Learning to code, coding to learn: youth and computational thinking

Introduction

Professor Jeannette Wing's provocative and influential paper entitled Computational Thinking appeared in the March 2006 issue of Communications of the ACM; in the 13 years since, educators, computer scientists, policy makers, and technologists have been working to define this conceptual space, measure it, and assess the role that computer science can and should play in the education of young people. Although Wing is by no means the first person to notice that computer science can play an important role in developing problem solving capacities in youth across the curriculum (Papert, 1980; Clements and Gullo, 1984; Harel and Papert, 1990; diSessa, 2001, to name just a few), her call to arms fuelled increasing research attention and policy interest (Aho, 2012; Cooper and Cunningham, 2010; Guzdial, 2008; Wang, 2015, Wing, 2008; Wing and Stanzione, 2016). With this special issue of the Journal of Information and Learning Sciences, we propose that computational thinking (CT) is a generative space residing between the learning sciences and information sciences, drawing on concepts of cognition and development (e.g. motivation and self-regulation), the system sciences (e.g. algorithmic representation and design of data structures) and areas of shared or interdisciplinary concern and interest (e.g. digital literacy, problem-solving, making and creativity).

But, what is CT? And how does it relate to critical thinking, not to mention pressing issues of digital literacy, and children's readiness to engage with contemporary digital media? Succinctly, CT can be defined as “the process of recognising aspects of computation in the world that surrounds us, and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes” (Royal Society, 2012, p. 29). Further definitions, including those offered by the College Board and National Science Foundation, identify key thought processes and techniques that help us operationalize CT as a set of practices that can be taught and fostered, including abstraction and pattern generalization; systematic processing of information; symbol systems and representation; algorithmic notions of flow and control; structured problem decomposition; iterative, recursive, and parallel thinking; conditional logic; efficiency and performance constraints; debugging and systematic error detection (Grover and Pea, 2013). This list shares a number of common points with the curriculum taught in mathematics and science; however, there clearly are procedures and habits of mind, such as abstraction and debugging, that are distinctive and not likely to be taught in general K–12 courses (Wang, 2015). CT has some overlap with general problem-solving processes that form the basis of information literacy models as well, but the emphasis on understanding algorithmic flow and symbolic representation suggests significant differences. Scholars are making the case that CT is its own form of literacy, one that complements other knowledge domains, but occupies its own place in the curriculum (Barr and Stephenson, 2011; Guzdial, 2008).

In this editorial article, I will briefly, though not exhaustively, review research from diverse perspectives on CT, first from education and the learning sciences, then from library and information science. I will then opine on the challenges confronting society by applied computer science that lacks a consideration of diverse human values and propose a way forward that relies an ethically grounded education in CT. I conclude with brief summaries of the articles that represent this special issue and additional areas of future consideration that the editors look forward to seeing published in this journal and other venues.
**Perspectives from education and the learning sciences**

In recent years, the Computer Science Education (CSE) movement has gained considerable momentum, led by a coalition of scholars, non-profits and industry partners. Coding interfaces such as MIT’s Scratch platform, Gamestar Mechanic, Kodu, and a host of others (Anton and Berland, 2014; Resnick et al., 2009) have opened new possibilities for youth to develop their own interactive games. The “Computer Science for All” Initiative (CS4All) begun during the Obama Administration suggests that the USA is not far behind France, the UK and other nations in mandating coding for children beginning in the elementary grades. Programs and initiatives in the North American context that contribute to these efforts include Code.org, Hour of Code, and the work of organizations including BlackGirlsCode, GirlsWhoCode, iRemix, Code Savvy, Globaloria, KidsCodeJeunesse and others.

Scholars in formal and informal learning have been working to make computer programming more accessible to young people. According to a recent survey, coding is already a part of the formal curriculum of 16 countries in Europe (Balanskat and Engelhardt, 2015). Curricula in game design, such as those developed by constructionist scholars and instructional design experts Yasmin Kafai, Idit Harel and their colleagues (Kafai, Peppler and Chapman, 2009; Fields et al., 2012; Reynolds and Harel, 2011; Reynolds, 2016) have engaged thousands of young people across several US states in formal, intensive in-school introductory CSE coursework.

Although these programs show promise, their over-emphasis on “games” as the endpoint of computer programming may be perceived as limiting. The Raspberry Pi, for example, was developed in part as a response to computer science majors at a university having computer skills limited to gaming and web design and not understanding simple engineering concepts that make hardware and software work together. Coding is increasingly linked with making, a movement that has engaged curious learners of all ages and backgrounds. Making as a way of fostering CT often expands into physical computing, where software and hardware come together, and learners “tinker” with wires, sensors, LEDs, and other components. Learning through physical computing has the potential to generate a more holistic approach to CT, involving reasoning about a wider range of systems and processes. New curricula are emerging inspired by the confluence of coding and making (Honey and Kanter, 2013; Sheridan et al., 2014) and the “connected learning” work of Mimi Ito et al. (2013).

**Perspectives from library and information science**

Although CSE might be perceived as in a renaissance of sorts, the literature in library and information science has given only modest attention to CT as an outcome of library-based informal learning programs. This is in part due to the historic focus on young people as users of information systems, rather than creators of information systems. The emphasis on system use, and in particular search systems and social media services, often casts young persons in one of two lights. In the first, we have the glamorization of the youth technology user, the digital native, fettered by the constraints of school structures and adult perceptions, a “power user” by nature whose talents we are only beginning to understand (Prensky, 2001; Palfrey and Gasser, 2008). In the second, what some call the deficit model, young people are described in unflattering terms as digital “naives”, ignorant and cavalier users, the “Google Generation”, whose focus on expedience and ease of use often leads to poor information processing and diminished learning outcomes (Carr, 2011). The reality is somewhere between these two characterizations (Boyd, 2014; Meyers et al., 2013; Wagner, 2008).

One way that CT is gaining ground in LIS scholarship is through research on the role of makerspaces as a *locus* of creative and critical skill-building for youth. As libraries have adopted “making” as an informal learning opportunity to engage preteen and teens, the links to coding and computational skill-building have followed in work by several LIS
researchers (Abbas and Koh, 2015; Bowler and Champagne, 2016; Koh et al., 2019; Martin, 2015; Prato, 2017; Subramaniam et al., 2018). Informal science, technology, engineering and mathematics (STEM) education through public libraries can complement school-based computational initiatives, providing interest-driven engagement for youth of all ages. Of note, these scholars, particularly Bowler and colleagues, maintain a twin focus on youth empowerment and critical engagements with technology. This area of scholarship provides a fruitful area of cross-domain development for scholars in LIS and the learning sciences.

A challenge for learners of all ages is the “black boxing” of technology, which makes contemporary tools and media easy to use and yet difficult to comprehend. Few undergraduate students, for example, even in Masters of Information Science courses, understand the difference between Google search and the contemporary online library catalogue, even as the latter is trying to look and function like the former. CT, then, supports critical understanding at the intersection between system design and system use, providing the vital link to comprehending how information systems work.

Coding and design: Connecting the analogue and digital
To connect CT with students’ everyday lives, it can be useful to create functional analogies that help demystify computing. Computer science does not have to be taught exclusively with digital tools, as evidenced by the CS unplugged curriculum (https://csunplugged.org/) and related research (Taub et al., 2009). In my own teaching in information science, I often encounter students who are apprehensive about coding and lack awareness of how information systems work. To further elaborate on how we can break down the “black boxes” of technology, I offer the following example.

Elementary school students can be found constructing and using “cootie catchers” (also known as fortune-tellers or chatterboxes), folded paper manipulatives that function much the same as a “magic 8-ball” device. Students invariably recognize this object but may not connect it to information concepts immediately. The idea behind these paper devices is to craft a series of scenarios or “fortunes” that might describe any number of persons or contexts, much like the slip inside a fortune cookie. The paper hand manipulative serves as a means by which the fortunes are delivered to persons on the auspices that they “select” a fortune by indicating a personal detail, a favourite colour or number. What consistently comes as a surprise to learners is that this device is a simple prototype of an information system. The hand manipulative is the interface; the favourite number or colour selected by the user becomes input for the algorithm; the “fortunes” written in advance inside the flap of paper constitutes a database. Items in the database are called via the interface through individual interaction with the algorithm. In a matter of minutes, we can assign these sophisticated concepts to familiar aspects of a child’s social world, crafted from paper and markers.

The power of crafting can then be translated to a digital iteration of the “cootie catcher” system through the programming of a fortune-teller in Scratch, an object-based programming language readily available on the web or installed as an app. The power of CT is unleashed when the learner makes the connection between the manipulative and the code, the physical artifact and the digital system. This is what I seek to do with this lesson: learners connect crafts and activities with the concepts of systems design and development. In the process, learners are prompted to identify the shortcomings of the physical system (i.e. your database is limited to how many flaps your cootie catcher has) and how the digital system might overcome these challenges or create new ones. As my students explore the connections between physical and digital information systems, they come to understand how our contemporary systems work and the role that individual choices play in how systems function, both as designers and users. The “cootie catcher” database is just one example of how we can connect and unpack user behaviours and
information systems to promote CT. These activities foster reflection in combination with coding skills, so students can see what they are doing and why it makes sense. I argue that only in the process of building their own information systems can users make the black box of today’s information infrastructure transparent. In doing so, they find the middle ground between users and designers, coders and end-users.

(Not) Thinking like (certain) computer scientists

I teach children and adult students to code, and I believe that CT can be a powerful tool in creating future systems that support vibrant communities and rich human interactions. Yet, recent revelations regarding the use of computing platforms and technologies give me and others cause for alarm. It is worth taking a moment to interrogate the challenges we see in contemporary technologies and identify the place of CT in moving toward a future where technology supports, rather than hinders, human thriving.

“To a computer scientist, the solution to a bug is often just more computer science”, noted Brooke Borel (2018), writing about the challenges of “Deep Fake” video creation. Deep fake technology, or image synthesis using artificial intelligence, is emerging as a development both useful and pernicious. The product of deep learning algorithmic mash-ups of audio and video, the technology will soon produce near perfect video of anyone saying anything, provided there is sufficient high-quality material to train the system. Although this technology was intended to streamline cinema production and enhance long-distance communication, it could also be used by individuals, groups or governments to create propaganda, misleading or slanderous political ads, or personally damaging videos that could be uploaded to social media (Chesney and Citron, 2018). The computer scientists who invented these techniques, however, suggest figuring out how to regulate this technology is someone else’s problem. Or, they suggest, maybe we can invent algorithms to identify when someone is misusing the algorithms (Adler, 2018).

Our news cycle is now filled with dramatic and painful examples that illustrate how this particular computer science mindset – that the solution to emergent problems of code is the development of more and increasingly powerful counter-code – is fundamentally flawed. Algorithms contribute to the suppression of dissenting minorities, live-streaming of terror attacks, racial profiling and discrimination and efforts toward ethnic genocide (Eubanks, 2018; Noble, 2018; Buckley et al., 2019). Little did we realize from the cheeky early moto of Facebook – move fast and break things – that the “things” being broken or subverted might include privacy laws, cultural norms and human decency (Toyama, 2015; Taplin, 2017). Although it is also true that algorithms are solving problems, saving lives and creating cleaner, safer living conditions for citizens around the world, technology companies need to take ownership of the failures as well as the successes. We need to ensure that there is a strong ethical component to coding curricula that facilitates thinking through the consequences, intended or unintended, of computer science. Code is a proximate cause of the unfortunate consequences of its (mis)use, but like guns and cars, everyone can likely agree that powerful tools needs oversight and regulation, as well as training that establishes ethical applications thereof.

This is not to suggest that computer science as a discipline is ethically bankrupt; far from it. Many of us benefit daily from the technical advances of computation, and algorithms promise new ways of realizing human potential when applied to any number of pressing geopolitical issues, including climate change, sustainable agriculture, women’s rights, and health care to name a few. Nor am I conflating the discipline of computer science with technology industries, although evidence that portions of the academy are increasingly under the sway of corporations is a significant concern (Giroux, 2007). Yet, we have come to realize that the largest players in the technology industries, such as Facebook, Google and Amazon, are incapable of regulating themselves; they appear to have chosen to focus on profits at the
expense of human thriving (Hughes, 2019; Susser, 2019). These companies, among the largest, most valuable corporate entities on Earth, are also key players in the development and propagation of free coding curricula for schools and individuals. If you have experienced a coding massive online open course (MOOC) or series of lessons on Code.org, you will find the assumptions and ethos of Silicon Valley are baked in. To create a check against tendencies to code first and ask questions later, we need to ensure that human values are at the centre of our thinking about code, its use and its consequences.

Toward critical computational literacies

Code, like any human language, is not value neutral. Both the building blocks of computation as well as what we construct with those blocks carry human values. Those values may be efficiency, parsimony, safety or accountability. How we define those values and make sense of them is an essential part of coding education. Learning to use the language of code is also a value proposition, as code facilitates entering a conversation as well as a community of other coders. The ethics and values of that community play a part in how we perceive technology and each other in the process of its use and deployment in the world.

Several of the authors in this special issue push back on framings of CT, and I am encouraged by their development and use of different terms to describe how people think with computers and code for diverse outcomes. In particular, Wells and colleagues propose the term *procedural creativity* to describe the process of game-based engagements with youth. Recently, Lee and Soep (2016) have combined an emancipatory philosophy with CT to coin the phrase “critical computational literacy.” My own work has taken a turn toward critically engaging with questions of how we use technology and how coding is represented, both in curriculum that supports CT, as well as the popular imagination. Let me provide two examples.

Through a series of workshops administered in local public libraries, my students and I have been introducing preteens and teens to computer science concepts using the Raspberry Pi, taking advantage of its built-on support for introductory coding (Scratch, Python, and SonicPi) and physical computing (Meyers, 2016). We took these activities on the road to work with urban teens in an all-day workshop sponsored by the Free Library of Philadelphia. With colleagues from several universities, we combined a physical computing activity, a design workshop, social interaction and play to focus on social justice, and how technology could help them improve their communities (Fisher et al., 2017). Before jumping into the activity, we reflected on the digital cameras they use every day and how little they understood about how they work. Next, teens used readily available electronics (such as camera modules, jumper wires and bread boards) and a Python script to build a working camera using the Raspberry Pi as a base. Youth worked in pairs; adult facilitators roved and provided guidance. A set of printed instructions broke the project into steps and prompted the teens to check their progress along the way. The goal was to help students reflect on the process of making technology and empower/inspire them as designers, coders and makers. After the build, teens engaged in a design workshop where they envisioned prototype technologies that could contribute to a better community. Their designs were simply astounding. What struck the researchers was how many of the designs focused on building empathy and social understanding – seeing the world through their eyes – and making systems that fostered peace, justice and accountability.

Most recently, I have been analyzing children’s stories and media that incorporate computational elements, either implicitly or explicitly, in the plot of the narrative. For example, I interrogate Gene Luen Yang’s *Secret Coders*, a series of six graphic novels (published 2015–2018) that use learning to code as an essential element of its narrative structure. The novels follow Hopper, Eni and Josh, three preteen outsiders who initially use their growing knowledge of computer coding to learn about their mysterious school, Stately Academy. Eventually, these
skills are used to save the world from the nefarious Professor One-Zero and his formula for “true happiness.” I argue that Yang’s series contains elements that play to Silicon Valley’s Randian, pro-technology motives, while also subverting them, enabling readers to critically engage with the discourses of technology and its role in society (Meyers, 2019 forthcoming). This analysis is part of a larger examination of the rhetorics of computation present in children’s everyday media and draws in other texts such as popular cinema (e.g. Big Hero 6), picture books (e.g. Hello Ruby), and children’s games (e.g. GoldieBlox). As part of this investigation I ask the following: whose story is actually being told? How are children silenced or empowered through narratives extolling the power of computation? When discussing children’s texts, we often look for the “hidden adult” (Nodelman, 2008), by which we seek to unpack the author’s relationship to the subject and reader, and interpret stories as social texts. One might argue that the hidden adult in Yang’s graphic novels is an entire industry: Silicon Valley and its neo-liberal agenda, which is using coding education and rhetorics of computation as a kind of “political technology” to shape both minds and social systems (Meyers, 2019 forthcoming). In addition to critically reading and analyzing these texts, my students and I are developing techniques for co-reading/co-viewing/co-playing with these media. We hope through this work to bring a needed critical eye to computing education, as well as empower young people to see the element of computation in the world around them and both celebrate and interrogate their value.

The articles in this issue

CT, as a way of framing the what, how and why of coding and related 21st-century skills, is a popular and generative topic. Our call prompted a range of submissions covering work with younger children through post-secondary CSE, using both qualitative and quantitative methods, as well as conceptual approaches. In the following text, I give a brief introduction to each article in this issue and conclude with areas for future study, including those topics we wish we could have included here.

Matthew Wells and Jason Boyd push back on the term “computational thinking” in their article Generating Gameworlds: The Case for Procedural Creativity. They pose this question to readers: is CT the best way to describe the student outcomes we seek to achieve though coding and computational activities in and out of school? They note a key shortcoming of CT, especially as professed by computer science educators, is the emphasis on abstraction and solutionism. Their proposed framework, procedural creativity, is more closely aligned with the language-action perspective, and we can see in their elaboration how it shifts computational activity from the cognitive (rational) position to a situated, socio-cultural standpoint. They propose that game generation, as well as the use of interactive fiction, serves as an ideal platform for engaging this concept. Their work dovetails nicely with the work of Proctor and Blikstein described in the following text.

In agreement with Wells and Boyd, Chris Proctor and Paulo Blikstein illustrate how interactive narrative may serve both a pedagogical and emancipatory function in Unfold Studio: Supporting Critical Literacies of Text and Code. Readers are introduced to Unfold studio, an interactive storytelling platform designed to foster “textual-computational multiliteracy.” Through several workshops, first to design the platform and then to test it with a diverse group of high school students, Proctor and Blikstein used a designed-based research approach that takes us from localized innovation toward a replicable pedagogy of critical literacy. The authors use the case of one student, Leanne, and her composition “Angela: A Bystander’s Story” to unpack the potential of interactive storytelling to elucidate the interaction between discourse (language) and discourses (ideologies). The work also demonstrates how critical computational literacy can be situated in both formal and informal learning environments.
Also working in the classroom, Eben Witherspoon and Christian Schunn examine the relationship between teachers’ dispositions toward CT and student performance in robotics curricula. When teachers emphasize CT as a learning goal, students not only show greater knowledge gains but also maintain more positive attitudes toward programming over time. As teachers may come to computational curricula and skills through means other than traditional pre-service training in computer science (e.g. in-service training, workshops, or self-directed learning via open resources), it is vital that researchers pay attention to how teachers deliver lessons in coding and computational problem-solving, not just how student attitudes influence their reception. As I mention later in this paper, more attention is needed on teacher training, particularly for classroom educators and informal learning facilitators, such as librarians, who are already in the field.

Focusing on informal learning, Ricarose Roque and Natalie Rusk share qualitative insights on coders’ practices in Youth Perspectives on Their Development in a Coding Community. They interviewed eight deeply involved members of the Scratch community to understand their motivations and engagements in coding. The interviews emphasized memorable moments – those pivotal interactions, conversations or projects – that participants identified as key to their trajectories in the community and their development as “coders.” Although a handful of highly-engaged participants is not meant to represent a broader sample, these success stories give us an opportunity to analyze what works (and what does not) in a diverse online community of learners. The results show how feedback and recognition, as well as opportunities to mentor others, may drive achievement in online learning platforms, which is crucial to understanding the role of informal learning networks that support life-long, life-wide learning.

Paula Haduong unpacks the reluctance of teens to engage in coding education in “I Like Computers, I hate coding”: A Portrait of Two Teens Experiences. Using the technique of portraiture, Haduong brings greater depth and meaning to the disconnect young people may feel between everyday computer use for communication and creative expression and the seemingly onerous task of writing code. Exploring concerns for motivation in the design of informal computational learning opportunities, particularly for women and underrepresented minorities, the authors identify the diverse expectations youth bring to such programs and how identity and social support affect learner experiences. The two portraits of youth developed in this piece give us a great deal to think about in terms of how we might make computer science education more equitable. The authors suggest that the solution is not strictly about providing more access to coding for persons of colour; rather, there are systemic inequities that need to be overcome before some persons of colour will want to engage in coding education.

Balancing these intimate, qualitative portraits of young coders, we are delighted to include a mixed-methods analysis of young women coders and their longitudinal trajectories in STEM fields. Seeking to identify paths to “persistence” as a way of realizing greater equity in computational and STEM-oriented occupations, Joanna Weidler-Lewis, Wendy DuBow, Alexis Kaminsky and Tim Weston provocatively build an argument for compulsory coding education in K–12. Their work also reinforces the notion, albeit with different evidence than Hanuong, that persistence in coding is multifaceted and is spurred by person ambition, family dynamics and social support structures. Although many can agree that greater diversity in the technology industries would be an essential good, we must also consider diverse outcomes for people who want to study computer science, such as applying coding skills in education, public policy and innovation in the non-profit sectors.

On the quantitative side of the conversation, Ömer Demir and Süleyman Sadi Seferoğlu introduce readers to their efforts to design a Scratch-based coding achievement test for
students in higher education in Turkey. As Scratch is widely used across education levels, and has set the standard for block-based coding education, we anticipate this paper will draw attention from researchers and practitioners alike. This work builds on prior efforts to construct validated tests, but with the key distinction that this new assessment is not coding language independent. The authors argue that prior work to establish computational literacy tests that are language neutral tend to measure algorithmic and logical processes, rather than contextualized application skills. We need robust ways of measuring instructional effectiveness for computational literacy, both for informal and formal learning, and this research project is a step in that direction.

Despite the richness of approaches to the theme of “learning to code, coding to learn” presented here, we acknowledge that there are several areas that could see further research development, and we encourage and look forward to work that addresses these subtopics:

- **Adult and non-traditional learners**: The emphasis of many coding programs, such as Girls Who Code and Kids Code Jeunesse, are youth, and more specifically pre-teens, teens who are yet to find employment. Yet, coding is also an essential competency that many adults find important once they have left school or are looking to develop for personal advancement, for re-skilling and employment opportunity, or simply to be more effective in an increasingly high-tech workplace. The locus of this training may be online tutorials via YouTube or Lynda.com, evening classes at a community centre or public library, coding MOOCs via EdX or tinkering with like-minded makers at weekend meet-ups. My colleague Huang Hong has been exploring “live coding” communities where people code in real time, and there is a great deal to explore in terms of how these communities function as performance spaces, as well as personal and professional development (Haaranen, 2017). Chilana et al’s (2016) work on “conversational coding” is another interesting area of investigation. In this work, Chilana and colleagues unpack the motivations and strategies of people who want to learn coding to better engage with co-workers who do software development, even if they have no intention of compiling and running a program themselves in the future. Coding happens across the lifespan, and many demographic groups could benefit from computer science education at different levels of sophistication.

- **Teachers and librarians**: As CT is embedded in more curricula and supported through activities in diverse knowledge domains as language arts, science, and social studies, it will become essential to train more educators in the basics of coding. To deliver on the promise of CS4All, teachers need to be confident enough with technology to open pathways for learners, even if they are never expert coders themselves. This will require upgrading teacher education programs, as well as providing extensive professional development for in-service educators. The American Library Association’s “Ready to Code” (RtC) initiative (www.ala.org/tools/readytocode/home) is one such program that seeks to build capacity for in-service school and public librarians by seeding local initiatives in libraries, as well as increasing the use of computational concepts in post-secondary library training. Programs like RtC are nascent but already showing signs of progress; yet, there is still a great deal to do in this area.

- **Early childhood**: An explosion of coding-related toys and games have entered the marketplace in recent years, some of which have tactile or manipulative elements, that target early learners from ages of 3 to 8 years. These toys may be stand alone or work in conjunction with screens, such as the iPad, to encourage early computational play. The Hour of Code (https://hourofcode.com/) initiative has encouraged schools to adopt a number of free or low-cost coding apps (e.g.,
LightBot, Tynker, Kodable, and Scratch Jr.) that target pre-school and early elementary audiences. Despite the amazing diversity of entries we are witnessing, there has been little research that validates the utility or efficacy of these toys for coding education. Parents and teachers are challenged in deciding which of these, if any, are worth investing in, or building curriculum around, and serious reviews and comparisons are difficult to find. One means by which libraries can support these decisions is through media mentorship (ALSC, 2015) and the development of maker spaces and technology “petting zoos,” where you can try before you buy.

These three focus areas are certainly not the only areas where work in needed, but the editors of this special issue call these out for their tremendous opportunity for additional research and potential to affect new audiences for computational education and training.

**Conclusion**

Fishman et al. (2004) identified that innovations involving learning require capability, culture and policy to scale beyond initial efforts. We look forward to the future work of the authors in this issue as well as remind ourselves of the significant work that is yet to come. CT can be approached from a cognitive position as a type of computational problem-solving, with attendant skills, knowledge and dispositions that frame the work of coding. However, a socio-technical perspective acknowledges that computational skills and dispositions are deeply entwined with other concerns, including identity, social infrastructure, public policy, ethics, and human values. I argue that code is a political technology, one that we cannot and should not disentangle from the applications of computing, and the social contexts of use. If we seek to empower learners young and old through CT, procedural creativity, or critical computational literacies, our first step is to envision how these tools can contribute to human thriving. And if we find that our tools lead us away from a centring on human well-being, perhaps the greatest lesson is knowing when to seek a non-computational resolution, that is, knowing when not to code.

**Eric M. Meyers**

*School of Information, University of British Columbia, Vancouver, Canada*

**References**


**Further reading**


Generating gameworlds with computers: the case for procedural creativity

Matthew Jason Wells and Jason Boyd
Ryerson University, Toronto, Canada

Abstract

Purpose – Despite the popularity of the Computational Thinking (CT) paradigm and the call for broad social diffusion of CS fundamentals, the authors argue that the concept is inherently limited and limiting and does not sufficiently convey an understanding of how to enable people to create with computational technologies. The authors suggest an alternate paradigm, procedural creativity, that calls for the development of conceptual creative spaces governed by procedurally generative principles. The authors also call for game development to be the focus of procedural creativity pedagogy.

Design/methodology/approach – The authors first discuss the limitations of the CT paradigm, focusing, in particular, on the issue of abstraction and representation as opposed to execution and action. The authors then define procedural creativity in more detail. Following that, they discuss the use of game development as pedagogy, with a focus on Margaret Boden’s notion of conceptual creative spaces.

Findings – CT is limited because it focuses overly on solutions to computational “problems”, because it is tied too closely with economic concerns and because it focuses on abstraction at the cost of action. Procedural creativity, on the other hand, focuses on the individual’s capacity for personal expression with the computer and on the generative capacity of code in action. Game development is an ideal platform for procedural creativity because it emphasizes the development of creative domains and conceptual spaces.

Originality/value – This paper offers a challenge to the CT status quo and presents a novel way forward for understanding computation as a creative practice.

Keywords Education, Games, Creativity, Programming, Computation, Coding

Paper type Conceptual paper

Introduction

Computer science (CS) and the computer science education (CSE) movement, not surprisingly, have eagerly endorsed Jeannette Wing’s concept (first outlined in 2006) of Computational Thinking (CT) and her call for the broad social diffusion of CS fundamentals as an essential contributor to the social good[1]. This special issue is in part motivated by the idea that current educational initiatives and related research “offer support that the incorporation of computer science concepts in learning programs is an idea whose time has come” (Meyers and Huang). However that may be the case, in this paper, we temper that certainty by arguing that CT, as a paradigm for understanding human and social processes and for developing ways in which to enable people (especially youth in learning environments) to become empowered to think (and create) with computational technologies, comes with significant limitations. Accordingly, we suggest an alternate paradigm – procedural creativity – that moves beyond the limitations of CT and the CSE pedagogies and curricula it has informed. Although, like Meyers and Huang, we see the making of video games as a valuable mode in which to engage youth in effectively using computational technologies, we suggest that coding interfaces and games design curricula designed to teach youth CS in the guise of game making are limited by the paradigm of CT, and that
procedural creativity enables the consideration of other game creation interfaces and a richer application of games design approaches—interfaces and approaches that are not predetermined by and extend beyond CS and CT.

