types (touch, keypress, and collision).

Artifact Types: While rule categorization provides a low-level look at each family's vocabulary of rules, artifact categorization aims to provide a high-level picture of the kinds of artifacts created by each family. Looking at all artifacts across the three sessions, the 1st and 2nd author clustered them into three groups based on the overall genre of each artifact. These groups emerged through a joint inductive-deductive approach [7]. For every family, we tracked whether they had (or had not) created an artifact that belonged to each of these groups.

Artifact Patterns: To provide yet another lens into computational sophistication inside an artifact, we looked for design patterns that commonly occur in video games. From the design patterns, we suggested to the children during the workshops (including a standard BlockStudio computational pattern called a "loop"), we developed a set of patterns to look for within their artifacts (Table 6). For every family, we deductively tracked whether they had (or had not) constructed an instance of a pattern from each category. We describe these patterns and examples in our next section.

CASE STUDY RESULTS

Our workshop design did not guarantee that families would demonstrate productive JME by engaging in creating interactive, computational artifacts with BlockStudio. As Roque et. al. [44] reported, parents sometimes "feel unsure of what roles they can play to support their children and how their current supportive practices translate in the context of computing." Parents could have passively watched while their children used the system. Alternatively, families could have assembled blocks in static arrangements, showing JME in using BlockStudio, but employing it as a painting application, instead of as a coding environment. Others might have created multiple rules, but without any overall understanding of how rules could work together. Our analyses showed, these ELL families jointly engaged in a variety of ways, making interactive artifacts with computational complexity.

Evidence of Joint Media Engagement

Co-creation and mutual engagement. In our sessions, families jointly engaged in the *co-creation* of artifacts using the BlockStudio system in numerous ways. We saw *mutual engagement*, where children and parents took turns using the mouse to move blocks, create rules, and author interactive behavior. They also gestured, pointed at, and directly touched the screen to offer guidance and to discuss the different aspects of their artifacts. For instance, Dalmar (F3) caught a mistake his mother was about to make: her spaceship motion rule would have moved the spaceship diagonally instead of straight down. Using a combination of pointing to the screen and verbal guidance in Somali, he helped her fix it. Figure 4(a) shows Galad (F1) explaining to his mother how a "loop" pattern works, using his finger to trace the mechanism. Importantly, parents and children spoke to each other in non-English languages, providing support for our idea that a text-free and visually concrete design could facilitate JME using non-English languages, letting ELL families mutually co-create.

Flexible assignment of roles. We designed BlockStudio to allow children to learn coding concepts in English, and then teach their parents in their own language. Our intentions were to create conditions conducive to Teacher and Learner roles [3] for children and parents respectively, providing a context for JME. At each of the three locations, we saw children teach their parent using non-English. At Rajo center, Safiyo (F5) taught her father in Amharic how to create rules, as he did not understand any English (Figure 4c). At Bulsho center, Dalmar (F3) guided his Somali-speaking mother through creating rules in Somali. They eventually created a two-player game where one player (the mother) would fire obstacles, which the other player (the son) had to dodge. In some families, these roles did not map onto the

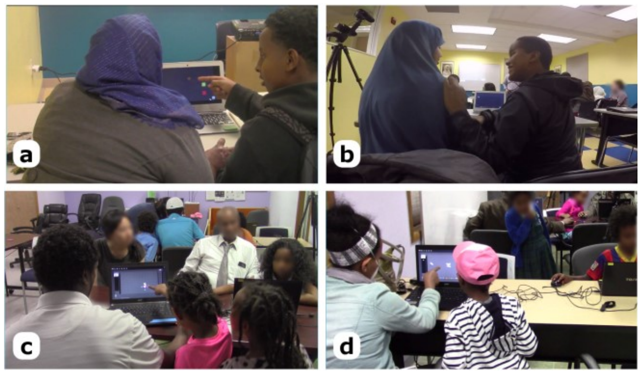

Figure 4: (a) F1: Galad explaining a loop mechanic to his mother, (b) F3: Dalmar celebrating with his mother, (c) F5: Safiyo teaching her father how to create firing pattern, (d) F8: Falis guiding her daughter to create a rule (child does not understand English). Names are pseudonyms.

parent and child as anticipated; they were opposite. At Rajo center, Falis (the parent in F8) guided her daughter Saado, who had emigrated from Ethiopia two weeks prior to the workshop and did not understand any English (Figure 4d). This family benefited from BlockStudio being text-free and visually concrete, allowing a parent with limited English fluency to teach her child using Amharic.