Our argument begins by outlining some of the presumptions and problems behind CT as a paradigm for human activity and endeavor, and particularly the shape it has taken in educational settings. It then discusses the concept of computation, and how it operates within a very delimited field of activity, a limitation which leads us to propose the concept of the procedural or procedurality, a more expansive concept than computation. Creativity is a common aspiration among proponents of computational thinking, and the concept of procedurality also enables an expansion of activity from thinking to creativity. Therefore, we propose the concept of “procedural creativity” as a more valuable paradigm in thinking how youth might use computing technologies in empowering ways. In this article we will expand on the definition of this term, and discuss the motivations and arguments for incorporating it into discourses on computational thinking. We will then work through examples of how computational creativity may be used as a tool for exploring facets of computation not addressed in computational thinking. Given the emphasis on games in computational thinking pedagogy, and given the high degree of procedurality inherent in game code, we will focus in particular here on games. The overall point, however, is that computation can not only solve problems, it can incite both creative action and reflection. As a concept, procedural creativity addresses all of these facets.

The limits of computational thinking
There are several aspects regarding the conceptualization of CT that require interrogation if it is to be promoted as key to learning environments for youth. The first is its initial development and deployment as a way of advocating for (or, more crassly, “selling” the idea of) the broadly social relevance of (and the public investment in) Computer Science Education (CSE), what Jeannette Wing, in her initial delineation of CT, called “a grand vision […] as we act to change society’s image of the field [of Computer Science]” (Wing, 2006, p. 35). Wing’s “call to arms” (Meyers and Huang) presumes that a broad diffusion (a social literacy) of Computer Science fundamentals is important to the well-being, the optimal functioning, of humanity and society. We suggest that, while CSE advocates may fervently subscribe to this claim, it is debatable whether Computer Science is the most empowering and enriching way in which youth might think with computers. In part, this is due to what we see as a key limitation of the conceptualization of CT, a limitation that is due in large part to the limitations of the concept of computation. One of its limitations is that it operates within a considerably circumscribed field: to compute is “to determine by arithmetical or mathematical reckoning” (OED Online, sense 1) (one might add here logical reckoning), and to determine something is “to put an end or limit to; to come to an end”; “to direct to some end or conclusion; to come to some conclusion” (OED Online, I and III). In these definitions we can see the highly specialized nature of computation, which relies on a scientific paradigm (mathematics) to furnish a definite end or result: thus Wing’s view that CT is directed toward solving problems—at coming to a definite result (a solution). CT conceives of human endeavor primarily taking the form of the (computational, logical) development of solutions to problems: “Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” (Wing, 2006, p. 33); “Computational thinking is a way humans solve problems” (Wing, 2006, p. 35). So, to take up the first aspect of CT mentioned above – what might be called the proselytizing zeal behind the original articulation of CT – with the tenet that CT is about solving problems, the result is an article of faith that believes that society
needs to mainstream CT in educational systems and settings (via CSE) and in society at large to effectively solve problems to benefit society. Unfortunately in real-world terms, despite Wing’s explicit claims that CT is not (or is not reducible to) computer programming (Wing, 2006, p. 35), the CSE agenda appears to have largely become “teach kids to code” to fulfill a neoliberal commandment that education should prepare youth to fulfill the current demands of the labor market: coding or programming is seen as a hard skill that (rightly or wrongly) will be in continual demand and will ensure an individual’s swift incorporation into the workforce.

But, we ask, what about problems that do not admit of logical/computational solutions, which are many and pressing? What about situations where there is not a problem at all? What about situations where the initial state is not a problem to be solved but instead a space of possibility/potentiality to be explored and interacted with? Our questions here are informed by a perspective on CT that is based on an intriguing and undeveloped reference in Wing’s original article: “Computers are dull and boring; humans are clever and imaginative. We humans make computers exciting” (Wing, 2006, p. 35). We consider it important to explore imaginative, creative, exploratory and experimental human-centered approaches to computers that are not driven by or limited to “problem-solving.” Even if a focus on a given problem is desirable, such approaches at least fit much more neatly with John Dewey’s theory of inquiry. As explained by Schön, “inquiry begins, Dewey believed, with an indeterminate (i.e. confusing, obscure, or conflictual) situation and goes on to make that situation determinate” (Schön, 1992, p. 122). This situation can only be made determinate, moreover, not through objective abstraction, but through subjective engagement with a possibility space, where “the inquirer does not stand outside the problematic situation like a spectator; he is in it and in transaction with it” (Schön, 1992, p. 122). We argue that CT must accommodate this more wide-ranging understanding of problem scenarios.

In a 2009 article on Scratch – MIT Media Lab’s well-known and highly-regarded visual programming platform – Mitchel Resnick and his colleagues at the Media Lab argued that with respect to computational thinking, programming “involves the creation of external representations of your problem-solving processes,” and thus it “provides you with opportunities to reflect on your own thinking, even to think about thinking itself” (Resnick et al., 2009, p. 62). Scratch will be discussed in more detail below, but for now, it is worth lingering on this particular statement, for it provides a useful entry point into a more general discussion on the limitations of computational thinking pedagogies. In this argument – and, indeed, in Scratch itself – the emphasis with respect to understanding programming is placed both on tangible representations and on abstract thought. As will be seen, these are the two cornerstones of most conceptions of computational thinking, and it is for this reason that the term is problematic, or at least restraining.

Both representation and thought are captured in the powerful term “abstraction”, which Wing has called “the essence of computational thinking” (Wing, 2006, p. 3717). Yet it is difficult to pin down a specific understanding of the term. Among the seven “big ideas” related to computation conceived by the College Board and National Science Foundation (NSF) in the USA, one focuses specifically on defining abstraction: “Abstraction reduces information and detail to focus on concepts relevant to understanding and solving problems” (Grover and Pea, 2013, p. 39). A separate group of NSF-funded researchers noted that abstraction is “commonly defined as the capturing of common characteristics or actions into one set that can be used to represent all other instances” (Lee et al., 2011, p. 33). Kramer offers two related definitions, one focused on removal – “the act or process of leaving out of consideration one or more properties of a complex object so as to attend to others” – the second focused on generality – “the process of formulating general concepts by abstracting
common properties of instances” (Kramer, 2007, p. 38). Curzon et al. add that abstraction “involves hiding detail – removing unnecessary complexity. The skill is in choosing the right detail to hide so that the problem becomes easier without losing anything that is important” (Curzon et al., 2014, p. 4).

For the most part, these definitions place an accent on various ways of thinking. There is a focus on the conceptual removal of detail from “information”, from “complexity”, and from “complex object[s].” There is also a focus on generality – on taking these mental conceptions and finding commonalities with other complex entities that can be similarly stripped of detail. This is all very cognitive, and makes computational abstraction sound like a largely mental process. However, consider the following statement on abstraction from Weintrop et al. (2016, p. 139):

[...]

creating an abstraction requires the ability to conceptualize and then represent an idea or a process in more general terms by foregrounding the important aspects of the idea while backgrounding less important features.

Here we see an emphasis on creation – by “creating” an abstraction, we both “conceptualize” and “represent” our ideas. The idea of concretizing an abstraction sounds almost paradoxical, but that is exactly what is promoted in most papers and articles on computational thinking. Kramer makes this link even more apparent later in his own work:

Abstraction skills are essential in the construction of appropriate models, designs, and implementations that are fit for the particular purpose at hand. Abstract thinking is essential for manipulating and reasoning about abstractions, be they formal models for analysis or programs in a programming language (Kramer, 2007, p. 40).

As an example, he cites Harry Beck’s 1931 London Underground map, which abstracted out most of the geographical specificity of the Underground station locations in favor of a schematic representation which is much easier to parse. The reification of the abstraction in this example – particularly with the invocation of such an iconic visual aid – makes representation appear to be the summit of computational thought. As Wing reiterates, “the nuts and bolts in computational thinking are defining abstractions, working with multiple layers of abstraction and understanding the relationships among the different layers” (Wing, 2006, p. 3718). Or, as Brennan and Resnick at MIT Media Lab put it, “abstracting and modularizing, which we characterize as building something large by putting together collections of smaller parts, is an important practice for all design and problem solving” (Brennan and Resnick, 2012, p. 9).

While abstraction is primary, the actual execution of these abstractions is something of an afterthought. As Wing notes, after emphasizing the importance of abstraction in the above citation:

[...] the power of our ‘mental’ tools is amplified by the power of our ‘metal’ tools. Computing is the automation of our abstractions. We operate by mechanizing our abstractions, abstraction layers and their relationships (Wing, 2006, p. 3718).

This two-step computational dance – first abstraction, then automation – is repeated throughout the literature on computational thinking. There is, however, an important problem with this situation: this particular paradigm has been commented on, and heavily criticized, both without and within computer science, for several decades. It is what Terry Winograd and Fernando Flores termed the “rationalistic orientation” to computing. Winograd and Flores were the first computer scientists to provide a comprehensive philosophical challenge to what they termed the “rationalistic” approach to solving problems via computation. They summarized the rationalistic process in the following steps:
Characterize the situation in terms of identifiable objects with well-defined properties.

Find general rules that apply to situations in terms of those objects and properties.

Apply the rules logically to the situation of concern, drawing conclusions about what should be done (Winograd and Flores, 1986, p. 15).

The links with the definitions of computational thinking concepts discussed above should be clear. The first two steps fit with virtually every conception of abstraction that was considered earlier, and the third step fits with Wing’s notion of “mechanizing” our abstractions. This can be thought of as a dichotomy between “thinking” – developing concepts and logical structures that represent some aspect of the world – and “doing” – initiating actions in the physical world related to these concepts and structures.

This is called the Cartesian perspective, after René Descartes, by Ehn, who notes that “the Cartesian philosophy is an epistemology and ontology of an inner world of experiences (mind) and an outer world of objects” (Ehn, 1988, p. 52). It is also a separation of language and action – as Winograd argues, we must think of “linguistic communication as the basis for understanding what occurs in information systems” (Winograd, 2006, p. 72). This could mean literal programming languages, of the sort described above or other languages we use to design and develop programs, like modeling languages (including both text and graphics.) But it could also refer to mental languages, such as mental models, defined by Jonassen and Henning as “[mental] representations of objects or events in systems and the structural relationships between those objects and events” (Jonassen and Henning, 1999, p. 37). From this perspective, we are readily able to associate computational thinking methodologies with language use, as abstractions are represented both by mental models and by programming languages.

The problem with the rationalistic approach, then, is that it does not match up with current ideas about how language is used. A key impetus to the shift that has taken place is the rise of constructivist ontological principles in a variety of fields, including computer science. Traditionally, as computer scientist Christiane Floyd has noted, software engineers were encouraged to think in terms of “correctly” modeling whatever real-world systems they were meant to digitally manipulate – that is, they were to assume that the “entities and actions” of the “real world” are “supposed to be ‘given,’” and that “the software developer’s task is to analyze, to abstract and to elaborate a correct model that can be manipulated by the computer,” with a focus on “matching the real world in the model with the greatest possible care” (Floyd, 1991, p. 16). Yet from a constructivist perspective, this process is envisioned much differently:

This picture changes drastically when we acknowledge our active role in bringing about what we hold for real, which is the key to constructivist thinking. The emphasis now is on the observer constituting the way he or she sees reality and inventing a suitable description. Thus, the software developer is portrayed as making choices in an open situation, where there is more than one possibility (Floyd, 1991, p. 16).

Ehn echoes this perspective when he states that “in using language we do not describe the world, we create it. Just as we do with other artifacts. The carpenters’ world is continuously reproduced in ready-to-hand use of language, and eventually recreated in situations of breakdown (Ehn, 1988, p. 69). This philosophy is now known as the Language-Action Perspective (LAP), a term first popularized by Winograd and Flores. There is a level of freedom in LAP that is not afforded to programmers in the traditional paradigm. Scholars such as Wing posit that there is a world “out there” that is waiting to be discovered and captured computationally, which can and must be done to fashion new discoveries and innovations. The constructivist stance is that computation is literally and inherently a creative process, and that code shapes a new reality.
As a point of emphasis, Floyd also notes that “the execution of programs [...] may be characterized as constructed reality” (Floyd, 1991, p. 16). This emphasis on running code is a fundamental quality of the language-action perspective. According to Winograd, the “second principle” of LAP “is that language is action,” which he explains as follows: “Through their linguistic acts people effect change in the world. In imposing a language-action framework on information technology, we emphasize the action dimension over the more traditional dimension of information content” (Winograd, 2006, p. 72). Note the importance of this latter statement as it stands in contrast with common conceptions of computational thinking, where the emphasis is placed on representation, not action. This is a point that Schoop also raises:

The conventional perspective on information systems stresses the contents of messages rather than the way they are exchanged [...] the focus is on the form and structure of messages. In contrast, the Language-Action Perspective emphasizes what people do while communicating, how language is used to create a common reality for all communication partners, and how their activities are coordinated through language (Schoop, 2001, p. 3).

There is also an important activist element to LAP; as Weingrad notes, “the fundamental assumption of LAP is that language is not only used for exchanging information about the world [...] but also for changing the (social) world” (Weigand, 2006, p. 46). Such thinking also echoes Dewey, who believed that “inquiry [...] is transactional, open-ended, and inherently social” (Schön, 1992, p. 122). This does not mean political action, necessarily; the key point is that the LAP approach assumes that code is an active agent in the world, shaping it rather than merely representing it.

While we do not associate our work directly with LAP research, we feel that there is much to gain from adopting this constructivist, action and language-oriented approach to understanding computation and computer programming. Computer scientist Seymour Papert advocated for a constructivist approach to programming pedagogies for decades, reflected most famously in his work *Mindstorms: children, computers, and powerful ideas* (Papert, 1980). Describing a theoretical computer-based virtual physics laboratory, Papert (1980, p. 126) noted that:

[... ] learners in a physics microworld are able to invent their own personal sets of assumptions about the microworld and its laws and are able to make them come true. They can shape the reality in which they will work for the day; they can modify it and build alternatives.

Papert was inspired in his ideas by the work of psychologist Jean Piaget, who focused much of his work on childhood education and intellectual development. According to Ackermann, “Piaget and Papert are both constructivists in that they view children as the builders of their own cognitive tools, as well as of their external realities,” while both believing that “knowledge is not merely a commodity to be transmitted, encoded, retained, and re-applied, but a personal experience to be constructed” (Ackermann, 2001, p. 7). Like the other scholars discussed above, Papert places the emphasis on computer users shaping reality, not merely reflecting it. It is in this “reality constructing” spirit that we invoke the term *procedural creativity* as a watchword to designate such approaches as described above, and their importance when teaching new learners about computing practices. How such teaching might happen will be explained in the next several sections.

**Procedural creativity defined**

The core of what we propose as a new approach to computer-based pedagogy is the concept that we have termed “procedural creativity”. Unlike computation, a procedure is a
“particular course or mode of action [...] a process, a proceeding” (OED Online, sense 2a): in other words, to focus on procedurality is to focus on the activity or doing and not on the result. Procedural creativity predates computers. One instance of this is OULIPO (Ouvroir de littérature potentielle) (“Workshop of potential literature”), founded in 1960 by a poet, Raymond Queneau, and an engineer, François Le Lionnais. OULIPO’s focus is on the development of constraints, essentially the proposal of formal rules and procedures that might lead to new and perhaps unexpected forms of literature. OULIPO’s focus, then, is not on the creation of literary works, but on algorithms or programs—processes—by which literary work might emerge, often through combinatorial procedures, such as Queneau’s *Cent mille milliards de poèmes* (1961), which consists of ten sonnets in which every sonnet’s lines can be combined with lines from other sonnets, which results in $10^{14}$ (100,000,000,000,000) distinct sonnets. A computer-based example is the work of Nick Montfort, who devises minimalist programs that can generate potentially interesting literary outputs. As with OULIPO, is it the creativity that goes into the process—the program—that is of primary interest. A useful example is Montfort’s *ppg256* series, a series of poetry generators written in Perl limited to 256 characters (https://nickm.com/poems/ppg256.html). Works like Montfort’s raise new areas of inquiry that call for new theoretical paradigms, as William Winder notes in his article on “Robot Poetics”: “Generated text has a robotic author, itself created by a human programmer. There is a poetics of creating that author (here creativity lies in the writing of instructions); a poetics of generating a specific text (how those instructions play out to make a given text); and a poetics of reading generated literature (how the reader will read a text knowing, or not, that it is automatically generated).” Developing these poetics would provide a deeper understanding of the dimensions of procedural creativity.

We define procedural creativity as the creation of possibility spaces—or, as will be explained in more detail below, what Boden terms *conceptual spaces*—governed by procedural generative principles. To put it less technically, we would say that procedural creativity is about making a functioning/dynamic world with the computer. A world is more than a single program, or a single solution. It is a collection of resources that may be leveraged to build imaginative, responsive environments. It is an opening of possibilities, not a closing. The term also expresses the fact that computers could and should be a vehicle for creative endeavor, whatever that might entail, and not just computation for instrumentalist ends. In other words, *pace* Wing, using a computer does not necessarily have to require thinking like a computer scientist. Moreover, thinking like a computer scientist is not necessarily the best way to use a computer. That is not to say that using a computer should not involve programming—programming, in fact, can be a valuable way to engage in procedural creativity. But this does not mean that learning programming for creative purposes must be in the service of understanding programming’s supposed reflection of computational forms and structures.

Procedural creativity does not preclude computational thinking, nor is it the antithesis of it. The relationship between the two is dependent upon the application of each. In the case studies to be discussed later in this article, traditional computational thinking seemingly takes on something of a supportive or even submerged role. Programming is an optional practice in each, but one that very much enriches the overall experience. However, as procedural creativity almost inevitably involves some form of coding (however abstract), it could be argued that from a conceptual perspective, it is all-encompassing. Regardless, pedagogical tools in the computational thinking space, such as Scratch, definitely have a role to play in teaching procedural creativity, but they would need to be expanded upon to completely fulfill the mandate. This topic will be revisited in the conclusion, but one final
point to make here is that a major difference between the two concepts is procedural creativity’s focus on action, and on running code, rather than studying code as a form of static representation. Creativity emerges as code is executed, or as executed code is applied elsewhere – this is an area where games serve as an ideal platform for computational creativity, as will be explained in more detail below.

Procedural creativity is about communication between the developer and the user, the developer and the computer, and even the developer and themselves. It is about playing with ideas, and shaping new realities through those ideas. When a user creates a program, they do not simply recreate some aspect of the “real” world – they create a new world unto itself, with specific rules and resources, some realistic and some definitely fantastical. Programs are jumping-off points for new programs, and for new ideas based on those programs. Programs are ways to communicate ideas, and are also ideas themselves. They invite and accommodate reconfiguration and expansion. They represent, but more importantly, they generate; they act, and initiate subsequent action.

A key modification in our approach, of course, is the substitution of the term “computational” with “procedural”. This is more than just a semantic shift – it represents a paradigmatic move from a focus on the work that a computer is doing to the work that the user does when using a computer. The term “computational” refers directly to the processes of calculation. Computation is about finding an answer. The very term “computer” was first used to refer to those individuals that manually solved algorithmic problems for the Ballistics Research Laboratory of the USA Army over the course of the Second World War. These problems calculated the ranges of various pieces of artillery in various conditions, filling out “firing tables” for use in battle (Polachek, 1997; and Grier, 2005). Computation was about following instructions and producing solutions, with human agency at a minimum. The term “procedural”, however, connotes process. Processes are generally constructed by human agents, and can be executed either by humans or machines. By referencing procedure instead of computation, then, we are reintroducing the human element, and then by pairing the term with “creativity” we are pointing to outcomes that are far more open-ended than simple solutions to basic computational problems.

To expand on these initial arguments, it is first necessary to understand the importance of game design, both as a vehicle for procedural creativity and as a pedagogical tool for learning associated creative practices. This will lead in to further discussion of Boden’s notion of conceptual spaces, and the links between game design and conceptual space will be delineated. Two specific examples of game development tools will then be explored.

**Game design as pedagogy**

Pairing a teaching philosophy with an engaging pedagogy can be difficult. The LAP approach, for example, has been used primarily in business software to improve workflow and communication (Schoop, 2001). While important, such applications are unlikely to draw learners, particularly young learners, to computing and coding. We instead propose that digital gaming, and in particular game design, is an ideal vehicle for teaching and understanding procedural creativity. In Yamin Kafai’s important and influential *Minds in Play*, she notes that “games, more than any other media, have brought technology into children’s homes, and children have received them enthusiastically,” while arguing that “I see programming games as a medium for children’s personal and creative expression” (Kafai, 1995, pp. 13-17). More than that, however – while acknowledging that Kafai does not speak for all children – games operate at a level of procedurality that other media types rarely, if ever, reach. Games are inherently transactional, and game code must continuously respond to user input, which itself changes based on visual and auditory feedback. Games
involve the user in a procedural loop, a blend of user action and programmatic reaction that is established and configured in the design process. Games are thus more than just vehicles for vaguely-understood notions of meaningful personal expression. By bringing in new theory, particularly the work of Margaret Boden, the direct benefits of game development as a creative process, and the importance of creativity as a fundamental computational process, can be tangibly expressed.

Boden’s research into the links between creativity and computation provide an ideal platform for understanding the true importance of games as computational artifacts. The key to Boden’s thinking about creativity is her notion of the conceptual space. Conceptual spaces are constrained sites of creative activity[2]. Boden has described them as “structured styles of thought,” and “particular set[s] of generative principles” (Boden, 2004, p. 4; Boden, 1995). Wiggins, operationalizing Boden’s definition somewhat, argues that “the conceptual space is a set of artefacts (in Boden’s terms, concepts) which are in some quasi-syntactic sense deemed to be acceptable as examples of whatever is being created” (Wiggins, 2006, p. 212). Pike, taking a more culturally-focused perspective, noting that:

[…] the composer who generates a musical motif, or the author who generates a constellation of words, does so with reference to a wider set of social conventions, values and beliefs that determine the meaning and significance of what is created (Pike, 2002, p. 87).

Boden presents a list of examples to clarify the concept:

This may be a theory of chemical molecules, a style of painting or music, or a particular national cuisine. The space is defined (and constrained) by a set of generative rules. Usually, these rules are largely, or even wholly, implicit. Every structure produced by following them will fit the style concerned, just as any word string generated by English syntax will be a grammatically acceptable English sentence (Boden, 2009, p. 25).

Conceptual spaces, then, are sites of constrained generation. That is, they contain a core set of rules, resources, and conditions that can be combined to create larger artifacts. Cognitively, a conceptual space may be difficult to pin down exactly, as it is difficult to divine the “rules” used to create, for example, a particular genre of music. When modeled procedurally, however, conceptual spaces may be specified much more precisely. The relationships, and tensions, between cognitive and procedural spaces are an important element in procedural creativity.

Conceptual spaces are meant to be explored, generally through the generation of specific examples of creative works that are generated out of the rules and constraints of the space. This is what Boden terms exploratory creativity, which she describes as follows:

In exploratory creativity, the person moves through the space, exploring it to find out what’s there (including previously unvisited locations) – and, in the most interesting cases, to discover both the potential and the limits of the space in question (Boden, 2009, p. 25).

How this exploration unfolds depends on the skill of the individual or entity doing the exploring. As Boden (1995) notes:

The ‘mapping’ of a conceptual space involves the representation, whether at conscious or unconscious levels, of its structural features. The more such features are represented in the mind of the person concerned, the more power (or freedom) they have to navigate and negotiate these spaces.

Conceptual spaces, however, can also be changed and created, and this is what Boden calls transformational creativity. As she explains it:
[... ] in transformational creativity, the space or style itself is transformed by altering (or dropping) one or more of its defining dimensions. As a result, ideas can now be generated that simply could not have been generated before the change (Boden, 2009, p. 25).

Wiggins puts it more succinctly: “transformational creativity is the process of changing the rules which delimit the conceptual space” (Wiggins, 2006, p. 212). Transformational creativity is about creating new generative principles, upon which exploratory creativity will find new configurations.

Boden grounds much of her work in AI research, so her perspective on creativity is at once computational, and capable of being represented computationally, at least in part. As she notes, “computational concepts help us to specify generative principles clearly [...] And computer modelling helps us to see what a set of generative principles can and cannot do” (Boden, 2004, p. 52). With respect to AI theory – which, as we noted, we can understand also includes game modeling – she states that “AI can provide dynamic processes as well as abstract descriptions” (Boden, 2004, p. 89). Note the parallels here with the discussion earlier on representation versus action. Here, Boden seems to imply that the two are on equal footing. When discussing computational representations of exploratory creative spaces, she further notes that “they concern the set of structures [...] described and/or produced by one and the same set of generative rules,” and then makes the following observation on the use of the terms “described” and “produced”: “one focuses on the structural possibilities defined by “generative rules” considered as abstract descriptions. The other focuses on the possibilities inherent in “generative rules” considered as computational processes” (Boden, 2004, pp. 49-50). In other instances, she appears to prioritize process and action, such as when she notes that “the generative potential of a computer program is often unpredictable and may even be indefinitely variable” (Boden, 2009, p. 31). It is this focus on action that makes games once again an ideal candidate for study from a creativity perspective.

**Games as conceptual spaces**

It could be argued that, from a play perspective, games are sites of exploratory creativity. Game design, however, is a site for transformational creativity. There are fundamental reasons why this is the case, related to how games are designed and organized from a procedural perspective, and it is for these reasons that we argue that game design is an ideal site for procedurally creative practices. To understand this, it is necessary to go into the history of the concept of the game engine.

In digital gaming’s earliest years, games were built directly in machine and assembly code, the languages directly processed by computer CPUs[3]. The Atari VCS home console system in particular is well-known for its entire lack of anything resembling an operating system, meaning that game programmers had to essentially tell the internal CPU how to operate along with building a game on top of that basic functionality (Montfort and Bogost, 2009). The Nintendo Entertainment System (NES) was even more complex and similarly lacked an OS, though its developers fashioned some generic processes to speed up coding (Altice, 2015). Computer game coders generally had operating systems to work with, but assembly language was still a popular language(s) of choice to optimize performance. Still, alternatives emerged. In the early 1980s, pioneering interactive fiction company Infocom, creators of the Zork adventure game series, developed the Zork Implementation Language (ZIL), a custom-made programming language that they used to create their vast array of text-based games (Montfort, 2003). ZIL allowed Infocom’s developers to create new games using what in many ways resembled a high-level programming language like BASIC[4], with the ZIL platform itself handling the translation of the language into the necessary assembly language[5].
Yet, ZIL was not a generic programming language, nor was it a generic interactive fiction programming language. It was, in fact, a programming language designed to generate more games like Zork. That is, in terms of fundamental structure and mechanics, every ZIL game was exactly like Zork. Where it allowed for flexibility is in how these fundamental structures were arranged, and, crucially, in the prose text that was displayed to the player. Games with very similar underlying structures could seem completely different if the text displayed to the user presented, for example, a science-fiction setting versus a traditional detective story. What ZIL did, then, was define a conceptual space within which developers could work to fashion new games. It contained both constraints and generative rules and affordances, the building blocks for a very specific type of game. Using this space to generate new games was thus a form of exploratory creativity. As Aycock notes, once a language like ZIL is implemented, the question becomes, “How would game programmers express themselves in the interpreted game language?” (Aycock, 2016, p. 52). ZIL games were the end results of programmers using the language as a form of personal and professional expression.

Through the 1980s and early 1990s, in-house tools such as ZIL became increasingly common in the computer game industry. Graphic adventure game company Sierra On-Line (later Sierra Entertainment), for example, developed the Adventure Game Interpreter (AGI) system for its landmark game King’s Quest, released in 1983 (Collins, 2008). As with ZIL, AGI was designed around a single game, but was then used by Sierra’s developers to create a variety of other graphical adventures with similar but reconfigured resources. Unlike ZIL, however, AGI also had to accommodate graphics and sound assets, making for a more complicated development platform. Sierra would go on to develop the more sophisticated Sierra Creative Interpreter (SCI) platform in 1988 for King’s Quest IV: The Perils of Rosella, serving similarly as a generative conceptual space for graphical adventure games (Loguidice and Barton, 2009). Interestingly, and importantly, both systems have been recreated by hobbyists in recent years, who have user-friendly, “studio”-style development environments for the crafting of new games and other experimental programs. This capacity to transform development platforms into new environments is reflective of the inherent generative quality of digital games, in that they motivate and accommodate continual reimaginings by their players and fans. This is a point that we will return to shortly.

By the early 1990s, development tools such as SCI were becoming much more common, but it was not until the 1993 release of the foundational first-person shooter Doom that the game engine, and the term “game engine”, fully emerged. id Software, the creators of Doom, were motivated in part to develop the engine system by the modding efforts of players and fans of their previous game, Wolfenstein 3D. As Lowood notes, when the game released, “dedicated players’ efforts to hack the game and insert characters such as Barney the Dinosaur and Beavis and Butthead made an impression on [id founders John] Carmack and [John] Romero” (Lowood, 2016, p. 207). As such, they opened up Doom so that “players could now alter assets such as maps, textures, and demo movies” (Lowood, 2016, p. 207). To be able to do this successfully, such tools required a stable software interface to plug into, e.g. to allow users to create maps, the map resource in the game code has to be separated out from the other game components, as well as from the hardware drivers which rendered the map and other game elements on-screen. As users were able to modify and create a multitude of different game resources, each had to be conceptually isolated as such within the code. This collection of components became known as the “game engine”, defined by Lowood (2016, p. 203) as follows:
As the term has developed since the early 1990s, game engine encompasses the fundamental software components of a computer game. These components typically include program code that defines a game’s essential “core” functions, such as graphics rendering, audio, physics, and artificial intelligence, although the components vary considerably from one game to another.

Lowood’s last point is worth emphasizing. Much of the academic literature on game engines focuses on the player side— that is, the creative potential inherent in tools such as level and asset editors. But it is also worth noting, as Lowood implies, that the creation of a game engine is a creative act in and of itself. The fact that their internal components “vary considerably” suggest design choices that must be made over the course of an engine’s development. And, as with ZIL and the other engines discussed above, these choices typically reflect the qualities of the game that the engine was first designed for. In the case of Doom, the engine was conducive to creating three-dimensional, action-oriented games with an emphasis on shooting as the mode of interaction. Other games that licensed and incorporated Doom’s engine, such as the 1994 game Hexen, used different aesthetic themes, but played very similarly to the original game. Doom led to the rise of the first-person shooter as a dominant game genre in large part because its engine was purpose-built for Doom-like first-person shooters, as were the engines created by its imitators and competitors.

Game design, then, is more than just a singular creative act. It is the creation of procedures that enable creativity. Often these procedures are tied together in one specific way, the end result being a unitary game release. In digital gaming’s earliest days, this is how things were done. But the emergence of development platforms, and then of full-fledged game engines, changed this calculus. It became possible to take the elements crafted to make one game, and repurpose them to create another. It became possible to release tools that plug into these engines to the players themselves, enabling them to experiment with alternate configurations of game resources. Such tools work because game design, as conceptual spaces, divide into discrete, generative elements. A single tool, such as a level editor, tends to focus on one element in particular, with links created to other types of elements. In addition to tools are scripting and other high-level languages, which are generally more powerful in that they allow more direct access to a given game’s underlying code. Such languages turn game infrastructure and mechanics into text, allowing for the shaping of both game content and form using the same representational forms. Language also allows for common routines and procedures to develop, giving developers entire libraries of new tools with which to make their games. It is this simultaneous shaping of generative resources and content, of game worlds and conceptual spaces, that is characteristic of game development, and the core value of procedural creativity.

Games and procedural creativity

Given our conception of procedural creativity, and given the discussion of game development that preceded it, the connections between the two should be clear. But a brief discussion, to be followed by examples, will be given here. Game development is an ideal platform upon which to build pedagogies for procedural creativity. As noted, games are essentially conceptual spaces, and the process of game development is the creation of a conceptual space. There is more to the story, however. Given the importance of procedural creativity, as outlined in the previous section, we argue that the goal of any programming pedagogy should be the creation of conceptual spaces. In other words, learning to program is not an end goal in and of itself, nor is learning to program in a traditional programming language necessarily an ideal approach to learning (though such languages can be powerful tools toward procedural creativity, as will be discussed below.) Rather, the goal should be to
promote creativity through the procedural development of conceptual spaces. And no genre of software is more conducive to such efforts than games.

Case study: Twine
Twine (twinery.org) is fundamentally a hypertext/hypermedia authoring platform that can incorporate programming elements, which enables it to be used as an interactive fiction and game creation platform. Twine (version 1) (in the form of a downloadable software) was created by Chris Klimas in 2009 using Python and JavaScript; a second version (Twine 2), written in HTML5 and JavaScript, exists in both a browser-based and downloadable versions. In large part due to its accessibility and flexibility (and because it builds into a HTML file that can be easily published on the web), Twine was eagerly taken up by a range of independent game creators and hobbyists, including Anna Anthropy (author of *Rise of the Videogame Zinesters*), Porpentine and others (a collection edited by Merrit Kopas, *Videogames for Humans: Twine Authors in Conversation*, provides an informative snapshot), as well as educators at all levels.

One of the strengths of Twine is its flexibility, or perhaps more precisely, its avoidance of presuming what kind of creative work its users might want to make. Unlike “teach programming through game making” platforms, Twine does not require the use of programming: one can create a hypertext consisting solely of interlinked passages of text (although anyone familiar with classic hypertext literature such as Michael Joyce’s *afternoon* knows that hypertext can be fiendishly complex in construction and navigation). But Twine allows for more than the linking of passages of text. Besides being able to incorporate a range of other media, it allows for the incorporation of code in the form of macros and functions, enabling the creation of game scenarios through the assigning, tracking and modifying of variables and the using of (for example) conditional loops. (Twine 2 is not yet at Twine 1’s stage of development in this regard, although the *Twine Cookbook* is rapidly closing that gap).

An instructive approach to using Twine in a procedurally creative way is what might be called a “backward” approach to authorship: instead of starting with an idea or plotline for a game or interactive narrative and looking for the means to realize that idea, one starts with the constraint of learning how Twine’s macros and functions work, and then ideates how they might be used to good effect in a story or game experience. This is an Oulipian approach, an example of procedural creativity in that one is thinking how processes represented by Twine’s macros and functions can be used for creative ends—it considers less their instantiation as a programming concept and more their ability to create an aesthetic effect in service of producing a “potential literature.” Twine’s flexibility includes the ability to (with knowledge of the Twine engine and JavaScript) create custom macros and add extensions—but again, the motivation here is creative rather than focused on learning or representing CS concepts.

Computer Science professor Chris Martens, in a three-part blog series, “Using Twine for Games Research,” explores the idea of using Twine as a prototyping tool for game design, examining how Twine could be used to model the game world or system of a hack-n-slash adventure. At the conclusion of the third post, she observes that the features of Twine 2 she has explored encompass “most of the basic ideas behind imperative programming”:

To those who say, “I’m not a programmer, I just use Twine,” or worse, “I am a programmer, so I’m above using something like Twine,” I hope this post encourages you to reconsider your perspective on what is and isn’t programming. Twine 2 is a (relatively minimal) *programming language* for prototyping games, and viewing it that way can give us a clearer sense for the relationship between programming affordances and design constraints.
What is interesting to note here is that one can say “I am a Twine author” and at the same time be a programmer without being aware of or indeed caring about that role. As Martens notes, using some of the built-in programming features of Twine 2 means in essence one is learning fundamental concepts related to a major programming paradigm (imperative programming), yet this not something a Twine user needs to realize to effectively engage in procedural creativity.

In essence what Twine represents is a flexible platform for procedural creativity that makes use of computer science concepts and their instantiation in features like macros and functions with the aim of making these features available for creative exploration and world building rather than as an instrument for CSE in the form of game creation tool. Twine approaches CS and programming as a resource to be used selectively to expand the creative affordances of the computer rather than seeing its purpose as a learning platform for CS and programming.

Case study: Inform 7

In the early 1980s, one of the most popular forms of computer game was the “text adventure”, a genre in which players interacted with text-only game worlds via command-line parsers. Such parsers enabled players to move about game worlds, interact with people and objects, and perform other specialized actions, all by typing in noun and verb-heavy phrases (Montfort, 2003). The Zork series, discussed above, were text adventure games, as were all the games made with the ZIL language. Recently, the genre, now rebranded as “interactive fiction”, has enjoyed renewed interest; the rise of similarly-structured hypertext games, discussed above, has likely amplified their reach. In the early 1990s, Oxford mathematician Graham Nelson developed the Inform system, which consisted of both a programming language and compiler, to generate new works of interactive fiction (Reed, 2011). While the language did not receive much attention at first, its popularity has steadily increased.

Since that initial release, the Inform programming language has taken on a number of incarnations. Inform 6, developed by Nelson in 1996, was the standard version for several years. Inform 6 was very much like a traditional programming language, or, more accurately, a traditional domain-specific language (DSL) – that is, a programming language designed for a specific task or tasks, i.e. a domain (Fowler, 2010). The following code snippet is taken from Nelson’s The Inform Designer’s Manual:

```inform
after [;
    Go: if (noun == w_obj)
        print "You feel an overwhelming sense of relief."
    ];
```


The semantics of these statements are not important here, but note how the syntax and overall structure are very much rooted in programming languages such as C and Java. The various brackets and parentheses, the mathematical look and feel, and the terse verbiage are all hallmarks of a very familiar form of high-level programming. Inform 6 is also an imperative language, meaning that it tells the computer directly what to do when it is executed[6].

In 2006, 10 years after the release of Inform 6, Nelson released Inform 7, a complete rework of the language. While the previous version of the language was imperative, as already noted, Inform 7 is a fully declarative language, meaning that it describes the logic and structure of a system, but does not define specifically how the system will be operated.
Language is important; as Boden notes, “some of the most important human creations have been new representational systems” (Boden, 2004, p. 107). In the case of Inform 7, this new notational system has profound implications for how the language serves as a vehicle for procedural creativity. First, it can be expressed in a much more readable syntax. Nelson made a point of making Inform 7 sound as much like natural English as possible. Compare the following code snippet to the one from Inform 6 shown above:

After switching off the emergency lights:
now every room regionally in Tower Vicinity is dark;
if headlights are switched on:
now every within range of headlights room is lighted;
say “The lights flicker and die out.”
(Reed, 2011, p. 174).

This is barely recognizable as programming code, given the almost total lack of specialized, mathematical syntax. In actual fact, there is still structure to this code, and the programmer is still making logical statements. However, all of this is embedded in a grammar that is built around short, declarative English-language sentences. The key takeaway here is that Nelson has deprioritized the importance of making Inform code look like a representation of how a computer “thinks” in favor of making something more recognizable to human readers.

Beyond readability, what makes Inform 7’s declarative style so effective in the way in which it allows for the easy construction of entire worlds – with their own rules, laws, objects and places – within which to fit specific games. An Inform 7 program is sort of like a rulebook, or a book of laws. It “declares” the existence of specific resources, the characteristics of these resources, and the ways in which resources may interact with one another. This paradigm matches up well with how stories are generally conceived of and told. As Reed (2011, p. 41) notes:

[...] good storytellers hold universes in their heads [...] To teach a computer with no imagination how to tell a story, we must first explain to it our story world - the places, things, people, and plans that make up a narrative.

Resources can range from in-game objects and places, to the actions that a player may take, or that resources may take with one another, to the very rules that define how the overall game is structured and played. In Inform 7, the game, the game world, and the basic rules that operate the game world are all fully modifiable in the same language. There are multiple ways to express similar aspects of game spaces, and there are innumerable ways to use the language to structure game worlds in specific ways.

Conclusion: next steps
The key points of procedural creativity are summarized in Procedural creativity, summary.

Procedural creativity: key points
- It focuses on contexts in which there is no logical/computational solution, or where there is no problem to be solved;
- It is opposed to rationalistic perspective on computation which emphasizes computational abstraction;
- It aligns with language-action perspective understanding of running code as communication and reality construction;
- It views computational activities such as programming as creative acts;
- It recognizes that programs can perform, generate, communicate and initiate further action; and
- It focuses not on the work that can be performed by a computer, but on what the user can create with a computer.

When developing a new concept such as procedural creativity, particularly as it stands in (partial) opposition to an established term such as computational thinking, it is important to push the case on a number of fronts. As such, we propose that research should continue in several related areas. First, it is worth exploring in more detail the meanings of terms such as procedural and computational. These terms have both technical and historical baggage that must be thoroughly investigated to develop meaningful and influential definitions for the concepts we will use to develop programming pedagogies in the future. One major point of inquiry would be whether or not procedure implies determinism (random factors notwithstanding). Is a procedure something that must always be followed exactly, or is there room for agency within certain types of procedures? The answer (or lack thereof) has implications with respect to how and when we can rely on computers to perform procedural work for us. Second, it would be worthwhile to catalogue and categorize more formally the ways in which game development platforms enable procedural creativity. An important aspect of this work would be determining the validity and effectiveness of game modding tools (discussed above) as resources for procedural creativity. It would also be useful to examine more closely supposedly “generic” game development systems such as GameMaker Studio and Unity to draw out the ways in which they are biased (or not) toward certain types of games, and the historical motivations behind such biases. The very discourse of genericity with respect to gaming tools is also worth interrogation. Third, it would be useful to further investigate the computer science concepts discussed above, such as LAP and AI-based creativity. We only touched on the core concepts in both areas, but there is a wealth of supplementary research that could provide further insights into how procedural creativity might operate. The use of computational and programming languages, as well as models, seems to be a particularly intriguing area of study, both in terms of language as a form of expression and as a means to generate and mediate other forms of language, including code. Finally, in keeping with current trends toward design research, it would be extremely useful to develop our own digital tools and applications to further explore the possibilities and limitations of computational creativity. Such tools would serve as vehicles for generative creativity in and of themselves, and could and should allow for the exploration of other creative domains. Game development utilities, for the reasons explained above, would be particularly useful, though other forms of creative, generative software would also be worth exploring. Procedural creativity is about opening up possible new vistas, not closing them, so ultimately any worthwhile research path should be followed.

Notes

1. This paper is the outcome of an extended and ongoing discussion between the two authors on the topics covered. Matthew Jason Wells is the primary writer of this paper, with Jason Boyd contributing the introduction and case study on Twine, as well as some other sections in “Procedural Creativity Defined.” Each author, of course, made additions and modifications to the other’s contributions in the course of writing.

2. Boden also discusses a form of creativity, combinatorial creativity, that relies less on her concept of conceptual space (see Boden, 2009). While combinatorial creativity is an important concept, even in the present context, due to lack of space it will remain outside the scope of this article.
3. Note that the earliest arcade games did in fact not use CPUs, but by the mid- to late 1970s they were starting to become standard. Home game consoles began to use CPUs at roughly the same time.

4. Though ZIL was actually based largely off of the more obscure high-level language Lisp.

5. Infocom also developed a program called the “Z-machine” which ran ZIL programs on multiple computer systems, acting as a “virtual machine” in the same way as the Java Virtual Machine (JVM).

6. Inform 6 is an object-oriented language, like C++ and Java, which means that it has more flexibility in terms of code execution than an older, strictly procedural language like C. But it is still technically an imperative language.

References


Reed, A. (2011), Creating Interactive Fiction with Inform 7, Course Technology, Boston, MA.


Further reading


**Corresponding author**

Matthew Jason Wells can be contacted at: matthew.wells@ryerson.ca
Unfold studio: supporting critical literacies of text and code

Chris Proctor and Paulo Blikstein
Graduate School of Education, Stanford University, Palo Alto, California, USA

Abstract

Purpose – This research aims to explore how textual literacy and computational literacy can support each other and combine to create literacies with new critical possibilities. It describes the development of a Web application for interactive storytelling and analyzes how its use in a high-school classroom supported new rhetorical techniques and critical analysis of gender and race.

Design/methodology/approach – Three iterations of design-based research were used to develop a Web application for interactive storytelling, which combines writing with programming. A two-week study in a high-school sociology class was conducted to analyze how the Web application's textual and computational affordances support rhetorical strategies, which in turn support identity authorship and critical possibilities.

Findings – The results include a Web application for interactive storytelling and an analytical framework for analyzing how affordances of digital media can support literacy practices with unique critical possibilities. The final study showed how interactive stories can function as critical discourse models, simulations of social realities which support analysis of phenomena such as social positioning and the use of power.

Originality/value – Previous work has insufficiently spanned the fields of learning sciences and literacies, respectively emphasizing the mechanisms and the content of literacy practices. In focusing a design-based approach on critical awareness of identity, power and privilege, this research develops tools and theory for supporting critical computational literacies. This research envisions a literacy-based approach to K-12 computer science which could contribute to liberatory education.

Keywords Critical literacy, Computational thinking, Literacy, Design-based research, Computer science education, Computational literacy, Multiliteracies, K-12 computer science education

Introduction

Literacy is about much more than learning to read and write. The practices which emerge within networks of people and texts often have prosaic goals such as conveying messages, documenting agreements and establishing authority, but they can profoundly reshape participants’ cognition, identity practices and social relationships. Ong (2013) argues that privileged access to reading and writing led to the emergence of new social roles high in the status hierarchy, and that widespread literacy in a society “restructures consciousness” (p. 77) by synchronizing frames of reference such as dates, facts and perspectives on the world. In addition to supporting practices which define social roles and relationships, Scribner and Cole (1978) found that reading and writing were associated with changes in individual cognition such as improved abstract communication, memory and language analysis skills (pp. 27-29). It is not necessary to argue for a direct causal link between

This research was supported by a grant from TELOS, Technology for Equity in Learning Opportunities, by Stanford's Transformative Learning Technologies Lab, and by the Lemann Center for Entrepreneurship and Educational Innovation in Brazil. The authors are deeply grateful to the teachers and students with whom they conducted this research, and to the colleagues who have read drafts of this article.
reading and writing and cognitive change, rather they may be seen as tools which have the potential to spur a different developmental path for the individual and for the society (Vygotsky, 1980).

Societies all over the world are increasingly becoming reliant on digital media, sometimes in place of print text. Digital media functions differently from print text (Murray, 2017), so we should expect digital literacies to function differently from print literacies. The designers and technologists who invented personal computing were inspired by not just by technological possibilities but also the possibility of new ways of living (Markoff, 2005). Engelbart (1962) pursued augmented cognition through *bootstrapping*, a reciprocal process by which technological advances allow new forms of cooperation and collaboration, which enable further technological advances. Interfaces such as virtual reality and ubiquitous mobile computing present not just new information channels but also “material exteriority” (Hansen and Hayles, 2000), extensions and transformations of the body which redefine the subjectivities we may inhabit (Haraway, 2006; Lanier, 2010). Nelson’s (1974) development of hypertext was motivated by the possibility of new social, political and economic arrangement. In contrast to the view of computers as primarily information-processing machines, we argue that personal computing and the internet have always functioned as technologies of literacy, making possible networks of humans and computers whose practices transform cognition, identity practices and social relations.

Educators who understand learning as situated in contexts of people, spaces, tools, ideas and purposes (Collins and Greeno, 2011) recognize the importance of bringing students into communities of practice which engage with media in discipline-specific ways. They also recognize the potential of new media to support the development of new cognitive and social structures (Pea, 1985). One goal of this article is to bring together two distinct research communities concerned with pedagogy, literacy and new media. The first includes learning scientists interested in how computers can support thinking and learning. The second includes scholars of critical multiliteracies interested in how the same literacy practices which can be so empowering also reproduce oppressive subjectivities and power hierarchies, and in strategies for resisting and subverting them. The spread of computation into all aspects of our lives, and the growing awareness that “Silicon Valley is not your friend” (Cohen, 2017), makes it urgent that we integrate these two perspectives into an understanding of critical computational literacies.

The learning sciences are concerned with the mechanisms by which people think and learn with technology, individually and as participants in larger systems (Bransford et al., 2000; Nathan and Wagner Alibali, 2010). Building on early socio-cultural theory, the learning sciences have produced functional accounts of literacy (diSessa, 2001), as well as complementary constructs, describing how communities think and learn through interaction with media. These include distributed cognition (Cole and Engeström, 1993; Pea, 1993), activity theory and figured worlds (Holland et al., 2001). Within these theoretical frames, design-based research (The Design-Based Research Collective, 2003; Bang and Vossoughi, 2016) develops new technologies to understand and improve learning. These include Papert's Logo (1980); diSessa (2000) and more recent computational media (Sipitakiat, Blikstein, and Cavallo, 2004; Barab et al., 2005; Resnick et al., 2009).

A second research community on literacy pedagogy is focused on multiliteracies (New London Group, 1996), a term which draws attention to “the multiplicity of communication channels and media, and the increasing saliency of cultural and linguistic diversity” (p. 63). Multiliteracies stands in a figure-ground relationship with the learning sciences approach to literacies, emphasizing sociocultural issues of identity, voice, positionality and power. Bringing attention to the ways dominant literacies also marginalize and disempower
demands taking a critical stance. Freire’s (1968) political activism teaching the poor to read was grounded in a recognition that text-mediated thought was responsible for constituting them as passive subjects incapable of action. Learning to read the wor(l)d means participating in social meaning-making instead of taking meaning as given, realizing that the present world is constituted in certain ways and could have been different, and working toward more just and inclusive futures. (When the term critical is used in this article, it is in this sense.) In addition to contesting the hegemony of dominant literacies, critical multiliteracies aim to re-value marginalized literacies as legitimate meaning-making processes in and out of school (Paris, 2011; Morrell, 2015).

These two traditions have not been sufficiently in dialogue with one another, respectively emphasizing the mechanisms and the content of literacy practices (Bang et al., 2007; Vossoughi and Gutiérrez, 2016). As educators and practitioners within both textual and computational literacy spaces, the authors are interested in studying the material and critical possibilities of textual-computational multiliteracy. This research adopts the methodology of participatory design research (The Design-Based Research Collective, 2003; Anderson and Shattuck, 2012; Bang and Vossoughi, 2016), working iteratively with adolescents to design and build a Web application that supports critical practices, drawing on both writing and programming and collaborating with participants to interpret the results. The starting point is the existing medium of interactive storytelling, which has the affordances of both writing and programming and a history of critical resistance to the sexism, racism and heteronormativity common in mainstream video games (Anthropy, 2012). The research questions guiding these studies are:

**RQ1.** What kind of medium and pedagogy might support textual-computational multiliteracy?

**RQ2.** What practices might emerge in such a textual-computational multiliteracy space?

**RQ3.** How might such practices support critical awareness and resistance to racism, sexism and other oppressive ideologies?

This article is structured around multiple iterations of design-based research developing a Web application for interactive storytelling called Unfold Studio. The next section grounds this research in existing literature and develops a conceptual framework for tracing how the affordances of computational media support different rhetorical practices, producing different meanings and identities and ultimately the possibility of critical activism. Then this framework is used to analyze the iterative development of Unfold Studio, a Web application for interactive storytelling. Finally, the article reports on an interactive storytelling workshop which demonstrated that interactive storytelling can effectively support critical textual-computational literacy. The article concludes with a discussion of how these results contribute to theory and pedagogical practice.

**Background**

**Literacy spaces**

This article considers literacy to be a particular form of situated learning (Collins and Greeno, 2011) in which a network of actors collaborate to do semiotic work. Actors may include people, texts, computers, objects, ideas: anything which engages in meaning-making, or which represents, interprets or mediates meaning, or which is marked as meaningful. The abstract term *actors* is used so the definition can include conventional communities reading and writing texts as well as scenarios where it is harder to distinguish...
between the authors and the media. In computational literacy spaces, computers function as semiotic media but may also introduce new ideas, engage in interpretation and author their own identities. Computer programs are texts written by programmers, but given the right environment, they can also function as authors, interlocutors or as interfaces mediating readings of other texts. Chatbots, online avatars and news feeds have complex relationships with the people who designed them, who control them and whom they purport to represent. The question of agency is urgent, but not easy to answer.

Gee's (1990) term literacy space is used as a near synonym for literacy community or figured world (Holland et al., 2001), following his rationale that it is unproductive to try to define the membership boundaries of a literacy community. Would-be participants may be excluded, while others may become implicated in a literacy space without their consent. A literacy space is similar to a Discourse (Gee, 1990) or an ideology, although the latter two tend to be larger networks with longer histories, so that participants are less able to transform meanings within them. For example, Reyes (2017) documents how students within a school literacy space in the USA were able to contest local stereotypes about what it means to be Asian and author new identities for themselves, but they had to work within more widespread ideologies about Asianness, Whiteness and legitimacy.

What kinds of interactions qualify as semiotic work? First, transforming old meanings into new ones. Holland et al. (2001) describe a process of “symbolic bootstrapping” (p. 38) or “heuristic development” (p. 40) by which actors take up external tools (or ideas or symbols), use them, internalize them as part of their developmental histories and thereby render their environments useful or meaningful in new ways. The pattern for this process is Vygotsky's (1980) account of how people acquire language and build concepts. For Holland et al., Bakhtin's (1981) heteroglossia – the recursive composition of meaning from prior meanings – runs parallel to the Vygotskian process of tool and concept construction. A point Vygotsky, Bakhtin and Holland et al. emphasize is that the generation of new meanings is grounded in and constrained by existing materials which always have histories. Each participant acts from a history of participation which encodes the meanings of other actors, and so continued participation sustains the historical meanings of the system.

The meanings of selves within literacy spaces develop via the same process. This article uses the term identity to denote a model of selfhood one authors and occupies in a literacy space, which exists at the interface “between intimate discourses, inner speaking, and bodily practices formed in the past and the discourses and practices to which people are exposed, willingly or not, in the present” (Holland et al., 2001, p. 32). Drawing again on Bakhtin's dialogic self, Holland et al. describe identity as the negotiated meeting place of unconstrained inner speech and an external subject position made available by social meanings. The subject position specifies the terms by which one is addressable and by which one will answer.

What we call identities remain dependent upon social relations and material conditions. If these relations and material conditions change, they must be 'answered,' and old 'answers' about who one is may be undone (Holland et al., 1998, p. 189).

Analytical framework
This article’s research questions contain an implicit chain of hypotheses:

- that the perceived affordances of a computational medium could shape the rhetorical practices for which it is used;
that these practices could shape the meanings and identities which are thereby enacted; and
that these meanings and identities could open possibilities for critical understanding and activism.

This paper's analysis follows a four-layer analytical framework aligned with these hypotheses: affordances, rhetoric, figured meanings and critical possibilities. Similar to Brooke's (2009) framework for analyzing digital rhetoric at the levels of code, practice and culture, each level permits analysis of the literacy space at a different scale.

The following subsections address each layer in turn, grounding its approach in prior literature and explaining how it is operationalized in this research. It may appear anachronistic to present this framework here, as its form emerged during the iterative design-based research described in the next section, and it was formalized during iterative rounds of grounded theory-based open coding (Charmaz, 1996) described in the subsequent section. However, starting with the framework allows for a more intuitive organization of the results which follow.

Affordances. The development of a literacy medium can be analyzed in terms of its affordances, or the ways it can be used to create meaning. The primary form of meaning-making considered by this research is authors composing texts to shape the experiences of readers. A second form of meaning-making, important to the critical possibilities of literacy, is the way texts can open new possibilities for future authorship. Norman (1999) distinguishes between affordances and perceived affordances – between the actionable properties of an object and those perceived as such by a user. When considering media which support literacy, this means distinguishing between the myriad ways a medium could potentially be taken up in meaning-making, and the subset authors perceive as likely to be recognized by an audience, within the context of the Web application interface. Therefore, this article analyzes the perceived affordances of media through instances of their use.

The two media with which this article is most concerned, i.e. text and code, function differently in supporting meaning-making and therefore offer different affordances. Although in practice literacies are multi- and trans- (New London Group, 1996; Thomas et al., 2007), there is value in distinguishing how reading a text differs from playing a computational artifact such as a game or an interactive map (Aarseth, 1997). One essential mechanism of text is representation. Representation can be achieved in many ways, such as evoking sensory experiences through descriptive language, voicing characters through dialogue and setting the mood through emotional descriptions of the setting. Rather than understanding representation as encoding some objective meaning, this framework takes a reader-response stance, viewing representation as offering provocations and opportunities to the reader. In Rosenblatt's (1968) account of reading as a unique, historically grounded transaction between a reader and a text, each is transformed. The reader's identity is changed through her response to the text, which then reciprocally transforms the text's possible meanings (Barthes, 1981).

One way we can interact with computation is through modeling or simulation. Interacting with computational models is a central practice in science (NGSS Lead States, 2013; Blikstein, 2014) and computer science (K-12 Computer Science Framework, 2016). Interacting with a model can be agent-based, emphasizing how one actor in the system can affect others, or systemic, emphasizing emergent properties (Weintrop et al., 2016). Papert (1980) used the term microworlds to describe computational models or simulations in which one can immerse oneself and learn how the world works through play or exploration. This can lead to authentic, embodied knowledge, more like getting to know someone than
learning a fact. For example, NetLogo (Wilensky, 1999) is an environment for modeling of
dynamic systems. Participating in a NetLogo simulation can help students understand and
predict the behavior of systems from both an agent-based and a system-level perspective
(Wilensky and Stroup, 1999). This article considers microworlds to be both models and
games[1].

Interactive storytelling combines the affordances of text and code in ways that are
difficult to classify, so the modes of interaction described in the previous two paragraphs are
best understood as heuristics for the affordances authors might perceive in interactive
storytelling, as instantiated in the Web application's interface. Much of the participatory
design process developing the Web application was devoted to discovering the ways
authors could use interactive storytelling and refining the interface and pedagogy to make
those affordances more perceptible.