Parents sometimes provided skills not directly related to coding but important for problem-solving. At Farxad center, Ummi's father (F10) had some prior experience with programming and was fluent in English. He kept encouraging Ummi to think of the "challenge" in the maze game she was making. Eight-year old Danabo (F6) often impatiently clicked the "✔" button before demonstrating the change in the blocks, prematurely ending her rule demonstration (thus creating a default empty rule). Danabo's father had grasped the correct sequence, so he encouraged her to slow down and do the three steps in order. These unexpected variations showed BlockStudio to be usable via role assignments other than what we had planned for, suggesting that our two design principles can accommodate different scenarios for JME among ELL families.

Success, celebration and self-efficacy. There were numerous instances where the ELL families celebrated success at creating rules, making games, and beating the games they made. Their celebrations involved physical affection and smiling. At Bulsho center, Dalmar (F3) hugged his mother when she created her first rule (Figure 4b), while at Rajo center, Danabo (F6) high-fived her father when they successfully finished creating a game. At Farxad center, Nafiso's mother (F13) even took pictures of the screen showing her seven-year old daughter's creation. Our observations do not imply that all BlockStudio users will celebrate successes. However, among ELL families jointly engaging in coding using BlockStudio, many of them made progress that they considered a positive result worth celebrating. Early successes with graphics and coding have been linked to increased self-efficacy regarding computers

Artifact	Explanation
Static	No rules, blocks arranged as a static scene (e.g.: a car).
Animation	Blocks with velocity, relying on default "bounce" to create animated artifact (e.g.: block bouncing around)
Interactive	At least one block that player can move on screen, either using cursor keys, keyboard, or by clicking other blocks.

Table 5. The types of artifacts we observed.

[21], hence such celebrations are noteworthy outcomes. At the end of each session, multiple parents asked for the name of BlockStudio (it was not publicly accessible), presumably to use it on their own later. Finally, even though some families arrived late, all participants stayed until the end of every session, showing more evidence of high engagement.

Evidence of Computational Complexity

In the following section, we describe the results to our analysis of the rule vocabulary, the overall artifact type, and the presence of certain patterns like game mechanics and loops in our families' games.

Rule types. All participants created multiple kinds of rules in BlockStudio (see "Rule Types" column in Table 7), which highlights two important aspects. First, since we ignored the default (trivial) rules, our results indicate that every family created some non-default rules, which meant that they had to engage with the system and modify the blocks on screen. This is significant because if most families were unable to create non-default rules, then that would have been a failure of the interface, and by extension, an invalidation of our design principles. Engaging with the system to create rules is also important because success with programming helps nurture positive self-efficacy beliefs regarding computers [21].

Second, separating rules into simple (affecting one kind of block) vs. complex (affecting multiple kinds of blocks) is analogous to classifying code as modifying one object vs. modifying multiple objects. From a SOLO Hierarchy [33] perspective, the ability to create simple or complex rules can be seen as reflecting *unistructural* vs. *multistructural* understanding of BlockStudio's programming model. Simple rules create simple mechanics (like clicking a block to move that block), while complex rules can achieve more sophisticated mechanics (e.g., clicking one block to create a new block next to it, or "firing"). As Table 7 shows (first three columns), several families created complex rules of multiple types. For instance, at Farxad center, Ummi (F10) created a maze game with four arrows controlling the motion of a smiley face. She created a complex touch rule for each type of arrow that could be clicked, reflecting the multistructural understanding that an event (mouse click) triggered on one block (arrow) could modify another (smiley).

Artifact types. Our analysis of the artifact types revealed three genres of artifacts: static, animated and interactive. Table 5 shows these categories, while Table 7 summarizes our classification of artifacts ("Artifact Types" column). At Farxad center, Nafiso (F13) drew a static scene depicting a church by assembling blocks on the screen (Figure 5b).

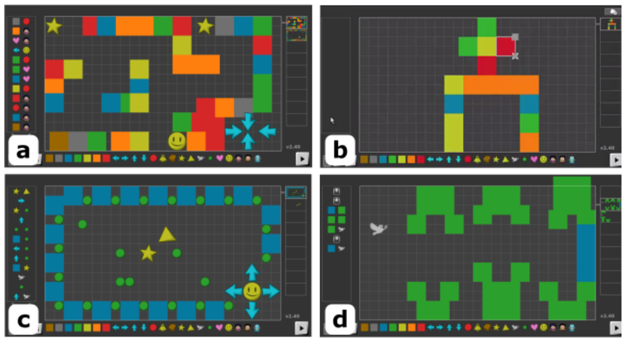

Figure 5: (a) maze (b) static artifact (church), (c) game with green dots (collectibles) (d) clone of Flappy Bird™

Pattern	Explanation
Collectibles	Collect items (e.g., PacMan™ has to eat the dots).
Obstacles	There are blocks the player must avoid (e.g., a frog has to cross the road, avoiding cars).
Firing	Player can create blocks (e.g., spaceship fires pellets).
Transition	A screen is loaded in response to an event (e.g. when character hits obstacle, the "Lose" screen shows).
Loop	Combining pre-placed blocks with rules, repeating collisions can automatically perform an action.

Table 6. The types of patterns we observed.

Ruqiyo and her mother (F11) created an animated artifact, where a character bounced around within an enclosure made up of other blocks. Ummi (F10) created a non-interactive animation that repeatedly created stars. Every participating family created one or more interactive artifacts (middle column in Table 7), indicating a productive outcome for all ELL families in our workshop. For instance, at Rajo center, 9-year old Dayib (F7) created an interactive artifact where the user could move a spaceship around and fire at other blocks. Other interactive artifacts included maze games, games similar to PacMan™ and Flappy Bird™, and variations of Space Invaders™. Creating interactive artifacts indicates an understanding of how to create multiple complex rules using BlockStudio. BlockStudio's visually concrete design allows blocks to depict static objects, be computational components, or both. Hence, a superficial, *prestructural* understanding of BlockStudio could explain creation of static artifacts without rules, or animations with default bounce rules.

Pattern types. Neither the rule-level nor the artifact-level analysis could capture the higher levels of understanding present in artifacts combining multiple complex rules to create higher-level behavior. For instance, at Farxad center, Ruqiyo (F11) created an artifact (Figure 5c) with five rules: four rules allowing the player to move a character, and one rule to let the character "eat up" green dots. Our rule-level analysis reports that the above artifact has four complex keypress rules and one complex collision rule. Our artifact categorization calls this an interactive artifact. However, what both of the above do not capture is that this artifact is a game where the player can move a character around, eating up the green dots. Our artifact pattern analysis labeled this the "collectibles" game mechanic. Table 6 lists the pat-

Family ID	Rule Types: Touch, Key, & Collision ●: simple rule, ★: complex rule			Artifact Types: Static, Animated, & Interactive			Patterns Types: Collectible, Obstacle, Firing, Transition, & Loop				
	T	K	C	S	A	I	C	O	F	T	L
F1	★	●	★		☑	☑		☑	☑		☑
F2		●	★			☑		☑		☑	☑
F3	★	●	★			☑		☑	☑		
F4	●	●★				☑					
F5	●	●★		☑		☑			☑		
F6	●	●	★	☑	☑	☑	☑				
F7	●★	●★	★	☑	☑	☑			☑		
F8	●★	●				☑			☑		
F9	●★	●	★			☑	☑	☑			☑
F10	●★		★		☑	☑		☑			☑
F11	★		★		☑	☑	☑	☑			
F12	●	●	★			☑	☑	☑		☑	
F13	●★			☑	☑	☑					

Table 7. Computational analysis of each family's creation.