Rhetoric. The second layer, rhetoric, is focused on how authors use the affordances of
media to create meaning. Theorists of digital rhetoric have argued for the importance of this
link: whereas traditional literary criticism could assume some universality to how text
functions, digital media cannot be understood apart from the affordances of its media
(Wysocki, 2004; Bogost, 2007; Brooke, 2009). Digital interfaces such as hypertext, interactive
stories, Instagram and mobile phones have such diverse affordances that the rhetorical
possibilities of each are quite distinct.

The interactive storytelling community has identified several broad categories of
rhetorical moves (Glassner, 2004; Montfort, 2007; Murray, 2017). Ryan (2001) contrasts
immersion with interactivity. Reading a story can involve constructing a world of meanings
around oneself through a transactional reading process. The reader potentially experiences
immersion, a sense of being embodied in and surrounded by that world. In contrast, when
playing an interactive story which functions as a microworld, it does the work of simulation
and its world is perceived as outside of oneself. The player experiences interactivity, with a
heightened awareness of the interface. There may be a tradeoff between these rhetorical
modes in interactive storytelling: the more the story handles the simulation (functioning as a
microworld), the more one can interact with dynamics that are too hard to simulate or which
one could not have imagined. The more the reader is left to do the simulation (as with a
representational text), the more she can experience intimacy and empathy through
immersive embodiment.

This article's analysis of the rhetoric of interactive storytelling extends traditional
reader-response literary analysis to include what Bogost (2006a, 2006b) calls procedural
rhetoric, or the ways computational affordances are used to influence an imagined player.
The focus is on how stories are crafted to make available possible readings. When available,
participants' reflections on their intentions are used to support this analysis. Common
interactive techniques include providing or denying agency to the player, allowing
omnipotent control of the world (for example, allowing the player to choose the reactions of
others or whether it rains) and inserting parenthetical remarks on the player's choices.
Immersive techniques include allowing the player to construct an in-world identity and
structuring choices so that the player becomes morally implicated in the story's events or is
presumed to have given consent to events taking place in the story.

Figured meanings. The third layer, figured meanings, describes the potential effects of
rhetorical techniques used by interactive stories. These may include how others read the
story and respond by reshaping their own identities, as well as by reshaping the sense-
making processes available for reading other texts. Following the earlier definition of
identity as the interface between internal self-conception and externally imposed
subjectivities, changes to figured meanings may expand or contract the kinds of identities
possible within the literacy space. As a concrete example, when a literature class reads texts featuring characters with potentially invisible life experiences such as being immigrants, queer or homeless, these possible selves become more available to students' identities.

In the participatory design process, one common way interactive stories reshaped sense-making processes was by invoking existing genres or developing new genres. Genres include literary genres such as horror, science fiction and role-playing games, as well as what Bakhtin and Holquist (1981) call speech genres, or the “sphere[s] in which language is used [and] develops its own relatively stable types” (p. 60). In their stories, participants used speech genres including quizzes, text messaging conversations and Facebook posts. Story topics, such as family, friends, dating and school, function similarly to speech genres in that they create expectations for the kinds of meanings that will be expressed. To be recognized, an author or speaker must adopt a register, a socially recognized form of communication which indexes some qualities of the speaker (Agha, 2005). By introducing speech genres in their stories, participants pushed for social recognition of new registers within the literacy space.

Critical possibilities. Finally, the fourth layer, critical possibilities, are ways in which stories enact or hold open the possibility of critical change. This article refers to critical literacy practices as those with the potential to enact transformation both within the literacy space (by changing the actors or the sense-making processes) and also beyond the literacy space (Gee, 2004a, 2004b). Fairclough (2004) refers to the performativity of texts as their “causal effects on nonsemiotic elements of the material, social, and mental worlds and the conditions of possibility for the performativity of texts” (p. 225).

When people find themselves within oppressive literacy spaces, where the existing language and cultural materials offer only marginalized subject positions, there are several possible responses. One might refuse to participate, retreating into the space of inner speech where for Bakhtin (if not for Vygotsky), one is free to fashion a self. One might also try to “use the master's tools to dismantle the master's house” (Lorde, 2003), or insist on the inclusion of other materials, for example, by legitimizing vernacular registers (Anzaldúa, 1987). Discovering and using critical strategies depends on understanding how identities are circumscribed by available subjectivities, and how registers are legitimized.

The goal of Freirian critical literacy is to develop this understanding. Freirian critical literacy depends on the representational function of text: once people become aware of the parallels between reading the word and reading the world, they may realize that neither has a fixed meaning, but rather the meanings of each are continually produced within a literacy space, and that the possible meanings are co-produced with one's identity. Of course, as mentioned earlier, it is much easier to open new possibilities for identity and register within a small discussion group than it is within the context of ideologies that span centuries and continents.

The computational affordances of interactive stories support additional critical possibilities (Bogost, 2007; Blikstein, 2008; Garcia et al., 2015). One powerful dynamic which emerged in the participatory design workshops was using interactive stories to model literacy spaces themselves. For example, participants wrote stories allowing the player to experience how one is treated differently when speaking English versus when speaking Spanish, or how two friends in casual conversation can also be engaged in a struggle to position each other. These stories foreground otherwise-latent uses of power within the literacy space, making them visible and accessible for analysis and critique. These stories potentially function as critical discourse models, a particular kind of what Vossoughi (2014) refers to as social analytic artifacts, or “tools that deepen the collective analysis of social problems” (p. 353).
Players of critical discourse models participate in the story's simulated literacy space. At the same time, the player and the story are both actors within a larger literacy space. The idea of a nested literacy space as an actor within a larger literacy space is not new: Bakhtin's (1981) multivocal understanding of texts in dialogue with existing meanings and Minsky's (1988) understanding of minds composed of many agents may each be understood as literacy spaces functioning as actors within larger literacy spaces. However, the distinct affordances of interactive stories (particularly the precision with which one can author them) offer unique critical possibilities.

Interactive storytelling
The goals of this research are to develop media and pedagogy capable of supporting textual-computational multiliteracy, to study the practices that might emerge and to assess their critical possibilities. The starting point is interactive storytelling (a generalization of interactive fiction), a medium authored with text and code to create single-player text-based games and stories. Interactive storytelling had a widespread following from the late 1980s through the 1990s, bounded chronologically by the emergence of personal computers and early access to the internet and its displacement by graphical games made possible by improvements in processors and displays (Labrande, 2011). Over the past several decades, interactive storytelling has retained a small but active community, often articulating feminist and queer critical responses to the ideologies dominant in the literacy space of mainstream video games (Anthropy, 2012).

Two recent works of interactive storytelling illustrate the dynamics the framework presented in the previous section: these stories use their affordances for immersive and interactive rhetorical effect, producing figured meanings and critical possibilities. In 80 Days (2014), a choose-your-own-adventure game loosely based on Jules Verne's novel, the player inhabits the role of Passepartout, valet to a wealthy Englishman who is attempting to circumnavigate a counterfactual nineteenth-century world. In choosing how the story should unfold, the player may have very different experiences depending on how she engages dialogically with other characters, and the ways in which she decides to explore beyond the bubble formed by her employer’s casually racist, sexist and elitist attitudes. In the interplay between interactivity (making strategic choices) and immersion (becoming invested in the lives of other characters), 80 Days functions as a microworld in which the player can discover how richly expansive or foreclosed the world (and one’s self-authored identity) can be, depending on the extent to which one chooses vulnerability and openness in the face of the unknown. The player potentially realizes that winning the game is not the point.

Nicky Case’s Coming Out Simulator (2014), an autobiographical “half-true story about half-truths,” powerfully demonstrates the capability of interactive storytelling to model how linguistic processes produce our social reality and shape how we can act within them. The game replays the evening during Case’s teenage years when he told his parents (or, perhaps, they found out) that he is bisexual. The interface mimics that of a mobile phone, superimposing text message speech bubbles over simple animations and presenting the player with dialogue options. In the prologue, the game highlights the way it functions as a critical discourse model, emphasizing that all the characters remember and respond to everything the player does. As the protagonist struggles to come out to his parents, they are equally committed to preserving their image of him by silencing his attempt at self-redefinition. The story is ultimately about negotiating what it means to be male and to be a good son within a cultural context. It is played through speech acts; the player struggles to
author an identity using language whose categories and meanings are largely under the parents' control.

We engage in these discursive struggles on a daily basis, but because they are fleeting and invisible, they can be difficult to perceive or understand. In contrast to our lived experience or linear narrative, in which we can only follow one path through a space of possibilities, the Coming Out Simulator:

[...] includes dialogue that I, my parents, and my ex-boyfriend actually said. As well as all the things we could have, should have, and never would have said. It doesn’t matter which is which.

The game takes no more than 20 min to play through, and it explicitly invites multiple replays through which a player can map out the space of interactional possibilities. In doing so, the player engages in an epistemic game (Collins and Ferguson, 1993) of modeling how characters position themselves and each other and analyzing the how the game’s reality is shaped by the characters’ speech choices. Both 80 Days and the Coming Out Simulator illustrate the critical potential of interactive storytelling as a medium for textual-computational multiliteracy. Studying language, identity and culture within a computational environment could make it possible to simulate, replay and share these otherwise-elusive phenomena. Reading and writing microworlds in the context of questions usually addressed by literature might imbue computer science, a potentially abstract and impersonal field, with profound personal significance.

Workshops I and II: developing Unfold Studio

This section reports on the initial development of Unfold Studio through Workshops I and II with middle-school students. An iterative design process focused on emergent and imagined interactive storytelling practices helped develop the Web application and the analytical framework described above. These results framed a hypothesis that interactive storytelling could be particularly effective in supporting critical change within and beyond the classroom literacy space. Workshop III designed to test this hypothesis is reported in the following section.

Methods

The workshops developing Unfold Studio took place at a private all-girls' middle school in western USA, at the border of two communities: a wealthy, largely white and East Asian community, and a largely black, Latinx and Tongan community, including many recent immigrants and having a much lower socioeconomic status. Approximately 80 per cent of the students pay full tuition; the majority of students receiving scholarships are Mexican-American and speak Spanish at home. In spite of the structural inequalities that come with charging high tuition, the school's explicit mission is to help its early-adolescent students understand gender, sexuality and race. Students had these analytical registers available to them, but there were also numerous tensions dividing the school that were seldom openly discussed. The first author who led these workshops (hereafter “Chris”) had been a teacher at the school several years prior, granting him legitimacy and trust in the eyes of the faculty. Even though he had never worked with these students before, they also accorded him insider status.

Workshop I included 12 participants who met for 3 h each morning over the course of a school week (15 h in all). The participants were consistently positioned as co-researchers. On the first day, Chris introduced interactive storytelling and an initial prototype of the Web application, saying he did not know what it might be good for. He proposed a writer's workshop-like structure, with mini-lessons targeting writing skills such as incorporating
dialogue, developing character, using sensory detail and structuring plot, as well as computational skills such as expressing stories as sequences and branches, using loops and conditionals and using trees and directed graphs to plan their stories. As the week went on, participants increasingly steered the agenda. The final 30 min of each day were devoted to discussing what worked and what did not work in the lessons, planning the following days and critiquing the Web application. Each night the authors updated the Web application guided by the participants' feedback. The authors collected ethnographic field notes, logged interactions with the Web application, analyzed the stories participants wrote and asked the participants to engage in reflective writing at the end of each session.

Workshop II was embedded within a two-week summer program designed to build community amongst students receiving full scholarships to the same school. The participants were 16 incoming sixth-grade students (age, 11-12 years), all bilingual speakers of Spanish and English. Chris also had the opportunity to plan and co-teach the workshop with an incoming eighth-grade student, a college sophomore majoring in creative writing (both alumnae of the same program), and the students’ future humanities teacher. This group designed a sequence of three introductory story prompts, and otherwise prioritized time for writing and workshopping stories with optional mini-lessons.

The college student in particular engaged the participants with a powerful writerly presence. In speech and writing, she allowed herself to be vulnerable and direct in her perceptions, unafraid to follow thoughts even when they approached topics that felt taboo. One participant noted:

I was struck by the difference between her real story and my skeleton of a story. Juli’s (all participant names are pseudonyms) had an authenticity, a sensory immediacy that was so vivid it was almost uncomfortable.

The authors repeatedly noted participants authoring their identities by writing about real issues (and sharing their stories) after Juli opened space to do so.

Development of the application was guided by participants' emergent and imagined interactive storytelling practices. To understand these, the authors analyzed field notes and participants' written reflections using a grounded theory affinity analysis (Iba et al., 2017) in which data were clustered and interpreted. The resulting patterns described how participants recognized affordances in the medium, how they used the medium rhetorically toward figured meanings and how they used the medium for critical analysis.

Results

Developing affordances. The initial prototype allowed users to browse the library of published stories and to select an individual story to play or edit. When viewing an individual story, two panes were presented for editing and playing, in much the same manner as Scratch (Resnick et al., 2009). Stories are written in Ink, a language originally designed to support the development of 80 Days and released as open source in 2016. Figure 1 shows a story written in Ink, demonstrating several of its key structures. The narrative is divided into knots containing anywhere from a phrase to several paragraphs of prose. Typically, a knot ends with several options to be presented to the player, whose consequences are redirects to other knots.

In response to participant feedback, three iterations were released over the course of the week. The participants delighted in suggesting improvements and reporting bugs and checking that the changes had been made. The most frequently requested features were support for additional typefaces, font styles, font sizes and colors. Participants also requested support for visual elements such as background colors and incorporating
animated GIFs into their stories, user accounts with the ability to mark stories as private and site navigation elements such as the ability to search, star favorite stories and add short descriptions to stories for ease of browsing. Participants came across several situations in which they needed to toggle between a prose-like interface and a code-like interface. Several participants attempted to use accented Spanish characters and emoji in their stories. At the time, Ink did not support unicode, so these characters caused the application to crash.

In both Workshops I and II, participants’ use of computational affordances was broad but shallow. In Workshop II, 81 per cent of stories used branching to create nonlinear structure, but less than 20 per cent used other computational affordances such as variables, logic, or randomness, despite mini-lessons focused on exploring their use. One frequently cited constraint was that error messages were not user-friendly. Nevertheless, many participants described the computational concepts as interesting and generative.

Affordances spurred imagination for rhetoric and figured meanings. Over the course of Workshop I, participants explored the nature of interactive storytelling, positioning it with respect to stories, games and computer programs, and playing and critiquing several published examples. These conversations generated a great deal of aspirational discussion about what could be created with interactive storytelling. When participants imagined “being lost” in a story world, “going deeper into characters” and “a world of different types of people,” the experiences they described were immersive readings. These generally relied on familiar representational affordances of literary text (also called literary elements) such as sensory detail, characterization and point of view.

Participants also imagined using interactivity, though not for new forms of digital media such as games, agents or simulations of systems. Instead, participants often described a desire to concretize an imagined reader’s experience, to connect with the reader and to shape her reading of the story. One participant wrote, “I love it because [...] I feel like a writer [...] you give the viewers the opportunity to make the story their story by choosing what path

Note: The story's source code is on the left and the running story is on the right.

Figure 1. Final interface for Unfold Studio
they want to take.” This participant imagined using interactive storytelling in much the same way as Graff et al.’s “They say, I say” (2006) pedagogical strategy for helping participants understand voice in academic writing, using choices and nonlinear branching to represent possible readings. Helping students learn that reading is an active, interpretive process (Rosenblatt, 1968) rather than one of passive uptake is a central goal of English/Language Arts and Freirian critical pedagogy, made particularly challenging because the student usually cannot observe the practices adopted by the expert reader.

Participants imagined writing various simulations of dialogical encounters, particularly in digital media. They frequently requested the ability to add emoji, GIFs and speech bubbles simulating text message conversations. Participants’ desires to appropriate these speech genres into their own storytelling (e.g. embedding texts or tweets into an interactive story) are evocative of Bakhtin’s (1981) analysis of how the novel fixes and stylizes other genres, while making them more “free and flexible [. . .] permeated with laughter, irony, humor, elements of self-parody” (p. 7) by putting them in dialogue with other voices. Social media is already dialogic and constantly changing, but platforms allow only prescribed usage aimed at commodifying the voices, identities and attention of users. The unauthorized uses imagined by these workshop participants point to new strategies for critical engagement with computational media.

Critical discourse models. One prominent theme which emerged in Workshop II was talk about language. One of the initial story prompts asked participants to model a real-life incident where each participant had a different experience. Many participants drew on their bilingualism to model how social reality changes when speaking English and when speaking Spanish. For example, one story considered a multilingual family context in which the player can position herself by drawing on a spectrum of linguistic practices. Others explored discrimination which results from speaking Spanish in public, or represented a multilingual inner voice. These stories functioned as dialogue-based microworlds simulating both heteroglossic possibility and exclusionary language ideologies (Rosa and Burdick, 2016).

The classroom sustained a lively and thoughtful conversation exploring these questions. Field notes and participants’ post-workshop reflections document the importance to participants of constructing, revising, playing and discussing these interactive stories. The authors’ field notes repeatedly describe participants juxtaposing their desks so that each could face her laptop screen and her partner while reading and writing – an embodied enactment of the literacy space. The stories functioned as what Vossoughi (2014) calls social analytic artifacts, “tools that deepen the collective analysis of social problems” (p. 353). However, the specific method by which the stories functioned, as microworlds simulating discourse within a literacy space, led to a new concept: critical discourse models. Critical discourse models may offer experiences which cannot be enacted with immersive representational texts, and introducing them as actors within a literacy space may offer new possibilities for critical literacies pedagogy.

Workshop III: toward critical multiliteracies

The result of Workshops I and II was a medium capable of supporting textual-computational literacy practices through interactive storytelling, and a hypothesis that these practices could be particularly effective in supporting critical change within and beyond the classroom literacy space. Following Schwartz et al.’s (2008) suggestion that design-based research ought to move from innovative design toward efficiency, we designed Workshop III to test this hypothesis.
Methods
Workshop III was set in a large comprehensive high-school drawing students from the same communities as the previous studies. The workshop took place over two weeks (15 h total) in students’ English and Sociology classes, linked together in a self-selected academic track focused on social justice. Twenty-three high-school seniors participated; two additional students did not return consent forms and were excluded from analysis. The first author co-designed the workshop with several of the students during meetings and email exchanges prior to the workshop, and co-led the workshop with two of the students’ teachers. His collegiality with the other teachers conferred legitimacy and some authority, while his intentional self-positioning as a researcher beyond the school’s authority may have contributed to participants’ willingness to discuss and write about charged topics.

The workshop was again structured as three introductory story prompts introducing affordances of interactive storytelling, followed by writer’s workshop time devoted to participants working on, sharing and revising their interactive stories. The final prompt addressed three ideas related critical understandings of discourse:

- **Models of personhood**: In any social world, people inhabit models of personhood which define what kind of person they will be seen as and what they can do.
- **Performativity**: Identities are dynamic, not static. We perform our identities, bounded by models of personhood, but possibly also redefining models of personhood.
- **Register**: A socially recognized way of speaking based in a model of personhood.

Participants were offered several options for exploring the following ideas:

- create an oppressive social world where the possibilities of speech are limited for the main character;
- create a world where the main character is assigned a model of personhood based on how he/she speaks; or
- create a world where the main character subverts a model of personhood he/she is assigned.

As they discussed these prompts, many participants offered examples from their own lives. Many participants continued to develop stories based on these prompts for the rest of the workshop.

Participants’ stories were analyzed through multiple iterations of qualitative coding. Initially, the authors relied on open coding via a grounded theory approach (Charmaz, 1996). Over multiple passes of coding, writing integrative memos (Emerson, Fretz, and Shaw, 2011) and refining the coding scheme, the outlines of the analytical framework (presented above) emerged. Two major groups of computational affordances were FLOW, affordances for controlling the flow of story execution, and STATE, affordances for keeping track of the past and using it to affect the future. Of the various textual affordances, DIALOGUE was chosen for further analysis because of its importance for critical analysis of social discourse. Codes for rhetorical strategies were grouped into IMMERSION and INTERACTIVITY. Figured meanings were grouped into three high-level categories: LIFE, for speech genres pertaining to family, dating, friends, jobs, school and drug use; LITERARY GENRE, prominently including science fiction, horror and genres of games such as role-playing games and puzzles; and SOCIAL MEDIA, for stories adopting registers characteristic of texting or other online discourse. Finally, CRITICALITY contains codes for when categories such as race, gender, sexuality and social class are either marked (for example, by noting a
character's skin color or manner of speech) or explicitly addressed by the story. Co-occurrence of codes within stories was calculated to analyze associations between factors at different layers of the analytical framework. The final codebook is included in Appendix 1[2].

Results

The results, shown in Table I, validate the construct of critical discourse models as an effective structure for exploring critical ideas in real-life contexts. Of the stories with critical engagement (CRITICAL), 62 per cent dealt with familiar settings such as family, friends, dating and school (LIFE), 62 per cent used immersive techniques (IMMERSION), 85 per cent used interactive techniques (INTERACTIVITY), 69 per cent used computational affordances to control story flow (FLOW) and 92 per cent relied on dialogue (DIALOGUE). These are precisely the properties hypothesized to be effective for exploring critical ideas.

There was a clear distinction between the use of computational affordances for controlling the flow of story execution (FLOW) and the use of variables to maintain state (STATE). FLOW affordances tended to occur in stories using both immersive (47 per cent) and interactive (71 per cent) techniques, while STATE affordances were seldom used in immersive stories (17 per cent) and more often used in interactive stories (42 per cent). While FLOW stories engaged with LIFE (41 per cent) and LITERARY GENRE (47 per cent) figured meanings, STATE stories focused predominantly on LITERARY GENRE. (The sets of stories coded with LIFE and LITERARY GENRE were disjoint.) Finally, 53 per cent of FLOW stories featured critical engagement (CRITICAL), compared with only 31 per cent of STATE stories. Broadly, these results suggest that FLOW affordances were often important components of critical discourse models, while STATE affordances were more often games or puzzles set in fictional worlds, whose figured meanings had lower stakes.

One particularly interesting set of stories were set in the speech genres of text messaging and social media. These stories effectively made use of affordances and rhetorical strategies such as acronyms, iconic textual effects, emoji and pacing – either the staccato of brief exchanges or ellipses denoting significant pauses. The affordances shared by texting, social media and the interactive storytelling medium have become infrastructural (diSessa, 2000) to identity in many youth cultures. Holland et al. (2001) emphasize that identities are authored within literacy spaces dependent on material conditions; as these conditions change, identities must be articulated anew. The prevalence of social media as a space of youth identity practices may make interactive storytelling particularly valuable for authoring identities and critical analysis of the literacy spaces in which they are authored.

Case study: enacting critical change

A case study of one participant's experience in the workshop shows the potential critical discourse models have to enact change in and beyond the literacy space. In an introductory survey, Leanne describes herself as female, biracial and primarily a speaker of standard English with African-American vernacular English at some family gatherings. Her pre-survey responses suggest a rich history of textual literacy practices, and very little history with computation. Leanne experiences gender-based discrimination frequently and racial discrimination daily. Leanne was an active, but not extremely vocal, participant in the workshop. Some days she sat by herself and worked, other days she sat with a small group of peers. She participated regularly in the quiet conversation taking place while participants read and wrote. In a survey halfway through the workshop, Leanne described having trouble deciding what to write about:
<table>
<thead>
<tr>
<th>Code (count)</th>
<th>Flow (%)</th>
<th>State (%)</th>
<th>Dialogue (%)</th>
<th>Immersion (%)</th>
<th>Interactivity (%)</th>
<th>Life (%)</th>
<th>Literary genre (%)</th>
<th>Social media (%)</th>
<th>Criticality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow (17)</td>
<td>100</td>
<td>29</td>
<td>65</td>
<td>47</td>
<td>71</td>
<td>41</td>
<td>47</td>
<td>12</td>
<td>53</td>
</tr>
<tr>
<td>State (12)</td>
<td>42</td>
<td>100</td>
<td>58</td>
<td>17</td>
<td>42</td>
<td>25</td>
<td>67</td>
<td>25</td>
<td>33</td>
</tr>
<tr>
<td>Dialogue (22)</td>
<td>50</td>
<td>32</td>
<td>100</td>
<td>59</td>
<td>68</td>
<td>45</td>
<td>32</td>
<td>27</td>
<td>55</td>
</tr>
<tr>
<td>Immersion (15)</td>
<td>53</td>
<td>13</td>
<td>87</td>
<td>100</td>
<td>73</td>
<td>73</td>
<td>13</td>
<td>33</td>
<td>53</td>
</tr>
<tr>
<td>Interactivity (21)</td>
<td>57</td>
<td>24</td>
<td>71</td>
<td>52</td>
<td>100</td>
<td>48</td>
<td>43</td>
<td>14</td>
<td>52</td>
</tr>
<tr>
<td>Life (13)</td>
<td>54</td>
<td>23</td>
<td>77</td>
<td>85</td>
<td>77</td>
<td>100</td>
<td>0</td>
<td>46</td>
<td>62</td>
</tr>
<tr>
<td>Literary Genre (15)</td>
<td>53</td>
<td>33</td>
<td>47</td>
<td>13</td>
<td>60</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>Social media (9)</td>
<td>22</td>
<td>33</td>
<td>67</td>
<td>56</td>
<td>33</td>
<td>67</td>
<td>0</td>
<td>100</td>
<td>33</td>
</tr>
<tr>
<td>Critical (13)</td>
<td>69</td>
<td>31</td>
<td>92</td>
<td>62</td>
<td>85</td>
<td>62</td>
<td>23</td>
<td>23</td>
<td>100</td>
</tr>
</tbody>
</table>

Table I. Relative co-occurrence of codes (columns) in stories with codes (rows). Critical literacies of text and code.
I grasp that I can pull from my own experiences, but I have trouble picking just one. I know that when I do, it could end up being pretty profound, but I haven't been able to zero in on one concept yet.

In the first week, she wrote a fan fiction-style story set in the world of Dune, which her English class had been reading. She also reported having some trouble with the programming aspects of interactive fiction: "I think I get the gist of it, but an error occurred on the story I did write. I don't know how I can fix these errors; the explanation given when I click on my story is still confusing to me."

At the beginning of the second week, an incident took place which motivated Leanne to write her final story. During a discussion on register, Mr Leo, a white man who was one of the co-teachers, shared an anecdote in which he had imitated the speaking style of several African-American freshmen girls he did not know well. He described how he had intended the interaction as a joke and a way to connect, but they were extremely offended. He explained that they would not have taken offense if they had been in his class, because they would have known him as someone who likes to tell jokes and understood his intentions. No participant responded to this anecdote at the time, but Leanne addressed it in her final reflection:

I was intrigued by the discussion on personhood and register, and the idea of having no way out of the stereotypes you are assigned. Mr Leo gave an example of register discussing some insensitive things he said to a couple of African-American freshmen girls, saying the same offense wouldn't happen in this class. I disagreed, since I was very offended by his words and conclusion, but didn't say anything in order to not make people (especially Mr. Proctor who I didn't know very well) "uncomfortable." I regretted not saying anything, and I think that regret evolved into my piece.

An excerpt from the story Leanne wrote in response to this incident is included below. Readers may play it as an interactive story by visiting https://unfold.studio/stories/1063. The story uses only FLOW computational affordances, offering nonlinear paths through knots (e.g. === start ===) of content. Knots end by presenting the player with options (*) which divert the story to another knot (→). When readers play the story, they see only the text contained within knots and their options. Readers may also choose to view the story's source code in parallel (see Figure 1).

Leanne's story functioned as a critical discourse model. It enacted critical transformation within the classroom literacy space by indirectly voicing an otherwise-unspoken contestation of Mr Leo's anecdote. The story is written in second-person, which highlights the player's role as a character in the story. The story begins by introducing Angela, a girl in your (presumably high school) class who has “dark skin, curly hair, and lively eyes, and she's probably the smartest kid in your grade” (Line 3). The first choice the player is presented with is not an action in the story but a choice of identity: deciding whether to like Angela or whether to be jealous of her. This is the beginning of a process by which the player self-authors an identity within the story, and thereby makes herself a witness and complicit in the action. Regardless of the player's choice, the plot continues with another girl, making a joke about how Angela is so poor that she probably has to wear second-hand clothing to school. The player is offered at least one opportunity to defend Angela against the attacks (Lines 37, 60, 70). However, the player's protests are ultimately ineffectual. Standing up for Angela only gets the player ostracized too (Line 75).