terns we saw, while Table 7 shows each family's creations (see "Pattern Types" column). Though we suggested some of these mechanics or patterns to the families, our analysis only counted the instances where a family successfully created a pattern on their own. Our participants created multiple complex rules in order to implement these mechanics inside their artifacts. For example, Barkhad and Barni (siblings in F9) created an interactive artifact with collectibles and obstacles, using multiple complex rules to implement these patterns. Two families created the transition pattern, where a goal or obstacle was set up such that reaching it loaded a different screen. Fahmo (F2) and her mother created three successively harder obstacles for a character to clear, where touching these obstacles would transition to a "Lose" screen. Kamal (F12) made a game similar to Flappy Bird^{TM}, where avoiding obstacles and reaching a goal allowed the player to transition to the next level (Figure 5d). A powerful pattern in BlockStudio is the "loop" pattern, analogous to a "for" or "repeat" statement in text-based coding. By setting up a block to bounce between two other blocks, collisions can be used as a repeating event to trigger changes to blocks. Ummi (F10) created an animation that repeatedly created stars using this pattern. Galad (F1) guided his mother through creating a loop that automatically spawned rocks, which acted like obstacles. Many families created one or more of these patterns, indicating some relational understanding of how to combine rules in BlockStudio's coding paradigm to create such patterns.

DISCUSSION

Forms of JME like parental support can be beneficial for children's learning [17,24] and helpful when children are learning to code [1]. With the goal of empowering ELL families to jointly engage in learning to code, we distilled two design principles (text-free, visually concrete), instantiated them in BlockStudio, and studied its use with ELL families at community centers. Our thematic analysis revealed multiple forms of JME, with parents and children mutually engaging and co-creating, flexibly taking on different roles, and celebrating successes. Our artifact analysis showed a range of creations, varying in complexity from static arrangements of blocks, to interactive games with obstacles and level transitions. We now discuss four possible interpretations of our findings.

One interpretation of these outcomes is as positive validation for our two design principles, implying that a text-free and visually concrete interface supports JME among ELL families for coding. JME outcomes like flexible roles [3] could be attributed to our text-free interface being dependent on English instruction in our workshop sessions, necessitating the person with a higher level of English fluency to be the teacher. Seeing parents provide other forms of guidance can also be interpreted as an affordance of our system design, which replaces the clutter of text-based code (or text-labeled code blocks) with the visual appearance of the artifact itself. This lack of clutter may have helped adults support their child's thinking about the process and goals, instead of forcing parents to contend with the low-level details of a profusion of code blocks on the screen. The JME indicators we observed, like co-creation, flexible roles, and celebrations, could be seen as a corroboration of Dasgupta and Hill's [10] point that allowing people to engage in their own language could be the main advantage of localization. Our findings could indicate these design principles not only support ELL families but might also provide additional entry paths to coding, besides translated versions of existing interfaces [52].

Second, it is possible that the text-free nature of BlockStudio was inessential and that other factors led to the JME and artifact complexity we observed, such as BlockStudio's overall interaction design. Children could have taught their parents how to use a text-based coding interface in their native language. However, given that our contexts had families speaking two or more non-English languages, it would not have been feasible for us to effectively teach multiple translated versions of the same text-based coding interface in a single session. Further, English text could have allowed a single instructor to teach the children, but their parents would have been forced to learn a text-based interface in an unfamiliar language. However, the various creations at these workshops showed that our text-free design did not hinder these ELL families. Instead, they created interactive artifacts with sophisticated patterns, evincing productive JME. A follow-up study could focus on the impact of text via text-based and text-free versions of BlockStudio.

A third interpretation of our results is that the families we observed may have had cultural propensities for JME. Since we recruited all of our ELL families from community centers serving East African populations, they shared a similar cultural background. It might be the case that such families habitually spend time together, irrespective of the shared activity. If this were the case, and if our ELL parents were not engaged, we might still expect them to support their children and patiently watch them use the system, even if these adults did not understand what was going on. However, our findings demonstrated otherwise; our JME observations showed stronger forms of engagement than watching or observing. We saw turn-taking as well as pointing at and touching the screen, behaviors not associated with a passive observer. Studying ELL populations from different cultural backgrounds would help address this uncertainty.

Finally, most of our ELL families could have had prior coding experience (the father in F10 seems to be the only one who did). We did not collect self-reported prior experience, due to conflicting accounts of self-reported programming ability being a good and bad predictor of performance [12,18]. What we do know is that our participants had never seen the BlockStudio system and they all produced interactive artifacts with computational complexity via minimal instruction in around two hours.