Angela: A Bystander’s Story (Lines 50-73)
50  === yet ===
The rumor gets back to Angela within the week. You keep an eye on her, wondering if she is affected, but she seems to be handling
all of this with grace. Angela continues to be a strong student, and is kind to the people around her. The girl who began the rumor becomes aggravated. One day you hear her shout at Angela, “You filthy n*gger!” Angela stares at her, stunned, her unaffected smile completely gone. She looks wounded. You know that a horrible line has been crossed. *You are shocked into silence.

--> bullyA

60 *You tell the girl, “That's too far. Stop it.”

--> bullyMe

=== bullyA ===

The new game spreads like wildfire. Angela is shoved on the staircase, excluded from everything. The word “N*gger” echoes through the halls and is written in Sharpie on her backpack. It scares you how awful your classmates are being to Angela. You never thought she was a bad person; what has she done to deserve this? More and more, Angela reacts to the taunts with a rag doll's indifference. Her empty eyes haunt you. At first she looked wounded, but now she looks dead.

70 *You've had enough. You tell your friends to stop.

--> bully

Me*If you say anything, you know they'll come after you too, so you say nothing.

--> fear

Regardless of the player's earlier choices, she ends up in the pivotal sequence at Line 50, excerpted above. Angela's refusal to be provoked by the mean joke causes the girl to escalate her attacks, which finally overwhelm Angela's composure. The player is faced with two options recapitulating the earlier structure, to either remain in shocked silence or to defend Angela. If the player does nothing, the attacks escalate (Line 63). Angela still does not respond to the attacks, but now instead of “handling all of this with grace” (Line 52), she reacts “with a rag doll's indifference” (Line 68). The player is offered one final chance to stand up for Angela as she goes from looking “wounded” to “dead” (Line 69). If the player stands up for Angela at either opportunity, she reaches one of two possible story endings. For the first time, the player is positioned racially – as a “White N*gger” (Line 77). The story concludes with the player's removal from school. Alternatively, if the player chooses not to defend Angela, the player's life goes on – the player dissociates from Angela and disappears from the narrative altogether. It becomes clear that the racial attacks were always about power: “The girl who started it all is on top of the world, the queen bee; the school is hers now. Angela is so far below her in the social hierarchy that she never has to feel jealous again” (line 88). In both outcomes, the player watches as Angela is dehumanized: she no longer makes eye contact with the player, is described as “IT” (Line 76) and is described as the other students' “plaything” (Line 93). In both endings, the narrator relates that Angela attempts suicide.

In neither outcome is the player able to meaningfully protect Angela; the only choice is whether to be stripped of one's whiteness and subject oneself to the same attacks, or to remain silent. The story offers no way out, nor does it make available exculpation or solidarity. Both immersion and interactivity are at work: the high stakes and appearance of choice invite replay and exploration of the action space in an attempt to find a solution. The parts of the story where the player's action is narrated rather than selected, and particularly the vivid imagery showing
Angela’s facade cracking, eyes dimming and progressive dehumanization, implicate the player in the story. A player might feel both inside and outside the story, and the effect could be a transformation of the player’s identity within the world of the story as well as in the real world. In refusing to grant the player agency, the story possibly enacts critical change in its literacy space, for example, by arguing against the easy answers of an anti-bullying curriculum claiming that bullying can be stopped by a simple act of moral courage.

The story functioned as a critical discourse model within the workshop. As Leanne wrote in her closing reflection, writing it was an opportunity to think about the dynamics by which someone can be trapped and silenced in models of personhood and a way for her to speak back against her teacher’s assumptions. While this did not lead to a confrontation with Mr Leo or a reckoning with his joke, Leanne’s story did contribute to change. In the final days of the workshop, the authors noted participants increasingly frequently sitting in pairs or triads, reading and discussing stories. One question on the closing survey asked participants to write an open-ended reflection on new ideas they considered in the workshop. In total, 44 per cent of respondents discussed ideas related to criticality or empathy, often using forceful language to describe their interaction with the stories. For example, one participant wrote, “I enjoyed learning about how interactive fiction can drive people to explore/understand limits and effectively force people to empathize.”

Discussion
The design-based research reported in this article yielded fruitful answers to the initial research questions. The first two studies explored the potential uses of interactive storytelling and developed the Web application’s affordances to better support participants’ aspirations for the medium. Workshop III validated critical discourse models as tools for critical engagement and documented the role of textual and computational affordances. In each workshop, the participants were involved in planning the workshop, framing the questions and interpreting the results. Their participation was essential to the validity of the findings and also to ensuring that the research process could play an equitable role in the literacy spaces which were the focus of study.

This research makes three primary contributions. First, the iterative participatory design process yielded a refined Web application capable of supporting textual-computational multiliteracy in a writer’s workshop environment. Development followed (and continues to follow) workshop participants’ imagined uses for interactive storytelling, so that the workshops themselves were a critical process of enacting imagined conditions. Unfold Studio has been publicly released and has already been used in several schools, including several months in an introductory computer science course, as well as in a teacher preparation program and professional development workshops focused on computer science and critical literacies. Framing interactive storytelling as an introductory approach to programming may help teachers of computer science view their subject as a literacy, as a resource for their students’ existing multi- and transliteracy practices, and as an opportunity to support their students’ critical perspectives. The expansion of Unfold Studio as an online literacy space and its efficacy as an introduction to programming are topics of ongoing research (Proctor, 2019).

Second, this research finds theoretical common ground between learning scientists and scholars of critical literacy, and it demonstrates the importance of continued dialogue between these communities. There is substantial overlap between the constructs used by each field to study situated, distributed and mediated meaning-making; this article’s framing of literacy spaces may be useful for integrating the perspectives and concerns of each field. This work is particularly urgent given the current emphasis on increasing access to K-12 computer science. Over the past two decades, the computer science education community has devoted increasing
attention to educational equity (Margolis and Fisher, 2003; Margolis et al., 2010; Kafai and Burke, 2013), focusing on unequal participation in computing and factors causing it. This work is important but incomplete. Too often, efforts toward more equitable participation do no critical interrogation of the practices in which they seek to increase participation. There are direct parallels to the decades of work by scholars and practitioners of English/Language Arts grappling with how and when to teach dominant American English. Computing too has a culture of power (Delpit, 1988) and a tendency to view other sense-making practices through a deficit lens (Moll et al., 1992). What might computer science look like if it centered culturally sustaining pedagogy (Paris, 2012), with its insistence on criticality?

Finally, this research yields the concept of critical discourse models, which may be particularly effective in supporting critical awareness in the multi- and transliteracies prevalent in youth culture today. Interactive storytelling offers affordances useful for modeling and analyzing in-person and digitally mediated discourse. Because they make phenomena visible and concrete, critical discourse models may be especially useful in contexts where people have different amounts of experience thinking about these concepts. For example, Unfold Studio was used in a course on literacies in a teacher preparation program where some participants had chosen to become teachers to combat the oppression they experienced on a daily basis. For others who had grown up in privilege, goals of social justice were not grounded in lived experience. These are often particularly difficult settings in which to discuss power and privilege. Those coming from habitual and seldom-questioned privilege may feel they need a nonjudgmental space to consider new self-understandings, and feel threatened by critical positions. However, for people who experience marginalization on a daily basis, the insistence on a safe space which excludes uncomfortable truths can be experienced as an act of erasure by dominant literacy practices. In such a space, interactive storytelling can be used to model experiences such as microaggressions. Those who do not understand how an offhand comment can shatter one’s sense of safety and belonging can play and replay an interactive story, empathizing but also coming to understand the mechanisms by which microaggressions can cause harm.

Conclusion
As our society completes its shift from print text to digital media, schools must prepare youth to participate in new forms of literacy. It is clear that computational media do not necessarily lead to the just, peaceful and inclusive social structures imagined by the pioneers of personal and social computing. Indeed, computational media have enabled powerful new forms of surveillance, control and amplification of oppressive ideologies. If we want to support youth in self-authorship, critical agency and participation in designing socio-technical futures, it is imperative that our schools cultivate critical computational literacies which center the lives and identities of the community. The design-based research reported in this article yielded a concrete step toward that goal. As Unfold Studio makes its way into classrooms and writing clubs, future research will continue the project of developing a medium well-suited to supporting critical literacy practices.

Notes
1. There is much more to say about the nature of games, which is not taken up in this article. In the early 2000s, there was a fierce debate between narratologists and ludologists about whether games ought to be analyzed using the machinery of literary criticism. The competing framings of games ran roughly parallel to the representational texts and microworlds presented here.
2. Appendix 1 is available online at http://chrisproctor.net/research/unfoldstudio
References


Further reading

Corresponding author
Chris Proctor can be contacted at: cproctor@stanford.edu

For instructions on how to order reprints of this article, please visit our website: www.emeraldgrouppublishing.com/licensing/reprints.htm
Or contact us for further details: permissions@emeraldinsight.com
Teachers’ goals predict computational thinking gains in robotics

Eben B. Witherspoon
Department of Education,
University of Pittsburgh – Learning Research and Development Centre,
Pittsburgh, Pennsylvania, USA, and

Christian D. Schunn
University of Pittsburgh – Learning Research and Development Centre, Pittsburgh, Pennsylvania, USA

Abstract

Purpose – Computational thinking (CT) is widely considered to be an important component of teaching generalizable computer science skills to all students in a range of learning environments, including robotics. However, despite advances in the design of robotics curricula that can teach CT, actual enactment in classrooms may often fail to reach this target. This study aims to understand whether the various instructional goals teachers hold when using these curricula may offer one potential explanation for disparities in outcomes.

Design/methodology/approach – In this study, the authors examine results from \(N=206\) middle-school students’ pre- and post-tests of CT, attitudinal surveys and surveys of their teacher’s instructional goals to determine if student attitudes and learning gains in CT are related to the instructional goals their teachers endorsed while implementing a shared robotics programming curriculum.

Findings – The findings provide evidence that despite using the same curriculum, students showed differential learning gains on the CT assessment when in classrooms with teachers who rated CT as a more important instructional goal; these effects were particularly strong for women. Students in classroom with teachers who rated CT more highly also showed greater maintenance of positive attitudes toward programming.

Originality/value – While there is a growing body of literature regarding curricular interventions that provide CT learning opportunities, this study provides a critical insight into the role that teachers may play as a potential support or barrier to the success of these curricula. Implications for the design of professional development and teacher educative materials that attend to teachers’ instructional goals are discussed.

Keyword Robotics

Paper type Research paper

Introduction

Computer science education is now widely considered to be an integral part of a well-rounded K-12 science, technology, engineering and mathematics (STEM) education. In the USA, the “Computer Science for All” initiative urges that computer science (CS) learning...
opportunities be provided not only within specialized elective classes or after-school clubs but also in general education classes that offer these experiences to every student (Smith, 2016). In part, this policy shift is driven by a growing need for some base level of competence in computing for students to remain competitive in a job market that increasingly requires computational knowledge and skills, regardless of career trajectory. The USA Bureau of Labor Statistics (2017) predicts that the fastest growing careers in the coming decade are likely those that will require some degree of computational literacy and the ability to use computers and programming logic to solve problems in a variety of applications. Educational researchers have sometimes used the term computational thinking (CT) to describe this particular twenty-first-century skill. A canonical and complete definition of CT remains unsettled in the literature, leading some to advocate for the pragmatic approach of identifying core and peripheral concepts of CT; core aspects typically include decomposing problems, designing algorithmic solutions and abstracting those solutions to multiple contexts (Voogt et al., 2015). Therefore, while many definitions of CT exist, most emphasize the importance of drawing on heuristics from the field of computer science to solve problems and applying the knowledge and skills of computer science to solve problems across a variety of contexts and subjects (Barr and Stephenson, 2011; Wing, 2006).

Educational psychologists have studied the possible cognitive benefits of using computer science in K-12 to develop generalizable problem solving skills like CT for decades (Klahr and Carver, 1988; Pea and Kurland, 1984). In particular, specific CT concepts from computer science such as “commands execute in sequence,” “conditional statements determine if and when to pass control of the program to a new set of commands” and “programs repeat the commands a set number of times or until a condition is met” may be generalizable across programming languages and contexts. However, still relatively little is known about particular pedagogical practices that might be linked to effective instruction in this class of generalizable computational skills.

Robotics is one field that has been studied by educational psychologists as a learning environment that could potentially provide authentic opportunities to learn generalizable computer programming skills in an applied setting (Grover and Pea, 2013). Relatively recent advances in the design of educational technologies, informed by research in the learning sciences, have shown promise in providing students with generative learning experiences that may help develop the generalizable programming knowledge and skills prioritized by initiatives such as Computer Science for All (Lye and Koh, 2014). For example, block-based graphical programming languages can reduce syntax errors, allowing novice programmers to focus on the logic of their programs control structure (Kelleher and Pausch, 2005; Robins et al., 2010). Specific to robotics educational curricula, virtual simulations such as those used in the current study can reduce the mechanical errors often introduced by physical robots, thereby reducing the cognitive load of beginning programmers. Such simulated virtual curricula have been proven to teach programming and physical robotics, but more efficiently (Liu et al., 2013b). Additionally, there is emerging evidence that certain features of these virtual robotics learning environments may be associated with measurable gains in generalizable CT knowledge and skills (Witherspoon et al., 2017, 2018).

In the context of Computer Science for All, educational robotics programs present themselves as a convenient option for school districts aiming to take up this initiative. In the past few decades, robotics programs have become almost ubiquitous in middle schools and high schools, both in elective after-school programs and more recently in compulsory education as the required technology becomes more broadly affordable (Melchior et al., 2005). However, in many K-12 settings, technology-rich programs like robotics are implemented within Technology Education (“Tech Ed”) departments, which have
historically focused on vocational training in specific and often localized industrial technologies, and are taught by teachers with varied training and experience in computer programming (Shields and Harris, 2007). Teachers in these classrooms often hold a broad range of teaching certifications, from Business, Computer and Information Technology to Career, Technical and Agricultural Education, and most teachers who are tasked with teaching robotics are unlikely to have received specific professional development targeted toward teaching either computer science or CT (Ericson et al., 2008; Stephenson and Gal-Ezer, 2010). As use of robotics for teaching CT expands, limitations in teacher expertise may act as a bottleneck on positive learning outcomes.

The critical role of teachers
It is well established that teachers play a critical role in student learning and attitudes; however, a variety of mechanisms may mediate these effects in technology-rich environments. Generally speaking, teacher beliefs about pedagogy and content interact with the written curriculum to determine ways that instructional materials are implemented, often creating disparities between curriculum as designed and curriculum as enacted (Remillard, 2005). Particularly, in technology-rich environments, external barriers such as lack of training and hardware or software resources and internal barriers such as confidence with the material, valuation of technology and beliefs about how students learn could inform how teachers interpret and enact curriculum (Ertmer et al., 1999). Teachers are known to vary greatly in their understanding of CT and their attitudes toward integrating it into their classrooms, but CT educational opportunities are often limited to preservice computer science teachers (Yadav et al., 2014). Further, inquiry and project-based STEM reform curricula like those often found in robotics aim for students to construct knowledge through largely self-directed exploration and require substantial shifts in teaching practice from traditional, direct instruction methods (Schneider and Krajcik, 2002). Therefore, it is likely that large variance exists in the particular curricular focus and pedagogical approach to CT instruction across robotics programs, as well as in learning outcomes for students.

In addition to influencing achievement, variation in the way curricular materials are presented in robotics classrooms may also influence another important outcome of Computer Science for All: students’ attitudes toward programming (Witherspoon et al., 2018). Maintaining students’ motivation to engage in programming activities may be particularly difficult in non-elective classrooms (e.g. in middle schools that require all students to take a course in technology education); research suggests that overall student valuation of STEM subjects tends to decline beginning in the middle-school years (Wigfield and Eccles, 2000). However, it is possible for well-supported activities in middle school to maintain individual interest levels, which can predict long-term, self-generated engagement through college (Harackiewicz and Hulleman, 2010; Hidi and Renninger, 2006). Other attitudinal interventions that can be linked to pedagogy, such as identity development through engagement in authentic tasks of the discipline and fostering students beliefs about their ability to do programming, can also predict students’ achievement and continued participation in computer science majors and careers (Collins, 2006; Engle, 2006; Lent et al., 2016). Therefore, examining students’ attitudinal responses to different pedagogical approaches while using a robotics programming curriculum could also offer important insights into effects on both students’ achievement and persistence.

Teacher goals
Understanding teachers’ instructional goal setting could provide one useful framework for predicting how teachers activate resources in ways that differ from the designed curriculum.
By “instructional goal,” we mean a specific statement that expresses what students should learn in the language of a particular discipline, and it is situated within a student-driven model of how learning progresses (Stein and Meikle, 2017). Teachers’ goals that are explicitly stated and refined into sub-goals at the lesson planning stage may improve the design of instructional activities that increase student achievement (Hiebert et al., 2017). Research has also suggested that instructional goal setting may be an emergent process that is responsive to a particular context (Aguirre and Speer, 1999).

In learning environments like Tech Ed classrooms, where a relatively recent shift in focus to computing technology has led to the acceptance of a broader variety of teacher certifications, teacher rotation between multiple topical units, and a range of new tools and curricula, departmental goals can often be complex and ill-defined. It is likely that Tech Ed teachers hold multiple instructional goals simultaneously, and that those may at times conflict with the written curriculum, determining which goals are implemented in the classroom (Davis et al., 2016). Therefore, rather than circumventing these challenges with “teacher-proof” curricular materials, it is necessary for curricular designers to consider curricular enactment as a “local phenomenon that arises as a result of a number of factors, including [...] teachers’ goals, local constraints, and teachers’ pedagogical values” (Drake and Sherin, 2006). Curriculum developers aiming to teach CT may benefit from understanding the goals endorsed by Tech Ed robotics teachers implementing their curriculum to better provide strategies to deal with potentially competing goals. Additionally, understanding Tech Ed robotics teachers’ instructional goals could aid in the design of professional development that ensures that all teachers have the knowledge and skills needed to align their instructional activities with higher level curricular goals.

Therefore, while robotics curricular materials may be designed with intent to provide opportunities to learn CT, these goals are often altered by teachers on the ground during moment-to-moment interactions with students. Particularly, in-service Tech Ed robotics teachers may hold alternate goals for their classrooms based on past experiences (i.e. general goals about problem solving vs specific goals about CT), and under the pressure of a complex and novel learning environment may be more likely to revert to prior pedagogical practices that are more familiar (i.e. focusing on performance outcomes like building the physical robot vs learning outcomes like understanding computational concepts). This variation in goals can lead to variation in student learning by classroom, even when teachers have relatively similar experience, teach in similar learning contexts and are using the exact same curricular materials.

A better understanding of the importance teachers place on the different goals they have in these classrooms may help predict when and how these differences in enactment may manifest and the effect that they have on student learning. Importantly, this information will be useful for curriculum designers to account for in development of teacher instructional materials and professional development. In this study, we examine how teachers’ ratings of the importance of instructional goals around CT in middle-school robotics classrooms are related to student learning of CT. Specifically, we were interested if we would find differences based on CT instructional goals for Tech Ed teachers using the same virtual robotics programming curriculum, suggesting that these goals may be contributing factors to discrepancies in enactment that produce variation in students CT learning opportunities.

**Methods**

**Sample**
We examined the development of CT in robotics classes in which all students in the school were enrolled, within schools across multiple regions of the USA. All human subjects’
research received Institutional Review Board (IRB) approval prior to the commencement of the study. The analyses presented here examine a sample of \( N = 206 \) middle-school-age students (\( M_{\text{age}} = 12.3, SD_{\text{age}} = 1.1 \)) within classrooms in four school districts, focusing on teachers with clearly differentiated instructional goals (described below). Students in this sample predominately identified as white (72 per cent), with multi-racial (18 per cent) and Asian (6 per cent) making up the next two largest groups; the rest of the students either answered “Other”, “I don’t know” or were from a variety of groups (e.g. Indian/Middle Eastern, Native American/Pacific Islander) that each made up less than 1 per cent of the data. Unlike elective robotics classes which are often predominately male, robotics classrooms in which all students in the school were enrolled consisted of a relatively evenly split by self-identified gender (51 per cent female). Many of the students in these courses (69 per cent) had some prior experience with robotics before, but the majority of students (77 per cent) were engaging with this particular virtual robotics curriculum for the first time.

In addition to student assessments, we also distributed multiple rounds of weekly surveys to \( N = 10 \) teachers across the USA, which asked them to rate their instructional goals for their classes on a weekly basis. Overall, our response rate from the teacher surveys was about 47 per cent. All of the responding teachers had earned a master’s degree, were certified in a range of specialties closely related to Technology Education (e.g. Business, Computers and Information Technology; Career and Technical Education; and Technology Education) and had a relatively high number of years of teaching experience overall (\( M_{\text{years}} = 12.6, SD_{\text{years}} = 6.0 \)). Additional details on the four teachers selected for further analysis are presented in a later section.

Curricular materials

The robotics curriculum used here, developed by Carnegie Mellon University and Robomatter, involves a sequence of lessons in robotics programming utilizing a visual programming language, ROBOTC Graphical [Figure 1(a)]. On average, instruction with the curriculum ran for about 10 weeks and included 24 mini-lessons across four units including topics both specific to robotics (i.e. basic movement, sensors and repeated decisions) and core CT concepts (i.e. abstraction, decomposition and systems thinking). Earlier versions of

![Figure 1.](image)

**Note:** In this task, using if/else statements, loops and sensors, students program the robot to sort flags onto the left or right conveyor belt based on the color, which is dynamically assigned.
a similar virtual robotics curriculum have been reported on in previous studies (Witherspoon et al., 2017, 2018). The curricular materials incorporate elements which were designed to support efficient learning and transfer of generalizable computational skills: procedural scaffolds (worked examples, guided videos), dynamic mini-challenges, a visual programming language, and Robot Virtual Worlds (RVW), a virtual robotics programming environment designed to emphasize the programming aspects of robotics while maintaining student interest and engagement [Figure 1(b)]. These features reflect a constructionist approach to instruction, in which learners build increasingly complex programmed solutions and construct an understanding of the requisite programming principles (Papert, 1980).

First, to provide a shared context for each unit, students are provided with a short introductory video to frame the subsequent lesson activities. These videos are learner-paced and present visual support together with a conversational narrative around the key concepts to reduce extraneous processing and foster generative processing (Mayer, 2008). Partial scaffolding (Puntambekar and Hubscher, 2005) is introduced by way of questions to check students’ understanding, step-by-step instruction on a conceptually related robotics programming activity and a brief post activity quiz to assess understanding, followed by the open-ended application of these skills within a game-like challenge in the virtual programming environment, allowing students to apply their knowledge more independently.

Students can iteratively test modular programmed solutions with simulated VEX IQ robots in a three-dimensional virtual platform. Finally, these solutions are “remixed and reused” (Brennan and Resnick, 2012) to complete more complex virtual challenges, in which learners must apply their previous programming knowledge to problem solving tasks that foreground CT principles and abstraction, decomposition and systems thinking. To solve these challenges, students used a programming language called ROBOTC Graphical [Figure 1(a)] to develop programmed solutions. ROBOTC Graphical has a visual programming language interface, intended to allow students to focus on the broader logic of programming while deemphasizing the particular syntactic requirements of more traditional programming languages.

By representing robotics challenges in a virtual environment, this curriculum offers affordances over physical robotics programs by reducing the potential frustration and distractors of mechanical error, enabling students to focus on higher-level computational principles of programming. While physical robots may have some advantages, a study by Liu et al. (2013a) found that students using an earlier version of this technology achieved learning gains in programming content equivalent to students using physical robots, but in significantly less time. Further, simulating robot movement reflects an authentic engineering practice (Michel, 2004), and virtual robots are also less expensive than physical ones, allowing the benefits of the curriculum to reach a broader population where the costs of physical robotics curricula can be prohibitive.

**Measures and procedures**

**Teacher instructional goals.** To understand which instructional goals teachers were emphasizing in these classrooms, we distributed a weekly online survey to teachers throughout the semester in which they were using the curriculum. These surveys were developed through pilot studies consisting of pre-lesson goal setting activity conducted with a small group of local robotics teachers using the same virtual curriculum. From these pilot studies, we noted that only some teachers were setting goals related to core CT concepts that were included in the curriculum. These teachers included goals such as “students learn that
in a conditional loop, the condition determines when/how long the commands repeat,” while other teachers identified goals such as “students will complete lesson activities 1-3.” The resulting surveys used in the current study asked teachers to rate the importance of a set of goals focused on specific CT learning outcomes (e.g. “During class this week, my goal was that students would learn [...] that programs execute commands in sequence”) on a three-point Likert scale ranging from (1) least important to (3) most important (see Appendix 1 for sample teacher goals measures). Additionally, teachers were asked to provide demographic information such as level of teaching experience, teaching certification and prior exposure to the curriculum. These surveys were purposefully kept relatively brief to promote survey completion.

Overall, teachers were given nine opportunities to respond to the survey over the course of a semester. From the total group of ten teachers who received the survey, four teachers provided a sufficient number of responses \(n \geq 5\) across all items to generate a reasonably robust measure of their average rating of each goal, and so these four teachers were purposively selected for additional analysis. The four teachers selected for final analysis were all white, male and had a similar level of teaching experience \(M_{\text{years}} = 13.8, SD_{\text{years}} = 3.0\). Overall, teachers tended to rate most goals as at least moderately important; based on the distribution of teachers’ responses, we used a median split to group them into two categories: “Low CT,” consisting of two teachers who had an average overall rating of CT goals of 2.5 or below \(M_{\text{rating}} = 2.1, SD_{\text{rating}} = 0.3\) across 11 combined ratings, and “High CT,” consisting of two teachers who had an average overall rating of CT goals of 2.5 or higher \(M_{\text{rating}} = 2.8, SD_{\text{rating}} = 0.4\) across 12 combined ratings. In other words, teachers who typically rated the goals as only moderately important versus teachers who typically rated the goals as most important; this difference in ratings was a large effect size (Cohen’s \(d = 2.2\)). Both High CT teachers held Technology Education certifications, while one Low CT teacher held a Business, Computers and Information Technology certification, and the other held both Career and Technical Education and Biology certifications. In each group, one teacher reported having approximately four years of experience with the curriculum, while the second teacher in each group was using the curriculum for the first time.

**Computational thinking assessments.** After grouping the four teachers based on their rating of CT goals, we then examined the pre- and post-test scores of students in each of these teachers’ classrooms to see if there were significant differences in learning as measured by the assessments of CT for students in Low CT teachers’ classrooms \(n = 57\) and students in a High CT teachers’ classrooms \(n = 149\); Table I.

The primary outcome measure was an externally created CT assessment used as a post-test. It consisted of five multiple choice items that were adapted for a robotics context from the Exploring Computer Science – Principled Assessment of CT (Goode and Margolis, 2011). These assessments were specifically created using evidence-centered design to assess knowledge, skills and attributes associated with CT practices[1].

An alternative assessment was needed that could be used to verify equivalence of both general programming skills and CT skills across classes before instruction, as well as avoid test-retest effects. We had previously developed such an assessment that contained programming and CT items (see Appendix 2 for sample assessment items). These items were developed to target three core programming concepts common across a range of accepted frameworks of programming and CT, sequences, conditions and iteration (College Board, 2016; Computer Science Teachers Association, 2016; Bienkowski et al., 2015) and have been shown in prior work to be a reliable measure of students programming and CT knowledge.
Our school district partners requested that we reduce class time required for an assessment that is only establishing equivalence at the class level. Therefore, students received one of four randomly assigned sections of the CT assessment at pre-test; each of the four sections of the pre-test consisted of five multiple choice items each. The overall average Armor's $\theta$ was $\theta = 0.44$ for the pre-test and $\theta = 0.74$ for the post-test. Relatively low theta values are common for relatively short assessments that are intended to cover a range of concepts. When corrected to account for possible attenuation of correlation caused by the measurement error, the pre-post correlation was $\rho = 0.60$ (Fan, 2003).

**Attitudes toward programming.** Additionally, students also completed a short attitudinal survey prior to the pre and post-test exams, with 12 items that asked students about their interest, competency beliefs and development of identity in computer programming (see Appendix 3 for sample survey items). Different scales were used across these items as a strategy for slowing respondents down and getting them to read each item more closely and to allow for different measures of intensity (i.e. frequency of experiences vs strength of endorsement). Interest was gauged through four items (e.g. “I wonder about how computer programs work,” Cronbach’s $\alpha = 0.87$), rated along a four-point Likert scale (e.g. “Never” to “Every Day”). Four items gauged level of identity as a programmer (e.g. “My family thinks of me as a programming person,” $\alpha = 0.88$), rated along a four-point Likert scale (e.g. “NO!” to “YES!”). Competency beliefs were gauged through four items (e.g. “I am sure I could do advanced work in programming,” $\alpha = 0.83$) rated along a six-point Likert scale (e.g. “Strongly Disagree” to “Strongly Agree”). Based on a prior pilot survey, which suggested that students struggled to accurately rate their competency prior to obtaining some knowledge of the content, only these items were measured using retrospective pre-items (i.e. students were asked at post to rate both their competency at the beginning of the curriculum and their competency now; Pratt *et al.*, 2000). Attitudinal measures at both pre and post were significantly correlated with each other, but not so high as to be redundant measures. For ease of interpretation across these different scales, prior to analyses all attitudinal measures were converted to a proportion, with the lowest rating as 0 and the highest rating as 1.

<table>
<thead>
<tr>
<th>Measures</th>
<th>Teacher Goal Groups</th>
<th>Teacher characteristics</th>
<th>Student characteristics</th>
<th>Student assessments</th>
<th>Student surveys</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CT Low ($N = 57$)</td>
<td>CT High ($N = 149$)</td>
<td>$t$</td>
<td>95% CI</td>
<td></td>
</tr>
<tr>
<td><strong>Teacher characteristics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teacher rating of CT</td>
<td>2.1 (0.30)</td>
<td>2.8 (0.38)</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>Teacher exp. (years)</td>
<td>14.0 (1.4)</td>
<td>13.5 (4.9)</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td><strong>Student characteristics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student robotic exp.</td>
<td>33%</td>
<td>30%</td>
<td>–0.5</td>
<td>–18%, 10%</td>
<td></td>
</tr>
<tr>
<td>Student CS2N exp.</td>
<td>9%</td>
<td>28%</td>
<td>–2.9**</td>
<td>–31%, –6%</td>
<td></td>
</tr>
<tr>
<td>Student age (years)</td>
<td>11.7 (1.2)</td>
<td>12.5 (1.0)</td>
<td>–5.2**</td>
<td>–1.2, –0.6</td>
<td></td>
</tr>
<tr>
<td><strong>Student assessments</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-test</td>
<td>2.2 (1.2)</td>
<td>2.3 (1.1)</td>
<td>0.7</td>
<td>–0.21, 0.46</td>
<td></td>
</tr>
<tr>
<td><strong>Student surveys</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Competency Beliefs</td>
<td>0.53 (0.20)</td>
<td>0.51 (0.20)</td>
<td>0.6</td>
<td>–0.04, 0.08</td>
<td></td>
</tr>
<tr>
<td>Identity</td>
<td>0.59 (0.13)</td>
<td>0.52 (0.19)</td>
<td>1.5</td>
<td>–0.02, 0.17</td>
<td></td>
</tr>
<tr>
<td>Interest</td>
<td>0.66 (0.15)</td>
<td>0.60 (0.19)</td>
<td>1.2</td>
<td>–0.04, 0.15</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:** **$p < 0.01$;** *For teacher data, a dash (–) is shown in place of t-statistics and 95% CI because of low teacher sample size*
Analyses
Average pre-test scores of students in the two High CT teachers’ classrooms were compared against the average pre-test scores of students in the two Low CT teachers’ classrooms using a simple t-test, as well as other teacher and student characteristics, to establish that the high CT and low CT groups were comparable. Post-test scores were analyzed using ANCOVA, comparing differences in average post-test scores in the two groups while controlling for the pre-test score, age and curricular experience, to increase statistical power by accounting for individual differences in students’ pre-tests, and to account for slight differences in pre-group composition on those variables. Finally, motivation variables were also measured using an ANCOVA of post-survey scores, controlling for pre-survey scores, age and curriculum experience.

Results
We first examined whether initially the two groups of students in the low CT and high CT classrooms were relatively comparable on the assessment of CT. A Levene’s robust test for homogeneity of variance showed that there were no significant differences in variance between the two low CT and high CT groups at pre-test, $F(1, 204) < 1, p = 0.79$. Further, no significant differences were found in pre-test scores ($t = -0.75, p = 0.45, d = 0.11$), between students in a classroom taught by a teacher with a low CT rating ($M_{score} = 2.2, SD_{score} = 1.2$) or students in a classroom taught by a teacher with a high CT rating ($M_{score} = 2.3, SD_{score} = 1.1$; Figure 1).

Critically, on the post-test, being in a classroom taught by a high CT rating teacher was associated with significantly higher scores ($M_{score} = 2.6, SD_{score} = 1.3$) than being in a classroom with a teacher who gave a low CT rating ($M_{score} = 2.1, SD_{score} = 1.4$; $t = -2.67, p < 0.01, d = 0.41$; Figure 2). Thus, we have evidence of differential gains by teacher goals even when the same curriculum is being used.

However, the two groups were not fully equivalent by background. To account for small differences found in age and prior experience with the curriculum between the low CT and high CT groups, an ANCOVA was conducted on post-test scores, controlling for pre-test scores, age, and prior experience with the curriculum. Even with these controls, students in the high CT group showed higher mean post-test scores than students in the low CT group, $F(4,198) = 2.90, p = 0.06$, although these differences were no longer statistically significant,

![Figure 2.](image)

Differences between student scores by teacher rating of CT goals (with SE bars) on (a) pre-test and (b) post-test, and (c) post-test with controls for pre-test, age and prior experience

Notes: $ns = $ not significant; $p < 0.10$; $*p < 0.05$; $**p < 0.01$
and the effect size was reduced to $d = 0.29$, meaning the two distributions overlap approximately 88 per cent (Figure 2).

Importantly, aligning with the priority of Computer Science for All in a robotics classroom in which all students in the school are enrolled, our results also show that only in high CT classrooms, girls had a significantly higher score [$F(4,141) = 3.95, p < 0.05, d = 0.30$] on the post-test ($M_{score} = 2.8, SE_{score} = 0.1$) than boys ($M_{score} = 2.4, SE_{score} = 0.2$), even when controlling for pre-test scores, age and curriculum experience. Thus, while in our sample, significant differential gains in CT between the two groups were not found overall when including these additional controls, having a robotics teacher that endorsed CT goals shows significantly higher learning gains for women relative to men.

For the final set of analyses, we examined differences in attitudinal measures for students in classroom with low or high CT teachers. Overall, at pre-test, there were no significant differences between the two groups in competency beliefs ($t = 0.62, p = 0.54$), identity ($t = 1.2, p = 0.23$) or interest ($t = 0.70, p = 0.48$). At post-test, an ANCOVA revealed that while there were no significant differences between the two groups in competency beliefs [$F(4,192) = 1.22, p=0.27, d = 0.16$], the high CT group had significantly higher post-survey scores in both identity [$F(4,108) = 6.73, p < 0.05, d = 0.59$] and interest [$F(4,108) = 10.88, p < 0.01, d = 0.71$] when controlling for pre-survey, age and curriculum experience (Figure 3). Importantly, these higher scores represent a relative maintenance of programming identity and interest from pre-test scores for those students in the high CT group, while students in the low CT group largely experienced significant declines in both programming identity ($t = -2.81, p < 0.05, d = 0.41$) and interest ($t = -3.74, p < 0.01, d = 0.46$).

**Discussion**

Overall, our results show that when teachers endorsed CT as a critical instructional goal, their students had gains in CT and also had greater maintenance of positive attitudes

---

**Figure 3.**
Post-motivation by teacher rating of CT goals (with SE bars), controlling for students’ pre-motivation scores (mean shown as dotted line), age and curriculum experience.

**Notes:** ns = not significant; $^\sim p < 0.10$; $^*p < 0.05$; $^{**}p < 0.01$
toward programming. Particularly, girls in mixed-gender robotics classrooms with a teacher who endorsed CT instructional goals outperformed boys, suggesting that this re-framing of the instructional focus of these traditionally male-dominated learning environments may help support achievement in programming for young women. Importantly, these differences in outcomes were found across teachers with similar experience and who were implementing the same virtual robotics curriculum. These findings suggest the key role instructional goals play in the development of CT, similar to the mathematics education literature which proposes that teachers’ goals act as a "north star," guiding a variety of instructional decisions (Stein and Meikle, 2017). Further, this study lays the foundation for future work examining how the diverse goals held by educators who teach CT in a broad range of learning environments may determine how designed curriculum materials are adapted during curricular enactment. While the current study did not examine the specific curricular adaptations that were made, this study makes clear that an understanding of the instructional goals endorsed by the teacher is a significant contributor to student learning outcomes, and one that must be accounted for in the design of curriculum. Including teacher educative materials that provide insight into the design of lesson activities, identify aspects of enactment that are critical to the learning goal, and develop own teachers’ capacity to design CT lessons may help teachers avoid adaptations that reduce opportunities to learn CT (Brown and Edelson, 2003; Davis and Krajcik, 2005).

Additionally, these findings suggest a need for ongoing professional development support for teachers that not only provides instruction on the use of the materials but also explicitly attends to the goals of the curriculum and potential areas where these goals may come into conflict with goals held by the teacher. Prior research has suggested that goal-coherent professional development can play a key role in reducing ambiguity and uncertainty and supporting teachers’ sensemaking around new curricular initiatives where conflicting goals may exist (Allen and Penuel, 2015). This may be particularly important in the expanding range of educational programs like robotics in Technology Education environments, which are often tasked with incorporating computer science and CT into their ongoing curriculum. The demands of these new initiatives often represent a large shift for teachers from prior pedagogical approaches and instructional goals with which they are familiar (Schneider and Krajcik, 2002). Without adequate attention paid to the ways in which these goals may diverge from those already in place, initiatives such as Computer Science for All and innovative curricular reforms may experience a bottleneck in their ability to see the desired gains in student learning.

**Limitations**

The inferences that can be drawn from this study are limited by a number of factors. First, the analyses conducted are correlational in nature, and there was no random assignment to experimental condition. Therefore, we cannot be certain whether the combination of exposure to the curriculum and the teacher instructional goals are in fact causing the observed differences in scores, or if other unobserved factors may be contributing to the larger gains for students in the high CT group. In a related way, due to our limited student and teacher sample sizes, analyses were unable to account for nesting within schools or classrooms. We are therefore uncertain that there are not school-level differences that may be contributing to differences in student gains. Another limitation with the current data was the relatively low correlation between our pre- and post-tests due to the practical necessity of a limited number of assessment items. The error introduced when these imperfectly correlated pre-tests were included as a control resulted in our statistical tests for knowledge gains being underpowered.
Second, due to the distributed nature of the classrooms around the USA, we did not have observational measures of instruction, and therefore, we do not know what teachers did to produce the changes in outcome. We also were unable to interview participating teachers to uncover how prior teacher certification programs and professional development opportunities may have influences their instructional goals. However, pilot interviews with local robotics teachers using this virtual curriculum indicate that although all teachers identified “problem solving and learning how to think” as core instructional goals, only those who selected CT goals similar to those in our surveys would enact lessons in ways that emphasized CT relevant features of the curricular activities. For example, while enacting an activity in which students program a robot to move a row of boxes, a teacher who selected a student learning goal that “conditional statements determine when to pass control of the program to a new set of commands” directed students to think about using conditions to generate a program that allows the robot to account for different distances between the boxes. Future research should explore the ways in which teachers introduce activities, guide class discussion and respond to student questions/struggles as possible vehicles of the effects of teacher goals on student learning outcomes (Stein et al., 2008; Stein and Meikle, 2017).

Conclusion

While prior studies using a similar virtual robotics curriculum have demonstrated that students may gain generalizable programming knowledge and skills from these learning experiences, here we show that even within similar classrooms using the exact same curriculum, differences may appear, and that teachers’ instructional goals may be a significant contributor to these differences. Future work would benefit from gathering a larger teacher sample, which would allow us to statistically account for different contextual factors, such as nesting effects within different schools and how certification may play a role in robotics teachers’ conceptualization of their instructional goals. Further, additional development of the survey of instructional goal setting and qualitative interviews and classroom observations with teachers could provide additional insight into the mechanisms through which these goals manifest in classroom activities, how teachers conceptualize goals around CT in the classroom and what framing of these goals may be most productive for teaching students generalizable programming knowledge and skills.

Notes

1. Sample items can be found at: https://pact.sri.com/resources.html
2. Armor’s $\Theta$ and polychoric correlations are similar to Cronbach’s $\alpha$ and Pearson’s correlations, respectively, but are more appropriate for binary data (item correct vs item incorrect; see Panter et al., 1997).

References


Appendix 1. Sample teacher goal items

Rate the following goals as Least Important to Most Important to your class this week. Then, indicate how often you spent time during class on the following topics.

You do not need to mark goals that are not applicable. If there are other goals you had that are not listed, use an "Other" box and briefly describe them.

“During class this week, my goal was that students would learn...”

<table>
<thead>
<tr>
<th>Goal Description</th>
<th>Least Important</th>
<th>Important</th>
<th>Most Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>That programs execute each command in order from top to bottom, unless otherwise directed.</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>That programs use conditional statements to determine if and when to pass control of the program to a new set of commands.</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>That programs will repeat the commands inside a looping structure either a set number of times, or until a condition is met.</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

Appendix 2. Sample assessment items

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Items</th>
<th>Concepts</th>
<th>Items</th>
<th>Example content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robot Programming</td>
<td>10</td>
<td>Sequences</td>
<td>3</td>
<td>What sequence of movements will get the robot to the end of the maze?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conditions</td>
<td>4</td>
<td>At what distance sensor value will the robot stop moving?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Loops</td>
<td>3</td>
<td>Which actions will the robot repeat if the bumper sensor is pushed in?</td>
</tr>
<tr>
<td>CT</td>
<td>10</td>
<td>Sequences</td>
<td>3</td>
<td>Will the removal of this line of the program change the display on a heart monitor?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conditions</td>
<td>4</td>
<td>At what combination of blood pressure readings will this heart monitor emit an alarm?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Loops</td>
<td>3</td>
<td>Which of these two programs will identify the correct blood pressure in the least number of iterations?</td>
</tr>
</tbody>
</table>

Table AI. An overview of the dimensions and programming concepts in the bank of items used for the pre-assessment
Sample robotics programming item

Take a look at the program plan below. How will each individual line of code be run once it is programmed?

Line 1: Move forward for 5 seconds, at 100% speed
Line 2: Turn left 1 rotation, at 50% speed
Line 3: Move forward for 5 seconds, at 50% speed
Line 4: Turn right 1 rotation, at 50% speed

Select one:
- Only the first command runs
- The commands are run in order according to their line numbers
- All commands run at once
- The commands are run in a random order

Which of the following is true about conditions?

Select one:
- They must always end up either true or false
- They represent decision-making logic in a program
- You can write a condition that is always true or always false
- All of the above

Sample computational thinking item

Scenario: Personal Fitness Devices

Personal fitness devices use electronic sensors to continuously monitor and track data about a user’s health such as steps taken, calories burned, and heart rate.

The BP-Sure company is developing a new feature for their fitness device that also measures the user’s blood pressure, using sensors that detect a user’s heartbeat. When the heart pushes blood through the arteries, the device records “Pressure 1”, and when the heart is resting, the device records “Pressure 2”.

The device can determine if a user’s blood pressure is in the Normal, Medium or High range, by comparing blood pressure readings to the chart below.

Use the chart below to answer questions #19, #20 and #21.

(continued)
A new programmer on the team writes the following series of steps to determine the display when a user is in the “Normal BP” range:

(Line 1) IF \( p1 \leq 120 \) AND \( p1 \leq 121 \) AND \( p2 \leq 80 \) AND \( p2 \leq 81 \)

(Line 5) THEN set display = "Normal BP"

Which lines can be removed to make the code more efficient, while not changing the code output?

Select one:
- O Line 1 and Line 4
- O Line 2 and Line 3
- O Line 2 and Line 4
- O Line 1 and Line 3
Appendix 3. Sample survey items

Sample competency belief items

<table>
<thead>
<tr>
<th>“I am sure that I can learn programming.”</th>
<th>Strongly Agree</th>
<th>Somewhat Agree</th>
<th>Agree</th>
<th>Disagree</th>
<th>Somewhat Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>With what I knew on the FIRST DAY of the course…</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>With what I know TODAY…</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“I could get an A on a programming assignment in class.”</th>
<th>Strongly Agree</th>
<th>Somewhat Agree</th>
<th>Agree</th>
<th>Disagree</th>
<th>Somewhat Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>With what I knew on the FIRST DAY of the course…</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>With what I know TODAY…</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“I am sure that I could do advanced work in programming.”</th>
<th>Strongly Agree</th>
<th>Somewhat Agree</th>
<th>Agree</th>
<th>Disagree</th>
<th>Somewhat Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>With what I knew on the FIRST DAY of the course…</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>With what I know TODAY…</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

Corresponding author
Eben B. Witherspoon can be contacted at: ebw13@pitt.edu
Youth perspectives on their development in a coding community

Ricarose Roque
Department of Information Science, University of Colorado Boulder, Boulder, Colorado, USA, and

Natalie Rusk
MIT Media Lab, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

Abstract

Purpose – Many initiatives are seeking to engage children in learning to code. However, few studies have examined how children’s engagement in learning and using coding develops over time. This study aims to seek young people’s perspectives on what they viewed as important in their long-term participation in a coding community.

Design/methodology/approach – This study identified youth with a high level of participation and who demonstrated emergent leadership in the Scratch online community. Using methods from qualitative research on youth development, individual interviews were conducted in which these youth were asked about memorable moments in their participation and how these experiences influenced them.

Findings – While each young person described a unique pathway and perspective, this study identified key experiences that motivated their participation, influenced their development and inspired their emergent leadership. These experiences included opportunities to learn through exploration, to receive feedback from peers, to engage in creative collaboration and to contribute to the community.

Practical implications – This study discussed these findings in light of previous research on youth development, and it suggests that building on practices and principles from research on youth programs can help more young people become engaged in developing broader skills with coding.

Originality/value – Youth highlighted experiences that enabled them to express their ideas, to build relationships, to help others and to see themselves in new ways. Their perspectives expand beyond the predominant focus of coding initiatives on computational thinking and problem-solving skills to also support social, leadership and identity development.

Keywords Coding, Community, Leadership, Youth, Scratch, Youth development

Paper type Research paper

Introduction

More and more schools and afterschool programs are offering opportunities for young people to learn to code. Some researchers and educators argue that as young people are learning to code, they are “coding to learn,” engaging in computational thinking concepts and practices that will help them solve problems and understand the digital world that surrounds them. Kafai and Burke (2016) expanded on these ideas to advocate for computational participation, emphasizing the role of young people actively engaging and contributing within a social context to enrich and deepen their experiences with coding.

This material is based upon research supported in part by the National Science Foundation (Grant number 1348876).
Over the past decade, young people around the world have been actively participating in creating with code within the social context of the Scratch online community. Since the Scratch programming language and online community launched in 2007, it has become a dynamic and international community with youth (often between 8 to 16 years of age) creating and sharing thousands of interactive projects each day. Scratch community members not only engage in computational ideas but also develop broader skills and experiences through their deepening participation in the community (Resnick et al., 2009). Youth have contributed beyond the initial expectations of the Scratch Team at MIT who developed the platform (Roque et al., 2016a). Many youth are taking the initiative to help others, such as answering questions, giving feedback, and creating collaborative activities (Brennan et al., 2010). In this paper, we want to highlight the ways that youth have emerged as leaders to help others and to create opportunities for others to learn. They are not only “coding to learn” but also coding to express themselves, connect with others, and contribute to the community.

We focus on the perspective of youth to understand what influenced their deepening participation and emergent leadership. To examine development over time, we interviewed members of the Scratch community and asked them to identify memorable moments in their participation and how these experiences influenced them. Through our analysis, we identified key experiences that mattered to these youth. These key experiences involved taking on new skills, relationships, interests and roles. Their perspectives reveal a trajectory of participation with key moments that supported their creative process, motivation and contributions in the community.

We situate these experiences using literature from youth development, which examines how to support youth to gain broad skills, including social and emotional development. Similar key experiences have also been found in research from youth programs and youth development, which we use to provide a lens in understanding their deepening participation with a coding environment (Strobel et al., 2008). We conclude by suggesting ways to design coding experiences and environments to better support young people’s meaningful participation and broader development.

This paper contributes to ongoing conversations on what matters when engaging young people in learning to code. Our paper contributes to these conversations with the youth perspective, from youth who have had long-term and active participation in a coding community with other young people.

**Background**

Increasingly, educators and researchers are recognizing the importance of young people going beyond traditional academic learning to develop broader skills to lead productive and healthy lives (Aspen Institute National Commission on Social, Emotional, and Academic Development, 2019; National Research Council, 2013). These life and career skills include the ability to take initiative, to collaborate well with others and to take on leadership and responsibility (Partnership for 21st Century Skills, 2016).

Within the field of youth development, researchers have examined how afterschool and other youth programs can support the development of a broad range of skills, such as social-emotional, collaboration and leadership skills. Salusky et al. (2014) studied adolescents who were carrying out projects within community-based youth programs. They found that youth developed leadership and responsibility skills when they took on roles with some structure yet in which they were able to make decisions and solve problems. Successful participation in a role motivated youth to want to take on new roles and responsibilities in the future.
Similarly, Eccles and Gootman (2002), in a consensus report on youth development, emphasize the importance of providing youth spaces where they can learn to take initiative:

Positive development is not something adults do to young people, but rather something that young people do for themselves with a lot of help from parents and others. They are the agents of their own development. To foster development, then, it follows that settings need to be youth centered, providing youth—both individually and in groups—the opportunity to be efficacious and to make a difference in their social world (p. 103).

A consensus report of youth development experts, commissioned by the National Academies, identified key aspects of youth programs that support positive development (Eccles and Gootman, 2002). These include opportunities to experience physical and psychological safety, supportive relationships with peers and adults, a sense of belonging and feeling valued, encouragement to build skills and develop confidence in one's abilities and make a contribution to one's community. Many youth program developers have focused on designing afterschool and other learning environments to meet the needs and interests of young people from underserved and nondominant groups, to address socioeconomic inequities (Carnegie Council on Adolescent Development. Task Force on Youth Development and Community Programs, 1992; Barron et al., 2014).

Although most research on youth development has been conducted in face-to-face settings, some researchers have looked at how youth can develop broader initiative and leadership skills in online spaces. Bers and Chau (2006, 2010) examined how a small 3D virtual community became a safe space for experimenting with decision-making, self-organization and civic conversation. Other researchers have observed collaboration and leadership skill development in participants in chat rooms (Bellerose et al., 2016) and online gaming communities, such as World of Warcraft, Everquest and Minecraft (Ito et al., 2009; Martin and Steinkuehler, 2010; Kafai and Burke, 2016).

Youth developing skills by learning to code
Over the past decade, educational initiatives have been rapidly expanding to provide opportunities for more young people to learn to code within schools (Margolis and Goode, 2016), libraries (Martin, 2017), community centers (Pinkard et al., 2017) and other settings (Kafai et al., 2011). Initiatives such as Hour of Code, CSEdWeek and AfricaCodeWeek have aimed to expose young people to coding by providing short activities. These introductory experiences can improve attitudes toward computer science (Phillips and Brooks, 2016). Some programs, such as CS First and Code Club, provide a sequence of activities and learning materials to scaffold learning of more advanced coding concepts (Braun and Visser, 2017; Mouza et al., 2016).

Many of these educational initiatives make use of Scratch, a visual, block-based programming language and creative coding environment designed to enable young people to create a wide variety of projects based on their interests. The design goals of Scratch are to support the development of not just coding skills but also broader skills including creative thinking, problem-solving and collaboration (Resnick et al., 2009; Resnick, 2013). While many studies of Scratch have focused on students' learning of specific computational concepts, some researchers have looked at the development of broader skills (Denner et al., 2019). For example, Koh (2013) found that adolescents using Scratch experienced a sense of agency from the ability to tinker, remix and share projects on topics of interest to them. Similarly, Ke (2014) observed that middle-school students demonstrated persistence in problem-solving when designing and coding their own math games in Scratch.
While programs are being developed and adopted to span the age ranges, few studies have looked at how children become engaged in learning and using coding over long periods of time (Barron et al., 2014). We leverage youth development principles to better understand how learning environments can support deepening engagement with coding over time and how broader skills can develop when these experiences are embedded in a community. In an earlier study, we applied youth development principles to understand how four youth members took on predefined roles in the Scratch online community and how they learned skills such as taking on responsibility, developing teamwork, taking others’ perspectives and expanding visions for how they can contribute (Roque et al., 2013). In the current study, we apply youth development principles to examine the long-term trajectories of youth who engaged in sustained participation with coding in the Scratch online community. To understand what supports youth to deepen their participation and to develop broader skills, we focused on what mattered to these youth and how their experiences influenced them.

Context: Scratch online community
Since launching in 2007, Scratch has grown to a dynamic online community for young people to create and share interactive projects with others from around the world (Resnick et al., 2009). Scratch is available in more than 40 languages and has become one of the top programming platforms for young people to learn to code.

Previous studies have examined the participation of Scratch online community members through creative collaborations (Brennan et al., 2010; Roque et al., 2016b), leadership roles in the online community (Roque et al., 2013) and civic engagement (Roque et al., 2016a). At the time of this writing, more than 20 million active members, primarily between the ages of 8 and 16, have created and shared more than 40 million projects (Figure 1).

Young people can create interactive media projects such as games, animations and stories with the Scratch coding environment (Figure 2). Moreover, young people can share their projects on the Scratch website. When they share a project, other participants can leave comments, love or favorite their project and look inside the project to see and play with the project code. Participants can also remix projects, adding media or changing the code to create, and share their own versions. They can also create studios where they can invite others to curate a collection of projects and engage in studio comments.

Young people use Scratch in a variety of settings around the world, such as schools, homes, youth programs, libraries, makerspaces and community centers. For some youth, the Scratch online community becomes a space they participate in weekly or even daily, making projects as well as friends (Brennan et al., 2010).

Methods
Our study design leverages theories of youth development that recognize youth as being producers and agents of their own development (Eccles and Gootman, 2002). Building on these theories, studies often ask youth directly about their learning experiences to investigate their development (Dawes and Larson, 2011; Strobel et al., 2008). We designed an interview protocol to ask youth about key moments in their experiences, an interview strategy used by other youth development researchers to understand youth development over time and to identify qualities that mattered to youth (Strobel et al., 2008). Moments are not isolated events, but points in time that are influenced by past moments and influence future decisions and actions.
Development in a coding community

Figure 1. Scratch website homepage

Figure 2. Scratch programming editor (with a project open)
Participants
We identified youth through our own participant observation in the online community as members of the Scratch Team at MIT, where we were able to observe and facilitate community-wide activities. One of the co-authors has worked on the Scratch Team since its initial development, while the other author joined in 2010. The roles and experiences of both authors on the Scratch Team provide deep contextual knowledge to examine in-depth and long-term participation in the online community and to reflect on the designed features and structures of the Scratch community. In addition to contributing to the design and educational outreach of Scratch, both authors were engaged in the online community design and maintenance, which involved weekly design and community meetings to discuss emergent activities within the Scratch community, to make decisions about website features and community policies and to create resources to support Scratch members’ online participation. Both authors have also been involved in research activities to study creative collaboration, leadership roles, motivation and civic engagement in the Scratch online community (Roque et al., 2016a; Roque et al., 2016a, 2016b; Roque et al., 2013, Rusk, 2016).

To investigate how some youth became deeply engaged in the community, we selected youth who demonstrated initiative and leadership in the ways they mentored, enabled, mobilized or supported other members of the community to create and share with coding activities without the intervention of the Scratch Team or other educators. During the spring and summer of 2015 and 2016, we contacted members who were widely recognized by Scratch Team members and the online community through exceptional projects, activities and/or leadership. Beyond creating and sharing projects, they catalyzed community-wide trends or led large collaborative activities. For example, one young girl mobilized 17 other members to create a multi-animator project, where each member created a small animation that she then combined into a larger project. Another young boy ran a workshop during a Scratch conference – an international gathering of educators, developers and researchers – serving as one of the few youth presenters. Another young girl applied for and participated in a summer internship with the Scratch Team. We saw these activities as signals of deep participation, broad development and emergent leadership.