Limitations

Our research was a small case study with 30 participants. Our contexts were community centers, not labs, and so there was considerable variability between sites. Therefore, our findings are focused on theoretical possibilities, rather than statistical generalizations [55]. We encountered issues like games not being saved at Bulsho center, and the log streaming failing to work at Rajo center. However, due to data redundancy, we were able to recreate games from logs (at Bulsho) and view the on-screen interaction using the video (at Rajo). Hence these issues did not diminish the quality of the data. The community setting could have caused a selection bias. The instructor and researchers (authors) not only taught these families, but they also spoke to them, sometimes sat beside them as they worked, and helped them when they got stuck. This interacted with what families learned, making the role of BlockStudio less clear. Overall, our observations did give us deeper insight into how ELL families interact [28]; however, as with any qualitative research, our work is prone to researcher bias.

DESIGN IMPLICATIONS

Programs like FCL have existed since 2013 [44]. Roque et. al. described how using the Makey Makey [5] in FCL allowed participation by a mother who "never saw herself playing a significant role in technology-related projects." ([45], p. 666) Such family-oriented programs could conceivably be augmented via text-free and visually-concrete coding tools like BlockStudio, thus expanding their reach to include ELL families. At each community center, we taught families how to create rules, and then allowed them to come up with ideas for how to use what they had learned. An alternate way for people to learn a new medium is by taking existing artifacts created by other users, then figuring out how they work [11]. When seeing somebody else's artifact, it can be difficult to figure out how it works, which is the focus of program comprehension [53]. Text-based coding often relies on comments, which are pieces of explanatory text next to a line of code, serving as documentation of how the code works. Similar to the notion of text-free curriculum, text-free documentation of code could enable program comprehension in BlockStudio, paving the way for ELL families to learn coding by remixing existing artifacts.

Though initial success with coding is helpful, learners can acquire longer-term benefits by continuing to develop their skills. This raises the question of how to support users in transitioning to their next coding environment, like Scratch, Python, or JavaScript. Since other environments are based on English text, there are questions about whether our design principles can be partially relaxed. Though being text-free appeared to show high ease of use in the initial stages, the BlockStudio system is intended to be used as a stepping stone to more complex systems. Thus, one of the most important questions one can ask of our system is "where do I go next?" BlockStudio is a standalone system now, but we intend to explore how to support children in transitioning and expanding their coding skills. Beyond community centers, appropriately-designed online curriculum could empower ELL families to jointly code at home without an instructor. Given BlockStudio's text-free design, its curriculum could also be free of text. While there are examples of static assembly manuals, like IKEA™ and Lego™, we are not aware of other text-free curriculum for a programming language, making this an interesting research topic.

CONCLUSION AND FUTURE WORK

In this paper, we have presented two design principles (text-free, visually concrete), a system implementing them, and a case study showing JME among ELL families learning to code using this system. Our discussion shows varied possibilities for extending this new understanding of how coding environments may empower and support such underserved populations in learning to code together. We encourage the community to investigate making family-based programs more accessible to ELL families, building text-free curriculum to teach coding at scale, supporting text-free program comprehension techniques, and finally, finding ways to support children as they transition from one coding environment to the next.

Our study focused on ELL families, but they are not alone in facing difficulties with English text. Neurodiverse people (e.g., those with dyslexia) experience challenges dealing with written text. The average reading ability of deaf children graduating high school is roughly at the third to fourth grade level [38]. Thus, the removal of text from coding could be useful beyond ELL populations. At the same time, it is not evident whether these same principles would be successful in empowering neurodiverse people to engage in coding, with or without family-based JME, providing another impactful avenue for future work. With these efforts, we can achieve more inclusive and diverse learning communities, and ultimately a more computing literate world.

ACKNOWLEDGEMENTS

The authors wish to thank all immigrant community partners, families and children that helped us with this study. This material is based upon work supported by the National Science Foundation under Grant 1314399.