The process of recruiting these youth for participation in interviews involved sending a message in the online community, sharing a description of the interview process, asking for participant agreement, obtaining consent forms and arranging for an online interview with the option of an online video call or phone call. Over the course of the year, we arranged and conducted in-depth interviews with eight youth who had a track record of deep engagement in the community. The participants ranged in age from 12 to 20 years and had participated in Scratch between 1.7 and 8 years. Four of the participants were from within the USA and four were from other countries, including the UK, South Africa and Indonesia. Five of the participants were female. All youth signed a research participant agreement, and those under 18 also submitted a consent form signed by a parent. Table I summarizes the backgrounds of participants, summarizing participants’ key moments and trajectories in Scratch. Below are case portraits for each participant, summarizing their trajectories and key moments:

- **Phoebe**, age 12, was introduced to Scratch by a teacher at school in the fourth grade. She became more interested as she was “messing around” with it during free time, making simple games with her friends. Many of her key moments involved organizing and mobilizing Scratch community members around creative collaborations, including designing and initiating a variety of contests and collaborative animations. Over time, the collaborations she organized and managed
became more complex, including providing ways for other Scratch members to fulfill roles where they would help others and celebrate others’ accomplishments.

- **Dean**, age 15, was introduced to Scratch by his teacher in school and started by “messing around.” His key moments in Scratch focused on in-depth collaborations with other Scratch members. He found a collaborative team in the Scratch discussion forums and joined its members. First, they made games on Scratch together and later they collaborated beyond the Scratch community, making websites to share their creations. Dean later built on these experiences to design and program his own website, where he advertised his services to build websites for others.

- **Erin**, age 15, initially engaged with Scratch through an in-class activity, but did not become personally engaged and motivated until a year later when she saw her friend make a game with Scratch. Her key moments in Scratch were marked by rich interactions with other members. She recalled that one of her first collaborations was with a Scratch member from another country, which turned into a friendship. One of her most valued moments was when she created an animated series about young people who take a stand against the inequalities in their society. The first

<table>
<thead>
<tr>
<th>Youth</th>
<th>Location</th>
<th>Time on Scratch</th>
<th>First encounter with Scratch</th>
<th>Contributions and other self-initiated activities in Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoebe</td>
<td>USA</td>
<td>1 year, 7 months</td>
<td>Introduced by elementary school teacher</td>
<td>Launched and managed multi-animator project (MAP); Participated as Front Page Curator; Served as Scratch Wiki Editor</td>
</tr>
<tr>
<td>Dean</td>
<td>UK</td>
<td>3 years, 6 months</td>
<td>Introduced by an activity in his IT course in school</td>
<td>Participated in “collabs” (collaborative projects) within Scratch; Developed external websites in collaboration with others he met on Scratch</td>
</tr>
<tr>
<td>Erin</td>
<td>South Africa</td>
<td>2 years, 10 months</td>
<td>Introduced by computer teacher in school</td>
<td>Initiated role playing games; Served as Front Page Curator; Launched a mentoring studio; Participated in “collabs”</td>
</tr>
<tr>
<td>Dani</td>
<td>USA</td>
<td>1 year</td>
<td>Introduced by family friend</td>
<td>Initiated role playing games; Created tutorials to teach others art, design, branding, and coding techniques</td>
</tr>
<tr>
<td>Nathan</td>
<td>UK</td>
<td>2 years, 7 months</td>
<td>Participated in course on e-safety in school</td>
<td>Created studios to organize feedback; Served as Front Page Curator; Served as Scratch Design Studio Curator</td>
</tr>
<tr>
<td>Ryder</td>
<td>USA</td>
<td>7 years, 3 months</td>
<td>Introduced by father</td>
<td>Served as Front Page Curator; Served as Scratch Design Studio Curator</td>
</tr>
<tr>
<td>Regina</td>
<td>USA</td>
<td>7 years, 1 month</td>
<td>Participated in game design course at community center</td>
<td>Helped at a local school; Pursued a college internship to conduct research on Scratch</td>
</tr>
<tr>
<td>Tia</td>
<td>Indonesia</td>
<td>3 years, 3 months</td>
<td>Introduced by a friend and a gathering at a nearby university</td>
<td>Edited and helped manage Scratch Wiki; Translated Scratch into Indonesian language; Served as Front Page Curator; Served on Scratch 2.0 Transition Team</td>
</tr>
</tbody>
</table>

Table I. Study participants
project in the series was well-received by the community, with many members asking if they could help or contribute.

- **Dani**, age 16, was introduced to Scratch by a friend who shared her strong interest in art and drawing. Dani started by creating drawings and simple animations in Scratch, and then figured out how to make interactive games with characters she had designed. A key moment was when one of her projects was featured on the Scratch homepage, and she received hundreds of comments expressing praise and interest in her work. She said this experience motivated her to share more projects and become more engaged in the online community. She also started making tutorials for other community members, sharing her knowledge of art, design and how to code games in Scratch.

- **Nathan**, age 17, started using Scratch at school and was proud to share his projects with the online community. His key moments centered around getting and giving feedback. Because getting feedback allowed him to improve his projects, meet new people and feel more connected to the community, he wanted to share these experiences with other Scratch members. His key moments included serving as Front Page Curator and Scratch Design Studio Curator, roles that enabled him to help other Scratch members get more feedback and recognition for their work.

- **Ryder**, age 18, found out about Scratch when he was nine years old from his father. He started by “just playing around” with it and shared a few projects online. He remembers a key moment was the first time a community member left him a comment inviting him to draw a character for a collaborative animation. He was excited to be asked, and was motivated to become more engaged in the community. He reflected on many memorable moments throughout his participation in Scratch, particularly highlighting ways he had made unique contributions, including organizing and facilitating in-person workshops and a summer camp for children when he was in high school. He also contributed to the development of Scratch itself, starting as a volunteer and then as a summer job during college.

- **Regina**, age 20, first encountered Scratch through a game design course at her local community center. Her key moments included meaningful connections in the online community as well as settings in her real life. After joining the Scratch online community, she started her own “online magazine” inspired by others she had seen in the community. For her online magazine, she reached out to interview other Scratch members and asked them to share their observations about events in the community. In school, she convinced her teacher to let her use Scratch projects rather than PowerPoint presentations for her assignments. Regina continued her interest by majoring in computer science in college, and she took the initiative to pursue a summer research internship at MIT with the Scratch Team.

- **Tia**, age 13, said she was “clueless” at first about how to use Scratch, but learned by experimenting and reading documentation. She was excited that she could make her own games. Most of her memorable moments focused on ways she had contributed and helped others in the Scratch community – including editing and managing the Scratch Wiki (in collaboration with other youth) and becoming a volunteer translator of Scratch into her local language, a role usually fulfilled by adults. She said that learning Scratch helped her “think like a programmer” and made it easy to understand other programming languages, like JavaScript.
We recognize that these deeply engaged participants are a fraction of the members in the 
overall Scratch online community. Participation in the Scratch online community follows 
similar participation patterns of other online communities, where a small population make 
up most of the community activity (Mislove et al., 2007; Johnson et al., 2014). We recruited 
participants until we had gathered a rich set of narrative descriptions of in-depth 
participation and had noted recurring themes across interviews. Rather than aiming for 
population representation, we sought to identify what these youth found valuable in their 
experiences and to identify the features within the environment that supported them 
(Charmaz, 2012). We believe that these key experiences are important considerations for 
coding environments to support broader development.

Data collection
We conducted in-depth, semi-structured interviews that ranged from 60 to 90 min, for a total 
of more than 9 h. While the interviews were 60 to 90 min, the richness of the data comes from 
these young people’s in-depth experience in the community over time. Interviews can 
provide windows into their experience from their perspective and what mattered to them 
(Seidman, 2013). These perspectives can help develop, as Brigid Barron describes, “portraits 
that go some distance toward “recovering the person” in our theorizing about human 
development” (Barron, 2006; Mishler, 1996).

Before we conducted the interview, we asked participants to think of three to 
five moments from their Scratch experience that were memorable to them. During the 
interviews, we asked participants about how they got started, to describe several key 
moments and how these moments influenced them. A similar interview technique has been 
used by youth development researchers for understanding key experiences in afterschool 
programs (Larson and Angus, 2011). In these interviews, we focused on young people’s 
reflections and interpretations of their own experience (also known as phenomenological 
inquiry, Creswell and Poth, 2018). This differs from developmental studies where 
researchers observe young people’s participation over time.

In addition to asking youth about these moments, we also asked them how they got 
started with Scratch and the online community, what they imagined doing in the future and 
how others in their lives such as their family and friends interacted with their interest in 
Scratch. To supplement the interviews, we also examined their online activities such as 
projects, comments and studios. In particular, we focused on the online activities that they 
mentioned in the interview on the Scratch website, such as projects that were featured on the 
homepage or studios that they created for a collaboration. All names used in this paper are 
pseudonyms.

Analysis
We audio-recorded all the interviews and used an external transcription service to transcribe 
the interviews. We reviewed the transcripts with the audio to ensure accuracy. We analyzed 
our data using grounded approaches (Charmaz, 2014) to uncover themes that youth 
identified as memorable and mattering in their experience. Both authors participated in the 
data collection, transcription review and analysis. Through a four-month period, we met 
weekly to discuss our data analysis, ensuring that each analysis was approved by both team 
members.

We developed initial case portraits of each participant that included background 
information, participation in Scratch, and key experiences that youth identified as mattering 
in their trajectories. For additional context, we looked at the projects, comments and studios 
they referenced in their interviews. These case portraits allowed us to see individual
participants’ collection of key experiences over time and explore the common themes across those moments.

We then engaged in an iterative process to analyze these key experiences, the structures that supported these experiences and how these experiences influenced their development. We did an initial phase of open coding focusing on the key moments and experiences that Scratch members identified and the interactions that preceded and followed these moments. We also coded structures or features of Scratch that were involved in these moments, such as receiving comments, creating studios or applying for a role. We organized these key moments and experiences into broader categories such as exploring and collaborating. In the second phase of coding, we reviewed the key moments again and focused on the strategies they enacted within those moments’ interactions (e.g. giving feedback, sharing their project, exploring a new interest), the structures that supported them and how it influenced their identity or how they reflected on their development. We used this phase to refine our coding categories to highlight the strategies and impact of these key experiences such as experimenting to explore interests or collaborating to advance skills. In a final phase of analysis, we converged on key experiences that all youth identified as mattering in the long-term and engaged participation, which we discuss in the next section.

Findings
Our analysis of the interviews led to the identification of key experiences that mattered and why they mattered to youth in their development as leaders within the Scratch community. We first describe early experiences where they experimented and explored with Scratch, which helped them to develop creative confidence and explore possible selves. These early experiences also included getting feedback and recognition, which motivated their continued engagement and sparked their interest helping others through feedback. We then describe experiences where they began to demonstrate leadership in the community by engaging in creative collaborations, taking on roles, leading initiatives and mobilizing community members. Through these experiences, youth were able to learn, to become motivated in their participation, to connect with others and make friends and to broaden their perspective about their interests and the community. In this section, we examine each of these key experiences, why these experiences were important in influencing their trajectory and how these experiences related to youth development principles in supporting broad and long-term engagement.

Experimenting and broadening interests
Each of the youth talked about their participation with Scratch as starting with exploration and experimentation. They described exploring both in the coding environment and in the online community, which led to new skills and new interests.

Gaining creative confidence through experimenting with coding
When asked about how they learned to program with Scratch, most talked about a process of experimentation. For example, Nathan said he figured out how to use the coding blocks in Scratch, “just by experimenting really, and, yeah, just put them together and see what happens. A lot of trial and error.” Through this process, he learned about variables, events, and other coding concepts.

Although exploration and experimentation may sound like an inefficient way to learn, Erin explained how important this process was to her engagement and development:
When I’m in other languages, where you are almost too scared to get something wrong and type the wrong thing and be judged, but Scratch it’s like playing, it’s like chucking things together, if they don’t work, that’s fine. And being able to make mistakes is part of the thing that develops creative confidence, well for me anyway.

She emphasized that the ability to play without being judged as the key to her developing confidence over time. She says, this is “why I’ve mostly stuck with Scratch more than any other programming language because you don’t have to worry about syntax or anything like that, just get in there and play.” Her statement reflects that she noticed and appreciated that Scratch was designed for learning and creating through tinkering, that is, playful experimentation and iteration (Carnegie Council on Adolescent Development. Task Force on Youth Development and Community Programs, 1992; Erikson, 1994).

Broadening interests through exploring the community

In addition to exploring and experimenting in the coding environment, youth talked about exploring other people’s projects and experimenting across interest areas in the online community. For example, Ryder emphasized that what was important to him was “exploring and experimenting with different styles and different areas of Scratch. Getting the most diverse experience on Scratch.” He explained that trying different types of projects gave him an appreciation for what other people have done and what he would like to do.

Phoebe talked about the importance of connecting with people across different interest groups. She found ways to connect with other Scratchers that spanned her different interest areas in her daily life, including music, favorite TV shows and more. She said, “I try to be with a lot of those different people, and try to make friends with a lot of different people from across the website. So you get to really see the talent that everybody has.”

Through the process of exploring on Scratch, they also began to explore and expand their views of their own interests and abilities. Phoebe reflected on how she had changed through her participation in the community:

I’ve gone from the people who don’t really understand Scratch […] to be part of the [groups who are] more artistic, more into coding, or into writing. Just to really take a look at everything. Because when I first started Scratch, I had no interest in art at all. And then I started having all these friends who are like amazing artists. And I thought, “Hey, that looks really cool. I wonder if I could try that.” And so, I started sketching cats, and so now I love doodling.

Similarly, Tia described how meeting friendly people sparked new interest areas:

I found all kinds of friendly people on Scratch and learnt a lot from them without even knowing their real names. I credit this guy for getting me fascinated with cryptography, this girl for giving me a good image of the open source community, […] another girl for motivating me to collaborate with people despite my previous lack of interest.

These early experiences were marked by “consequential transitions,” or moments where youth saw themselves in new ways as they experimented with Scratch and explored new interest areas (Beach, 1999). Scratch members cite the role that other members played in broadening their interests and crediting these relationships in expanding their learning.

These youth’s reflections suggest that they view Scratch as an environment in which they feel safe experimenting and exploring new skills and interest areas. Through the process of experimentation, these youth were learning coding skills, expanding their interest areas and exploring new possible selves. The Scratch environment is designed to encourage individuals to explore and experiment. There are no error messages in the editor. Instead, people learn primarily by seeing the results of their actions and by seeing how others create...
and interact. The community guidelines emphasize the importance of being respectful and friendly, and clearly state that Scratch “welcomes people of all ages, races, ethnicities, religions, abilities, sexual orientations and gender identities” (Scratch Community Guidelines, 2018). Respectful communication and inclusiveness have become norms that experienced participants communicate to newcomers and others, in addition to being modeled and encouraged by staff on the site (Lombana-Bermudez, 2017).

Their experiences resonate with youth development research on the importance of providing environments in which young people feel safe to take creative risks and to explore new areas (Hirsch, 2005; Montgomery et al., 2013). Face-to-face programs support a positive emotional climate by modeling friendly interactions and providing and maintaining clear guidelines for treating others with respect (Smith et al., 2016).

**Motivated by feedback and recognition**
Many of the youth highlighted the importance of receiving feedback on the projects they shared (Figure 3) as a motivating catalyst to improve their projects but also to help others by giving feedback on projects in the community.

**Getting feedback and improving projects**
Even those youth who had participated for years in the community recalled the first time they received feedback and saw this as a memorable moment in their experience in the community.

Nathan described how receiving feedback early on positively influenced him:

**Figure 3.**
Scratch project page

*Note:* Members can interact with a project and leave comments below for the creator.
I decided to share my project and join the community, and I’m glad I did because it was really helpful getting a lot [of] feedback from Scratchers. And, it helped me improve my project and it’s just generally really nice, getting ‘love-its’ and comments from other Scratchers and getting tips on how to improve your projects.

For Nathan, receiving feedback motivated him to continue improving his project and deepening his engagement.

Some youth, such as Regina, actively sought feedback from others:

I definitely sought out the affirmation from other Scratchers. I saw a project recently where in the comments it was like, “I worked really hard on this. Please comment.” I remember looking at that and being like, “Yeah! I wanted people to look at my [project].”

Some youth described receiving broad recognition when projects they had created were featured on the Scratch homepage, a highly visible space on the Scratch website. When Dani created a project about collecting ingredients to make lemonade, she was expecting to receive a few comments. She was amazed to receive more than five thousand messages. Dani shared the experience as an important turning point for her: “I started getting obsessed with Scratch because I realized that people would actually appreciate what I was making.”

Helping others through sharing feedback

For others, receiving broad recognition on the homepage became an opportunity to see the community more broadly. After creating a project that invited people to come up with creative ways to incorporate a “blob” into their project, Phoebe’s project emerged on the homepage as one of the top remixed projects: “I was not expecting that at all, and I got to see this incredible talent. It’s kind of crazy just to see all that creativity from just a circle on a string.”

These experiences of feedback and recognition within the Scratch community were also important in beginning to help others and building relationships, especially when people reciprocated and gave feedback. After receiving helpful feedback, Nathan decided to reciprocate and in the process started building friendships:

I went around on other people’s projects and left feedback on their projects. And some of them returned the favor and left feedback on my projects […] They often appreciate this and they look at your project and you become friends.

An authentic audience can create a motivating context to deepen and sustain participation (Larson, 2007; Magnifico, 2010). For these youth, the authentic and engaged audience in the Scratch community motivated them to improve their projects and make and share new projects. These interactions provided new learning and growth opportunities. Giving feedback to others was also an important experience. Their interest in sharing feedback would continue to motivate their interest in helping others as they later take on roles in the community. Through interacting with an authentic audience, these participants developed their ideas, advanced their programming ability and enriched their communication skills.

Within Scratch, participants often receive positive feedback through simple gestures (e.g. when others click a button to “love” or “favorite” their projects). However, participants particularly value when others take the time to type positive comments on their project pages (e.g. praising, recognizing or give constructive feedback on projects they have created (Monroy-Hernandez et al., 2011)). These participants expressed excitement when one of their projects was featured on the Scratch homepage, as this provided special recognition, widespread visibility, and, as Nathan described, motivation to share these helpful experiences with other Scratch members.
The ability to understand another person’s perspective and communicate ways to build on that perspective has been identified as a key skill that youth can learn from their participation in youth development programs (Hirsch, 2005). Informal learning programs are often structured to provide constructive feedback during the process of creating a project, as well as broader recognition when sharing their work in an exhibition or other community-wide event (Smith et al., 2016).

Creating opportunities for creative collaboration

All the youth we interviewed mentioned collaborative activities as important moments in their Scratch experience to deepen their abilities with coding as well as to learn how to work with others, to build relationships and to more opportunities for collaboration.

Building relationships through invitations to collaborate

Often youth became engaged in collaborative activity through an invitation from another member. After writing a comment on a Scratch project that she liked, Erin received an enthusiastic reply from the creator, Tessa, who then invited Erin to work on a project (“collab”) together. As they worked together, Erin said she learned how to “communicate” with her code so that Tessa could understand and build on top of what she did. Erin also learned new techniques from Tessa:

It’s amazing to see a different person’s way of coding. I don’t know, my brain just did not work in that way. And I learned a lot of animating tricks from them. Yeah, it was a learning experience. And we still talk, we’re friends, which is cool.

In addition to receiving invitations, the youth also looked for opportunities to collaborate. When Dean was browsing a Scratch discussion forum about collaborations, he found a group that called themselves “Forever Corp.” He started by helping them make games in Scratch and eventually helped make external websites that featured the group’s work. Dean learned new techniques from this experience and also became friends with his collaborators. Like Erin, Dean reflected on the collaboration as an opportunity to create new types of projects and to make friends.

Mobilizing members to collaborate

Many of the youth eventually initiated new collaborations, inventing creative activities for other Scratch members to join. For example, Dani created an improvisational storytelling and collaborative world-building activity through role-playing games (RPGs), a popular activity within the website. Dani used a studio in Scratch to invite others to create characters and invent stories through role playing. Phoebe’s Scratch experience was marked with many moments where she created collaborative activities for people in the community. She developed a multi-animator project (MAP) where she invited members to create short animations that she would string together into an animated music video for a Doctor Who song. Her MAP would later inspire other MAPs to emerge from the community. Phoebe also created an initiative in the community called “Happy Scratchaversary.” After celebrating her first full year using Scratch, she wanted to help celebrate others.

I wanted people to be able to celebrate that, because a lot of people, they go kind of unnoticed when that happens, they’re like “Hey, I’ve been on here for three years now, and that’s really cool.” So I made this studio […] People are assigned a different group. They choose a different job. So they can be a gift giver and create a gift for them. When people sign up to be part of the studio, then on their Scratchaversary, it’s a really big deal and it’s a lot of fun.
An important feature of all these collaborative activities was how they were designed and led by Scratch members themselves. To accomplish their collaborative goals, they leveraged different aspects of the website. For example, Erin and Tessa used remixing to keep building on top of a project. Dani and Phoebe used studios (Figure 4) to create spaces for their role-playing and Scratchaversary activities, while Dean and Forever Corp. used the discussion forums. These studios and forums created a shared space for their collaborative endeavors.

Their collaborations with members of the Scratch community provided them with opportunities to learn from one another and to expand their teamwork abilities. As they worked together on projects, youth developed friendships that continued and often led to further collaborations.

Their experiences align with two key principles from research on youth programs:

(1) the importance of peer relationships for supporting young people’s engagement and learning; and

(2) the value of collaborative projects for developing teamwork and other social skills (Dworkin et al., 2003; Strobel et al., 2008).

High-quality youth programs often engage participants in working together on extended projects – for example, collaborating to prepare performances, paint murals or create documentaries (Larson and Angus, 2011). Program staff support the process, but ensure participants are empowered to make decisions and feel agency and ownership over the project (Larson and Angus, 2011).
Taking on and creating new leadership roles
For many of these youth, a catalyst that deepened their participation in Scratch was the experience of taking on roles in the online community as well as creating roles for others to help and participate.

Expanding perspective by taking on community roles
Several youth described their experience taking on the role of Front Page Curator, which involves choosing other people’s projects to be featured on the homepage. To apply for this role, any community member can make a project expressing why they want this role and how they plan to approach it. The Scratch moderation team at MIT reviews the applications and selects one Scratch community member to serve as the Front Page Curator for two weeks.

Each of the youth recalled the amount of effort they put into their applications – working for “hours and hours” and often weeks at a time. They also expressed their excitement and surprise in being selected to take on the role. When asked why she wanted the role so much, Phoebe explained that she was seeking to influence the community, as well as to gain experience and have fun. When asked how she wanted to influence the community, Phoebe explained:

I wanted to show them not just to look at their groups […] Don’t just look at those projects. Take a look at a wide variety of things. So I curated some newer Scratchers’ projects that were totally unnotice because my guidelines were a lot more strict than most of them are […] I wanted them to be pretty much undiscovered, so there’s that discovered talent. Just when I saw those people who learned that they were going to get curated and then they did and they got Top Loved, their reactions were just amazing.

Others also talked about their specific guidelines for selecting projects to curate, and ensuring that projects that showed a lot of effort but had received less attention were featured. Nathan explained, “Giving Scratchers more attention. It’s something you know that I took a lot of pride in and I spent a lot of time looking for projects that was useful and for Scratchers that really deserved them.” He and others saw featuring others’ projects as a way to give back to others.

Creating new ways to contribute to the community
In addition to taking on existing roles, some youth also created their own roles and other ways to contribute to the community. For example, Phoebe defined several types of jobs in the Scratchaversary studio and recruited other community members to fill those jobs. These jobs included “advertisers” to promote the studio, “gift givers” to create projects for others and “monitors” to curate the studio. In her interview, she reflected on what she had learned from experience of designing and managing roles, such as how to shape the role so that another person would be interested and able to carry out the role successfully. She also talked about the benefits of providing roles – including getting other people more engaged in the studio and keeping the initiative running.

When asked how taking on and creating roles affected them, the youth talked about developing a deeper connection and broader perspective on the community. Erin said:

I felt so much more part of the community […] I met so many more people and seen many more amazing things on Scratch and I was just blown away […] It just amplifies this same journey of meeting more people, collaborating more, and seeing all this awesome stuff and getting inspired.
Ryder suggested that ideally everyone on Scratch would eventually take on a role and contribute to the community. He said:

I really like the idea that there are different roles that you can take, so you’re not just producing and consuming just for you or whatever – you’re also helping other people.

Tia contributed by becoming a language translator and was promoted to become an administrator on the Scratch Translation Server, a role typically carried out by adult volunteers. She said: “I interpreted the ‘promotion’ as some sort of endorsement, and it really motivated me to give my best – to prove that I can act all professional even though I’m just a kid.” She was motivated to do research and ask others in her local community to decide on how to translate unfamiliar phrases. She shared how she discovered an interest in translation and linguistics through carrying out this role.

For these youth, the opportunity to contribute to the community was a key experience that deepened their participation. Youth were motivated to carry out their roles. Taking on roles with responsibilities (such as serving as a Front Page Curator) broadened their perspective of the community, prompting them to think about how they could help others with less experience or fewer connections. Carrying out their roles also shifted their perspective on their own abilities, helping them see themselves as capable of making a difference.

Their descriptions highlighted two different ways that Scratch enabled them to contribute to the community:

(1) by offering defined roles; and

(2) by providing open-ended structures that youth can use to develop their own roles.

For example, the role of Front Page Curator offered these youth a visible role. The application process prompted the youth to reflect on the types of contributions they could make in the community. Being selected for the role made the youth feel excited and gave them a sense of responsibility. In addition, the open structure of studios allowed the youth to take the initiative to organize others to improve the community.

The observations that youth made on what they learned from taking on roles aligns with findings by researchers studying the impact of roles in after-school programs. In an in-depth study of high-quality youth programs, Salusky et al. (2014) found that youth who carried out roles voluntarily and successfully became more diligent, organized, and aware of others – and began to see themselves as interested and capable of taking on new roles and responsibilities in the future. Similarly, taking on roles in online environments can broaden young people’s perspectives on others in their community and expand their vision of their own future pathways.

**Discussion**

*Broadening visions of coding to learn*

In this study, we looked to youth perspectives to highlight what mattered in their experiences, which went beyond learning to code and instead focused on broader experiences. The youth in our study highlighted key experiences that sustained their engagement, influenced their development of broader skills and emergent leadership. These experiences included key moments of exploration, recognition, collaboration and contribution. In all these experiences, youth pursued their own goals and shaped their trajectories, which reflects theory from youth development research that youth are “agents of their own development” (Eccles and Gootman, 2002). As they actively and deeply engaged in this community, the youth developed skills that
included computer programming, building on others’ ideas, working with others and carrying out responsibilities.

Interestingly, these youth had broad visions for what they can gain from learning to code. They saw learning to code as providing opportunities to explore interest areas, to make new friends, to collaborate on projects and to contribute and help others learn. When they began to explore other programming languages, they continued to seek out these kinds of broader opportunities, in addition to developing their coding abilities. Similarly, some adults advocating for children to learn coding and computer science recognize a variety of benefits in life beyond coding (Vogel et al., 2017), ranging from career skills and civic engagement to fun, fulfillment and personal agency. These youth perspectives on coding spanned these areas. Importantly, their trajectories often started with fun and enjoying the process of exploring, and progressed to gaining skills to express themselves, connecting with others and eventually seeing ways that they could apply these skills to help others and pursue their goals.

Supporting broader development in coding environments

When looking at outcomes of coding initiatives, researchers and educators often focus on evaluating individuals. However, what about evaluating how well learning environments provide support for broader development? As we have shown in this paper, research on youth development offers principles and practices for designing learning environments that support young people’s active participation, contribution and development. For example, studies of youth development in afterschool programs highlight the importance of creating a safe and welcoming environment free from judgment to encourage newcomers to explore and experiment to meaningfully engage in a community (Strobel et al., 2008). To these youth, it was important to engage in a welcoming and nonjudgmental environment in Scratch. In addition, our participants’ experiences suggest that it was valuable to have resources to explore beyond computational concepts, such as other people’s projects, interests and activities. Youth participants were able to build their confidence as they experimented with computational ideas, but they were also able to expand their interest areas and explore possible selves as they interacted with other people’s creations.

As in previous research on youth development, our analysis identified the importance of providing youth opportunities to collaborate on projects and to learn from others’ feedback and perspectives (Dworkin et al., 2003). In addition, our findings highlight the value of providing flexible structures for youth to initiate their own collaborative activities. Our participants leveraged the studios and remixing features of Scratch to curate projects, create role-playing games, mobilize multi-animator projects and coordinate co-constructed projects.

These positive experiences that encouraged their ongoing participation also developed their interest in supporting other community members’ learning and engagement. Youth development researchers have found that offering youth roles where they can make decisions and contribute to their community enables them to develop responsibility and leadership skills (Salusky et al., 2014). Our analysis of their experiences showed how taking on roles helped them to expand their perspectives on what they were able to do and how they could help others. In addition, participants were able to invent roles for themselves and their peers. Through these experiences, they developed as emergent leaders, inspired to pay it forward to support others developing in this coding community.