REFERENCES

1. Uzi Armon. 1997. Cooperative parent-child learning in a LEGO-Logo environment. Retrieved September 13, 2017 from http://eurologo.web.elte.hu/lectures/armon.htm.
2. Rahul Banerjee, Jason Yip, Kung Jin Lee, and Zoran Popović. 2016. Empowering children to rapidly author games and animations without writing code. In *Proceedings of the 15th International Conference on Interaction Design and Children* (IDC '16), 230-237. https://doi.org/10.1145/2930674.2930688
3. Brigid Barron, Caitlin Kennedy Martin, Lori Takeuchi, and Rachel Fithian . 2009. Parents as learning partners in the development of technological fluency. *International Journal of Learning and Media* 1, 2 (May 2009), 55-77. https://doi.org/10.1162/ijlm.2009.002
4. Aaron Bauer, Eric Butler, and Zoran Popović. 2017. Dragon architect: Open design problems for guided learning in a creative computational thinking sandbox game. In *Proceedings of the 12th International Conference on the Foundations of Digital Games* (FDG '17), Article 26, 6 pages. https://doi.org/10.1145/3102071.3102106
5. Beginner's Mind Collective and David Shaw. 2012. Makey Makey: improvising tangible and nature-based user interfaces. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (TEI '12), Stephen N. Spencer (Ed.), 367-370. https://doi.org/10.1145/2148131.2148219
6. Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. In *Proceedings of the Fifth Annual CCSC Northeastern Conference on the Journal of Computing in Small Colleges* (CCSC '00), John G. Meinke (ed.). Consortium for Computing Sciences in Colleges, 107-116.
7. Juliet Corbin and Anselm Strauss. 2014. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE
8. Allen Cypher and Daniel Conrad Halbert (eds). 1993. *Watch What I Do: Programming by Demonstration*. MIT press.
9. Allen Cypher and David C. Smith. 1995. KidSim: end user programming of simulations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '95), 27-34. https://doi.org/10.1145/223904.223908
10. Sayamindu Dasgupta and Benjamin Mako Hill. 2017. Learning to code in localized programming languages. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale* (L@S '17), 33-39. https://doi.org/10.1145/3051457.3051464
11. Sayamindu Dasgupta, William Hale, Andrés Monroy-Hernández, and Benjamin Mako Hill. 2016. Remixing as a Pathway to Computational Thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (CSCW '16), 1438-1449. https://doi.org/10.1145/2818048.2819984
12. Oscar Dieste, Alejandrina M. Aranda, Fernando Uyaguari, Burak Turhan, Ayse Tosun, Davide Fucci, Markku Oivo, and Natalia Juristo. 2017. Empirical evaluation of the effects of experience on code quality and programmer productivity: An exploratory study. *Empirical Software Engineering* 22, 5 (February 2017), 2457-2542. https://doi.org/10.1007/s10664-016-9471-3
13. Betsy DiSalvo, Cecili Reid, and Parisa Khanipour Roshan. 2014. They can't find us: the search for informal CS education. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (SIGCSE '14), 487-492. http://dx.doi.org/10.1145/2538862.2538933
14. Andrea A. diSessa and Harold Abelson. 1986. Boxer: A Reconstructible Computational Medium. In *Communications of the ACM 29, no. 9* (1986), 859-868. https://doi.org/10.1145/6592.6595
15. Allison Druin. 1999. Cooperative inquiry: developing new technologies for children with children. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (CHI '99), 592-599. http://dx.doi.org/10.1145/302979.303166
16. Benedict du Boulay. 1989. Some difficulties of learning to program. In *Studying the Novice Programmer*, E. Soloway & J.C. Spohrer (eds.). Lawrence Erlbaum, Hillsdale, NJ, USA, 283-289.
17. Paul G. Fehrmann, Timothy Z. Keith, and Thomas M. Reimers. 1987. Home influence on school learning: Direct and indirect effects of parental involvement on high school grades. *The Journal of Educational Research* 80, 6 (July 1987), 330-337. https://doi.org/10.1080/00220671.1987.10885778
18. Janet Feigenspan, Christian Kastner, Jorg Liebig, Sven Apel, and Stefan Hanenberg. 2012. Measuring programming experience. In *2012 IEEE 20th International Conference on Program Comprehension* (ICPC '12), 73-82. https://doi.org/10.1109/icpc.2012.6240511
19. Louise P. Flannery, Brian Silverman, Elizabeth R. Kazakoff, Marina Umaschi Bers, Paula Bontá, and Mitchel Resnick. 2013. Designing ScratchJr: support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children* (IDC '13), 1-10. http://dx.doi.org/10.1145/2485760.2485785
20. Ephraim P. Gilnert and Steven L. Tanimoto. 1984. Pict: An interactive graphical programming environment. *Computer* 17, 11 (1984), 7-25.

21. Bassam Hasan. 2003. The influence of specific computer experiences on computer self-efficacy beliefs. *Computers in Human Behavior* 19, 4 (July 2003), 443-450. https://doi.org/10.1016/s0747-5632(02)00079-1
22. Donald J. Hernandez, Nancy A. Denton, and Suzanne E. Macartney. 2007. *Children in Immigrant Families-The US and 50 States: National Origins, Language, and Early Education*. Research Brief Series. Publication# 2007-11. Child Trends.
23. Donald J. Hernandez, Nancy A. Denton, and Suzanne E. Macartney. 2008. Children in immigrant families: Looking to America's future. *Social Policy Report* 22, 3 (2008), 1-24. Society for Research in Child Development.
24. William H. Jeynes. 2005. A meta-analysis of the relation of parental involvement to urban elementary school student academic achievement. *Urban education* 40, 3 (May 2005), 237-269. https://doi.org/10.1177/0042085905274540
25. Joel Jones. 2003. Abstract syntax tree implementation idioms. In *Proceedings of the 10th Conference on Pattern Languages of Programs* (PLoP2003).
26. Ken Kahn. 1996. Toontalk TM—an animated programming environment for children. *Journal of Visual Languages & Computing* 7, 2 (June 1996), 197-217. https://doi.org/10.1006/jvlc.1996.0011
27. Rosemin Kassam, Régis Vaillancourt, and John B. Collins. 2004. Pictographic instructions for medications: do different cultures interpret them accurately? *International Journal of Pharmacy Practice* 12, 4 (December 2004), 199-209. https://doi.org/10.1211/0022357044698
28. Barbara B. Kawulich. 2005. Participant observation as a data collection method. *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research*. Vol. 6. No. 2. 2005.
29. Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys* (CSUR), 37, 2 (June 2005), 83-137. https://doi.org/10.1145/1089733.1089734
30. Finn Kensing and Jeanette Blomberg. 1998. Participatory design: Issues and concerns. *Comput. Supported Coop. Work* 7, 3-4 (January 1998), 167-185. http://dx.doi.org/10.1023/A:1008689307411
31. Amy J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 59-66. IEEE. https://doi.org/10.1109/vlhcc.2010.17
32. Kyu Han Koh, Ashok Basawapatna, Vicki Bennett, Alexander Repenning. 2010. Towards the automatic recognition of computational thinking for adaptive visual language learning. In *2010 IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC), 59-66. https://doi.org/10.1109/vlhcc.2010.17
33. Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L. Whalley, and Christine Prasad. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *In Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (ITICSE '06), 118-122. http://dx.doi.org/10.1145/1140124.1140157
34. Sonia Livingstone, Leslie Haddon, Leslie, Anke Görzig, and Kjartan Ólafsson. 2011. *Risks and safety on the internet: the perspective of European children: full findings and policy implications from the EU Kids Online survey of 9-16 year olds and their parents in 25 countries.* Deliverable D4. EU Kids Online, London, UK.
35. Jane Margolis, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme, and Kim Nao. 2010. *Stuck in the Shallow End: Education, Race, and Computing*. MIT Press
36. Indrani Medhi, Aman Sagar, and Kentaro Toyama. 2006. Text-free user interfaces for illiterate and semiliterate users. *International Conference on Information and Communication Technologies and Development* (ICTD '06), 72-82. IEEE. https://doi.org/10.1109/ICTD.2006.301841
37. Sharan B. Merriam. 1988. *Case Study Research in Education: A Qualitative Approach*. Jossey-Bass.
38. D. Morere. 2011. Reading research and deaf children. *Visual Language and Visual Learning Science of Learning Center*, Research Brief No. 4.
39. Greg L. Nelson, Benjamin Xie, and Amy J. Ko. 2017. Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (ICER '17), 2-11. https://doi.org/10.1145/3105726.3106178
40. Laura M. Padilla-Walker, Sarah M. Coyne, and Ashley M. Fraser. 2012. Getting a high-speed family connection: Associations between family media use and family connection. *Family Relations* 61, 3 (June 2012), 426-440. https://doi.org/10.1111/j.1741-3729.2012.00710.x
41. Alexander Repenning. 2013. Making programming accessible and exciting. *Computer* 46, 6 (June 2013), 78-81. https://doi.org/10.1109/mc.2013.214
42. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: programming for all. *Communications of the ACM* 52,

11 (November 2009), 60-67. https://doi.org/10.1145/1592761.1592779

43. Ricarose Roque. 2016. Family creative learning. In *Makeology: Makerspaces as Learning Environments*, Volume 1, Kylie Peppler, Erica Halverson, and Yasmin B. Kafai (eds.). Routledge, 47-63.
44. Ricarose Roque, Karina Lin, and Richard Liuzzi. 2015. Engaging parents as creative learning partners in computing. *Exploring the Material Conditions of Learning* 2, 687-688.
45. Ricarose Roque, Karina Lin, and Richard Liuzzi. 2016. “I’m not just a mom”: Parents developing multiple roles in creative computing. In *Proceedings of the International Society of the Learning Sciences* (ISLS '16), 663-670. https://dx.doi.org/10.22318/icls2016.86
46. Seattle Times Staff. 2010. Seattle’s Rainier Valley, One of America’s ‘Dynamic Neighborhoods.’ *The Seattle Times*. Retrieved September 13, 2017 from http://www.seattletimes.com/opinion/seattles-rainier-valley-one-of-americas-dynamic-neighborhoods/.
47. Sandra D. Simpkins, Pamela E. Davis-Kean, and Jacquelynne S. Eccles. 2005. Parents' socializing behavior and children's participation in math, science, and computer out-of-school activities. *Applied Developmental Science* 9, 1 (January 2005), 14-30. https://doi.org/10.1207/s1532480xads0901_3
48. Juha Sorva. 2012. *Visual program simulation in introductory programming education*. Doctoral Dissertation. Aalto University, Espoo, Finland
49. Lori Takeuchi and Reed Stevens. 2011. *The New Coviewing: Designing for Learning through Joint Media Engagement*. The Joan Ganz Cooney Center at Sesame Workshop. New York, NY.
50. Steven L Tanimoto. 2003. Programming in a data factory. In *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environment*, 100-107. https://doi.org/10.1109/hcc.2003.1260209
51. Laura Tucker, Rachel E. Scherr, Todd Zickler, and Eric Mazur. 2016. Exclusively visual analysis of classroom group interactions. *Physical Review Physics Education Research* 12, 2 (November 2016), 1-9. https://doi.org/10.1103/physrevphyseducres.12.020142
52. Aaron Tyo-Dickerson and Kristi Williams. 2017. ICS Addis and MIT Translate Scratch into Amharic. Retrieved September 13, 2017 from www.tieonline.com/article/2100/ics-addis-and-mit-translate-scratch-into-amharic.
53. Anneliese von Mayrhauser and A. Marie Vans. 1995. Program comprehension during software maintenance and evolution. *Computer* 28, 8 (1995), 44-55. https://doi.org/10.1109/2.402076
54. Seungwon Yang, Carlotta Domeniconi, Matt Revelle, Mack Sweeney, Ben U. Gelman, Chris Beckley, and Aditya Johri. 2015. Uncovering trajectories of informal learning in large online communities of creators. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (L@S '15), 131-140. http://dx.doi.org/10.1145/2724660.2724674
55. Robert Yin. 2013. Validity and generalizations in future case study evaluations. *Evaluation, 19*(3), 321-332.
56. Jason C. Yip, Carmen Gonzalez, and Vikki Katz. 2016. The learning experiences of youth online information brokers. In *Proceedings of the International Society of the Learning Sciences* (ISLS '16), 362-369. https://dx.doi.org/10.22318/icls2016.48
57. Jason C. Yip, Kiley Sobel, Caroline Pitt, Kung Jin Lee, Sijin Chen, Kari Nasu, and Laura R. Pina. 2017. Examining adult-child interactions in intergenerational participatory design. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17), 5742-5754. https://doi.org/10.1145/3025453.3025787