We recognize the need to further investigate whether the experiences of the youth we selected connect to the experiences of other youth. While many of the participants started without any prior knowledge or interest in coding, the youth we interviewed had become
invested in Scratch and enjoyed spending time in the Scratch community. We recognize that these youth had supports and resources within their homes, schools and/or communities that enabled their long-term engagement with Scratch. For example, many of the youth we interviewed had their own computer and some had family members with technology-related education and careers. Despite these limitations, we suggest that it is valuable to highlight the types of experiences that these youth valued during their long-term engagement over the full trajectory of participation – starting from their initial explorations to becoming valued contributors who provide new opportunities and support for others in the community.

Conclusion

In an essay describing ten recommendations for the Computer Science for All movement, Margolis and Goode (2016) suggest that computer science is part of a “well-rounded education.” Computer science education is thus positioned within an educational landscape that includes areas such as sciences and humanities, which can be complementary rather than in opposition to each other. Similarly, we argue that learning opportunities with coding can support an individual’s “well-rounded development.” The opportunities in learning to code should not be limited to computational thinking or problem-solving skills, but rather designed to provide broader experiences that support the development of collaborative, leadership and socioemotional skills.

What qualities and supports should broader learning opportunities with coding provide? We want to encourage educators, designers and researchers working in the coding movement to pull from literature on youth development within afterschool programs to understand what these learning environments can look like. Our study contributes to ongoing work that highlights how online spaces can be sites for youth development (Ito et al., 2009; Martin and Steinkuehler, 2010; Kafai and Burke, 2016). We want to emphasize that activities like coding, which at the surface might seem to be mostly about cognitive development, can also promote social, emotional and identity development when embedded within a supportive peer community, whether in-person or online (Barron et al., 2014).

Youth perspectives can also highlight what matters, especially as researchers, educators and other decision-makers deliberate on what counts when learning to code and how to design learning environments to support youth. As the coding movement continues to spread, there are opportunities for coding environments to not only focus on teaching specific coding skills, but also to support youth in their development of broader skills that enable them to take initiative, to help others, and to lead efforts for others to learn. We want all young people to feel the way Erin described when she reflected on what she gained from her experience:

It’s creative confidence. Feeling empowered, not just a consumer who plays games on the computer [. . .] Being able to say, I have good ideas, I can make them and I have the power to do that and I can get things wrong and I can make mistakes, but I will get there if I just keep going. And that to me is creative confidence. That’s the kind of people that Scratch is growing and I honestly think Scratchers are gonna change the world, seriously.

References


Ito, M., Baumer, S., Bittanti, M., Boyd, D., Cody, R. and Herring, B. (2009), *Hanging out, Messing around, Geeking out: Living and Learning with New Media*, MIT Press, Cambridge, MA.


Further reading


Corresponding author

Ricarose Roque can be contacted at: ricarose@colorado.edu
“I like computers. I hate coding”: a portrait of two teens’ experiences

Paulina Haduong

Harvard Graduate School of Education, Cambridge, Massachusetts, USA

Abstract

Purpose – Some empirical evidence suggests that historically marginalized young people may enter introductory programming experiences with skepticism or reluctance, because of negative perceptions of the computing field. This paper aims to explore how learner identity and motivation can affect their experiences in an introductory computer science (CS) experience, particularly for young people who have some prior experience with computing. In this program, learners were asked to develop digital media artifacts about civic issues using Scratch, a block-based programming language.

Design/methodology/approach – Through participant observation as a teacher and designer of the course, artifact analysis of student-generated computer programs and design journals, as well as with two follow-up 1-h interviews, the author used the qualitative method of portraiture to examine how two reluctant learners experienced a six-week introductory CS program.

Findings – These learners’ experiences illuminate the ways in which identity, community and competence can play a role in supporting learner motivation in CS education experiences.

Research limitations/implications – As more students have multiple introductory computing encounters, educators need to take into account not only their perceptions of the computing field more broadly but also specific prior encounters with programming. Because of the chosen research approach, the research results may lack generalizability. Researchers are encouraged to explore other contexts and examples further.

Practical implications – This portrait highlights the need for researchers and educators to take into account student motivation in the design of learning environments.

Originality/value – This portrait offers a novel examination of novice programmer experiences through the choice in method, as well as new examples of how learner identity can affect student motivation.

Keywords Computer science, Portraiture, Scratch, Women in STEM

Paper type Research paper

Introduction

For many young people, particularly from historically marginalized populations, evidence persists of both a civic empowerment gap (Levinson, 2012) and a digital divide (Ito et al., 2013). Much of the intriguing youth participatory action research, aimed at addressing the civic empowerment gap, exists successfully in informal education and can be in tension with formal contexts (Rubin et al., 2017), but informal computing education has been demonstrated to be even more biased in favor of wealthier and male students (Guzdial, 2015). Although the history of computer science (CS) education in US K-12 is fairly extensive, in the past decade, coding has experienced a resurgence. However, learning to code has often been framed around vocational opportunities (Blikstein, 2018). Although youth are often self-motivated in informal contexts, marginalized youth in formal
classrooms may be initially reluctant or skeptical about learning to code, but because our society is highly reliant on technology, computational fluency is becoming essential for civic participation. Through the qualitative method of portraiture, this article focuses on the experiences of two reluctant learners in a six-week summer program aimed at developing students’ civic capacity as well as computational fluency.

350

Background

Traditional CS education has largely focused on teaching students conceptual knowledge (mastering variables, loops and more) (Soloway and Spohrer, 1989). Because this concept mastery is challenging and introductory CS course failure rates are high worldwide (Watson and Li, 2014), researchers have been preoccupied with questions of how to more effectively teach these computational concepts. Lack of success in CS has been tied not only to challenging content but also to broader issues of diversity and inequity, and there exists a growing body of research empirically examining these systemic challenges, in both formal and informal contexts (Vakil, 2018; Margolis et al., 2015; Goode, 2008; Cheryan et al., 2015).

However, a focus on computational concepts ignores what young people might do with their mastery. In a digitally networked society, there is a need to ask what students might express, build and communicate about their experiences and desires to shape the world, particularly as the links between technology and power become increasingly stronger. If CS is framed with more attention to equity (Vakil, 2018), then educators and learners are asked to also attend to issues of power, ethics and "technological solutionism" (Morozov, 2013).

Critical computational literacy: 

[...] provides a way to create and theorize conditions for the potent learning that can take place at the intersection of engineering and computational thinking on the one hand, and narrative production and critical pedagogy on the other (Lee and Soep, 2016, p. 481).

Like many other forms of youth-driven research and action, this kind of work is often long-term and intensive, existing in informal contexts because of the constraints of K-12 formal education. But participatory action work can be transformative for how young people understand their own capacity to be change actors, and the challenges of authentic implementation are worth pursuing, because:

[...] students are required to be in school; while imperfect, classrooms are places where students who might not otherwise have the opportunity can participate in a learning experience which foregrounds their experiences and perspectives and values their agency (Rubin et al., 2017, p. 190).

This question of agency underscores the important role of intrinsic motivation in learning. From self-determination theory and a growing body of motivational research, it is clear that the prevalence of competence, autonomy and relatedness supports can help explicate how and when learners succeed (Deci et al., 1991; Ryan and Deci, 2000). When learners have high intrinsic motivation, their higher interest and confidence can manifest in persistence and creativity (Ryan and Deci, 2000). Though critical computational literacy can be transformative and could address both the civic education and the CS education gaps, if learner motivation is low from the beginning (which it may well be for historically marginalized youth), it may be challenging for learners to realize their potential. And while there are many challenges to supporting learner agency in formal schooling (e.g. state standards, assessments and lack of funding), the connected learning framework, as well as a large body of related literature, points to informal learning contexts – both online (e.g. fan
communities) as well as offline (e.g. libraries and afterschool programs) – as places of opportunity where young people are already pursuing their interests, connecting their own identities to their communities and thriving (Martin, 2015).

This is the starting place from which I sought to examine learner motivation within the context of a six-week program on critical computational literacy. This program was facilitated in July–August 2017 for 17 high school students (ages 14-17), diverse across age, race, socioeconomic status and prior CS background. Most learning activities were developed in close collaboration and co-facilitated with Raquel Jimenez, a fellow doctoral student. These activities were designed with constructionist learning aspirations in mind, creating space for young people to create, share, reflect and personalize through both on- and offline media (Brennan, 2013). Through the lens of portraiture, a qualitative method, the author explores the following research question:

**RQ1.** How do Carolina and Lange[1], two high school participants, experience learning in introductory CS contexts?

Though this program was an informal opportunity, the project happened through the city’s Mayor’s Summer Youth Employment Program (or Mayor’s Program), one of many similar citywide initiatives across the state. In many cities, over a thousand young people are paid each summer for 20 hours of work each week at dozens of work sites across the city. Most informal programming is voluntary, but while high schoolers applied to the Mayor’s Program, they could have been assigned anything from gardening outdoors to working with the fire department, or even attending summer school full-time. And while most of the program’s students attended the local public school, their backgrounds vary.

To investigate this research question, I used portraiture for the method for this study. Portraiture draws from traditional qualitative sources of data collection, and in addition to analyzing artifacts produced during the program (students’ design journals, computer programs and group reflections), individual hour-long exit interviews with all students were conducted, as well as follow-up hour-long interviews with Carolina and Lange. Unlike other qualitative methods, portraiture asks the researcher to search for goodness, which is not a search for an “idealized portrayal of the human experience,” but rather a search “for the complex and competing truths that combine to shape an authentic narrative” (Lawrence-Lightfoot and Davis, 1997, p. 146). And because the standard of portraiture is authenticity, not validity (Lawrence-Lightfoot and Davis, 1997), the portraitist must, through both data collection and subsequent analysis, seek to co-construct knowledge with their participants, instead of placing them as the objects of study (Xiang, 2018). The aim of co-construction asks the researcher to pay particular attention to both the context of the narrative and the position of the portraitist (Lawrence-Lightfoot and Davis, 1997), for “a reader who knows where the portraitist is coming from can more comfortably enter the piece, scrutinize the data, and form independent interpretations” (p.96).

To support the portrait’s authenticity, throughout both the process of data collection, as well as my analysis, I actively looked for the “deviant voices” (Lawrence-Lightfoot and Davis, 1997, p. 85) which challenged not only my own observations of students’ actions, my presuppositions of “where learning happens,” but also what participants appeared to say; I also invited the two main participants, as well as my co-teacher, to read and give feedback on an early draft of the manuscript. In this portrait, central themes of identity, community and relatedness and competence are explored through Carolina and Lange’s participation in an informal learning context, as well as their reflections on their prior CS experiences in formal schooling. Portraiture seeks to find “the universal in the
particular” (Lawrence-Lightfoot and Davis, 1997, p. 80), and in the discussion, implications of these learners’ experiences and aspirations for educators are explored.

A portrait of Carolina and Lange

Setting expectations

When I applied for the Mayor’s Program, I said I specifically didn’t want to do STEM […] and they gave me STEM! So, at the beginning, I was like, oh my gosh, this is so unfair. I should just, like […] ugh, this is going to be so horrible. I’m going to be waiting until the last minute for it to end.

Fourteen-year-old Carolina sits across from me at a light-yellow table in a small first floor study room of the university’s library on a sunny August morning, telling me at length about her lack of initial interest in the “Mayor’s Program,” which assigned Carolina to my summer project. Carolina is skinny and dark-eyed, and she swings rapidly back and forth in her chair, her ponytail moving a beat after the rest of her body.

“What don’t you like about STEM?” I ask, when I manage to edge a word in. It is just the two of us in this small room, my phone-as-recorder between us and my laptop off to the side, displaying some of the digital artifacts she brought to this interview: her design journal, some photos she took in the city and a project she built in Scratch, a programming language built for young people.

Carolina rolls her eyes. “I don’t know what I was against. I was just, like, I had a horrible science teacher back in eighth grade […]”. She pauses, her eyes darting around, then points at the recorder, mouthing words to me.

I laugh as I figure out what she is trying to say. “Yes, you can talk about your teachers.” We went over the guidelines about anonymity and confidentiality before we started the interview, but I remember that, as a child, talking about teachers always felt precarious, particularly to other teachers and adults.

A look of relief passes over her face, and she settles into her chair, telling me that this particular teacher was a “bad listener,” disorganized and, perhaps, unprepared: “She just didn’t know her stuff.” She specifically said, almost every class, “Guys, I’m not a scientist, so bear with me, you know.” And I was like, I mean, what do you mean, “bear with me?” Her voice rises at the end, indignant.

In another room upstairs, Lange tells Raquel (my co-teacher/researcher) that his expectations of the Mayor’s Program had also been low, though for different reasons. This was Carolina’s first summer job, and she entered with expectations drawn from her classroom experiences. But Lange had worked before, both in the Mayor’s Program and elsewhere. “Goals for the Mayor’s Program were for me to work and to get money […] that was my mindset coming in”. Tall, with a long face and thick, close-cropped hair, Lange is a lanky mixed-race 17-year-old. In a plain shirt and black shorts, wearing thick athletic socks and Adidas sandals (the classic black and white slides), he looks like any other jock; on the street; you might pass him without another glance. In groups, he speaks softly, and often last, with a sardonic self-deprecating style. When asked about his expectations around computers and other science, technology, engineering and math (STEM)-related work, Lange is upfront with his dislike of computer programming, although he wants to study animation in college: “I like computers. I don’t like coding. I think I liked the idea of being able to create something, but I don’t think coding was the thing for me”.

I first met Carolina and Lange through a six-week work-site/summer program for the Mayor’s Program. It is the past week, and Raquel and I are conducting exit interviews. In speaking with Carolina and Lange, I am particularly interested in learning more about their experiences with CS in schools, as well as whether and how their time in this program has affected their attitudes about coding at all.
Learning to code, through Hour of Code and About Me projects

I was a sophomore in college when I took my first programming class, convinced that I wanted to be a computational linguist. Within the first week, we had gone from typing out simple commands such as print “hello world” and set a = 0 to attempting to write hundreds of lines of code, learning a new language by attempting to learn syntax and semantics and write and read it, all at once, in only a few short weeks.

I passed by the skin of my teeth, heavily reliant on the goodwill of newfound friends and terrified of failure (Figure 1). So terrified was I, in fact, that I dropped out of CS. Years later, discovering Scratch, a block-based programming language developed by the Lifelong Kindergarten research group at the MIT Media Lab and designed for young people, was like a breath of fresh air. In Scratch, which has been designed with a “low floor” to entry and “wide walls” of possibility (Resnick et al., 2009, p. 63), the creator moves the blocks around, and they snap together easily. You do not have to memorize words and structures; you can tinker with a set of blocks to create a computer program with ease. By moving the blocks around, you can make your “sprites,” or characters, do things quickly and easily. Although high schoolers are a little older than the target group for Scratch, and I am nervous about this group regarding the colorful Scratch blocks as childish or “not real programming” (Powers et al., 2007, p. 217), we have decided to use Scratch as the creative medium of choice for the duration of our program.

Before this summer, the first and only time Carolina was introduced to computer programming was through Hour of Code,[2] a national initiative with ambitious aims to bring CS to every school. From sixth through eighth grade, once a year, she would be asked to spend on hour on the computer with all of her classmates, learning Scratch:

It was not fun. I dreaded it every year, because it was a set project that I had to figure out, and it was super boring. Like, once you got it, it was just like, oh, glide two spaces, next slide, glide four spaces, next slide […] that sucked.

As she talks, she leans toward me across the table, and then gesticulates wildly with her arms. “Hour of code […] it was, like, Moana-based stuff, or Frozen. We’re in middle school, you know. Could we, like, figure it out ourselves?” From her dramatic eye rolls, I gather that Disney-themed CS “paint-by-number” projects are perhaps better suited for elementary school students.

Carolina’s experiences were not uncommon; though I have heard teachers (and, more often, district leaders) speak enthusiastically about Hour of Code, other students, including many within the program, have complained about the step-by-step nature of the projects. Lange, in an interview with Raquel, tells her that he thought Hour of Code:

[…] was fun. I thought it was nice to do it. It was confusing, but I enjoyed seeing how typing certain things and watching them run and work; it was really interesting.

His voice is unenthusiastic, though his words betraying a mixed bag of emotions. In his speech, he often makes one assertive claim, and then backtracks with the following utterance, until you are left wondering if he is ambivalent, thoughtful and/or perhaps disinterested. But
perhaps in this instance, he is simply forgetful; although Carolina’s memory of this classroom
experience is fresh, Lange is recalling something that happened almost four years ago. For
both of them, however, this was the only time they tried CS in the classroom.

In the first week, we decide to introduce students to computer programming through the
making of an “About Me” project, both to get a sense of where our students are at, as well as to
learn more about each other. Though a couple of the older boys come in as friends, nearly
everyone else seems to be only vaguely familiar with one another. Everything this summer
takes place in the Lauchpad, a university room designed to “launch” ideas. Consequently, the
room is filled with all of the accoutrements of a design thinking space: shiny whiteboard walls,
white tables and bright red chairs, with glossy Mac desktops lining one wall. Everything is on
wheels, meant to convey the underlying idea that to create and disrupt, designers must be able
to reconfigure the room. The room attempts to make a statement that it’s a “cool” space, but the
harsh fluorescent light and drab carpeting, as well as the lack of windows, is reminiscent of
1970s offices, particularly because of the beige walls. In refashioning this space, the room only
barely escapes its cubicle past. And because the computers line the wall opposite the
whiteboards, with low tables and chairs dotting the space in between, the room feels less novel
and more like a well-furnished K-12 classroom (Figure 2).

During the year, the space is lightly used by student groups looking for meeting space. Over
the summer, we flood the space with hungry and loud teenagers, piling donuts and
orange juice and snacks in one corner and blasting music while they work. Lange and
Carolina are noticeable in their differences and similarities. In the mornings, Carolina is
either extremely early or a hair late, depending on whether she went to her second job
beforehand (as an apprentice dessert chef in a famous local restaurant, between the hours of
4 to 7 a.m.), coffee clutched in her hands, her world-weary adult expression at odds with her
teenage movements. She walks over to the suitcase of 4-year-old Chromebooks we have
borrowed from another lab, pulls one out and chooses to sit at one of the small tables closer
to where we sit. Lange generally ambles in a few minutes late, oversized black headphones
covering his ears, giving me only a nod in greeting before he pulls out a chair for a Mac and
slides into it, turning his back. We have let them sit wherever they want, and Carolina
chooses to be close to us, at a communal workspace, whereas Lange sits by the far wall.
As the summer progresses, they move around the room, but I often find Carolina in the thick
of things, whereas Lange talks to one or two people on the sidelines.

When they work on their About Me projects, they stick to their usual spaces – Carolina at a
table with others she has hardly met but chatting with nonetheless, and Lange in the far, far
corner, working in silence. Though neither of them has really spent any time in Scratch,
Carolina has clear ideas about who she is and how other people see her. Her “About Me” project
is an interactive acrostic poem, inviting the user to click on the letters of her name for more
information about how much she loves to organize; her love for soccer, baking and her desire to
smile more: “I try my best to be happy most of the time because […] why not? It’s better.” Even
though her project is unfinished, it sparkles with her personality – colorful backdrops of dark
purple behind bright orange letters. Lips pursed in concentration, she works hard on the visual
design of her project. I know what she is stuck on, because she is not afraid to ask for help,
either from other students at her table (most of whom are equally lost, save for the three 17-
year-old boys who previously took CS classes at the high school), or from Raquel and me. Like
the science teacher she complains about, Raquel and I are not computer scientists and do not
have all the answers (just many questions) or content knowledge, but she seems to have no
difficulty raising her hand at every obstacle. Once I show her the step-by-step tutorials within
the program, she jumps into them easily and independently.

Lange gives no indication of whether he is struggling, and it is only when I walk over to
him and get his attention that he pulls off his oversized headphones to look up (Figure 3).
“Do you need any help?” I ask, and he lifts one rounded shoulder high and drops it down,
glancing up at me briefly and then back at his screen. It is still only the first week, and I am
just getting to know these students, but where Carolina’s excitement, boredom and other
emotions flash starkly on her face, Lange is inscrutable. Is he bored? Interested? When I ask
what he’s working on, he shrugs again. As he is not off-task, I am not too worried, and
perhaps it is because we are still strangers to one another, or perhaps I have offered help at
the wrong time in his project process. In the exit interview, he says:

I didn’t know how to present myself really […] but I think that having a silly animation with silly
text and silly jokes inside of it that are kind of humanizing is a very good way to introduce myself.

At the end of the first week, we sit in our daily closing circle. Though we ask for everyone to
sit on the floor, some students refuse, looking at the brown carpet with disgust and opting to
perch on the red chairs, using their feet to push the chairs back and forth. Carolina sprawls,
legs long and Lange hugs one knee to his chest, his shoes off to one side. He is the only one who has decided to go shoeless, and some of the other students are looking at him askance, but he does not notice. Once everyone is settled, we pass out small white cards and explain that they are meant to be reflection cards, for students to reflect on the past week and also let us know anything they would like to change. Although some students focus on the content, writing things like “I’m disappointed we’re using Scratch” or “Scratch was really hard at first and I hated it but I’ve gotten better,” Carolina and Lange focus on other aspects of the program. Based on their faces and attention when we work on Scratch projects, they are doing what we ask of them, perhaps because we are still authority figures in a school-like setting, but they are not exactly the most enthused about it.

Carolina writes, “I really like [the program]. I love the freedom of being able to leave the building and being allowed to go on our phones.” Her scrawled message fills up the entire index card, black ink on white paper and her name squished into the bottom right corner. Her iPhone, carried at all times in a tiny phone-sized orange cross-body bag, is indeed something she reaches for constantly, though she is certainly not alone among our students, Lange, whose music tastes are eclectic and wide-ranging, hears the playlist that Raquel and I created and writes back in a reflection card, “Going to go on a whim here and say that the music is funky, but not in the greatest way.” Later, he tells us about his vast music collection and contributes songs from Michael Jackson and David Bowie, as well as the Sonic the Hedgehog (video game) SEGA soundtrack to the group playlist, though he continues to work on his projects with his large over-ear headphones on his head, listening to his own music even as the music he has chosen for the group plays in the background, ABBA’s “Dancing Queen” floating from the speakers, whereas everyone crouches over their own computers.

**Drawing inspiration and advice from community**

But students do not only work on projects by themselves; in the next week, we assign students to group projects, hoping that they will get to know each other better and also learn to build on each other for ideas more often. Carolina’s first group project focuses on creating an interactive story about mental health support for LGBTQ teens; when they work on a storyline for it, I watch as she does not necessarily lead (another young woman takes charge of pen and paper, whereas a young man dictates a storyline), but she interjects opinions and commentary, adding her voice as color and texture in the narrative. The group, though not without the occasional distraction of their phones and the latest, hottest mobile games, mostly functions cohesively, finding one idea and sticking with it. Because one of the students in the group is more advanced, it is easy for him to take the reins when it comes to Scratch. We had hoped that by grouping students with mixed abilities, they might learn from each other, but instead, I watch as 17-year-old Henri hunkers down over his brought-from-home Macbook (other students are using our borrowed and battered Chromebooks), looking every inch the stereotypical male coder, whereas Carolina and the others hover over a piece of paper, storyboarding ideas for what their game will look like. Even though it seems to me like Carolina is not learning much about coding through hands-on experience, she tells me later that this time both helped her learn more about Henri’s expertise and get to know him better:

> Henri is really good at coding. And I sat next to him and we […] Alaska and I were, like, stuck for a good twenty minutes on this, and we were like, “Okay, Henri, please help us!” And he came over and he immediately had the answer. He didn’t just tell us, he, like, helped us figure it out. And that was really helpful. I don’t think we could’ve finished it, if he hadn’t done it.

The freedom to ask for help is not always available during the school year, she tells me during the exit interview. Rolling her eyes and groaning, she talks about “silent
workshop,” an hour-long class period that various subject teachers often impose during the school year:

We couldn’t talk to anybody. And it was just, like, if you were stuck, you were stuck. There’s nothing you can do about it […] you couldn’t even go to the teacher […] you were not allowed to do anything […] You just sit there the entire class, like, “OK, I can’t talk. I’m not allowed to Google anything. I can’t do anything”.

This restriction of her physical freedom weighs heavily on her. Her clothing (loose, colorful t-shirts, cropped athletic leggings and comfortable slip-on shoes) reflects her need to move, around rooms and buildings and – in the afternoons – soccer fields.

Lange’s group struggles, he admits later, to find a cohesive idea, and ultimately, their group makes a “city design” game where the player makes decisions about how to design their city in preparation for climate change. The game was completed, but no one asserts a clear vision, either for the group process or for the project, and because of a quirk of fate, Lange is the only one from his group present on the day the final presentations happen.

On that day, three weeks into the summer, we fill the Launchpad with our friends, partners, advisers and pretty much anyone else we can think of who is spending the summer in the area. I am excited for our young people to have the opportunity to present their work, and I am also nervous for them. Are they prepared? Are they proud of what they have made?

Although other groups of three to four present together, he stands alone at the front, alternating between slouching and putting his hands in his pockets or leaning on a nearby table, both presenting what his group did and following up with a light, apologetic comment about what does not get done, talking quickly in a jocular manner like a young standup comedian, one eyebrow quirked. Aren’t you in on this joke?, his eyebrow seems to say, and I wonder if that is the message our audience members receive. At times, his comments are literal and chronological, saying, “And then we did this […] and we decided to that […]”, but sometimes he follows up with, “Well, I don’t know why we did that,” or “This isn’t as complete as some of the other wonderful projects […]”.

As I watch him, I am reminded of my evenings reading his entries in his online design journal (part of our daily reflective practice). His writing is fashioned in a similar style – he writes one large block of text, talking about what we did that day, and often follows up with one or two side-comments. Instead of footnotes, these comments are in tiny, barely legible (six point) font, right at the ends of paragraphs. It is as if he carefully considers a long assertion and then remembers an additional point, tacked on right at the end. Capital letters are never used in his writing, lending the text an additional feel of quiet musing.

In his exit interview, he says:

I think that was probably the most enclosed project that we did, ‘cause I didn’t feel like we had very many options to get the idea out there, and I felt like if we were a little more free we would’ve had more inspired projects.

Building competence in community
In an effort to create more space for creativity and inspiration, we let students decide whether they would like to work alone or in groups for the past couple weeks, as well as letting them choose what they want to work on. Carolina writes, “I love my group not just because I enjoy working in groups that work but because I have developed friendships.” Her group is boisterous and loud, their laughter echoing across the room, with one or more of them constantly leaping out of their chairs and pacing around the table or drawing elaborate designs on the whiteboard. Often, I walk over there to check that they are actually working
and on-task, instead of just chatting (over the course of the summer, they have become preoccupied with a game where you as the player move up and down the screen to avoid sharks swimming toward you, and frequently I catch students playing that instead of working on their own games).

One person in the group, Vic, is sometimes louder and more opinionated, and I worry sometimes that Carolina and some of the others may not be making themselves heard, or that they have been assigned to work on things that are less exciting, but consistently Carolina writes and talks about how much she loves her groups. For Carolina, much of what she loves about the program is wrapped up in the connections she’s made. She tells me that “it’s really perfect […] it has amazing people.” In general, she seems to get along with everyone, though she definitely gravitates toward certain people, she holds true to her attempt to smile regularly and keep up a buoyant attitude.

Behind Carolina’s group, which insists on cramming five people around one small table and whiteboard and crowding into one corner, Lange sits with his groupmates Christopher and Joshua. Carolina’s group sits in a circle, talking across the table, no one clearly leading the group but everyone contributing ideas quickly back and forth, but Lange, Christopher and Joshua sit silently in one line at the back of the room, each boy at his own Mac. I can see their screens, so I know when they are hard at work, but unless they have developed telepathic abilities, it is not clear how they are communicating with each other. Christopher and Joshua, also 17, but highly skilled in CS, are best friends with an idea of what they want to make, a project to raise awareness about the diversity issues in CS. From their lack of interactions and reticence in talking about the project to me, I wonder whether Lange, the only nonwhite member of the group, has a role and an interest in this project. His headphones are in, like always, and he toggles quickly between his Spotify playlist and the online Scratch editor, leaning back in his chair, his phone half-hanging out of his pocket.

What he does not say out loud, though, he writes in his design journal, which is semi-private – all of the journals are Google Slides in a single folder, so although anyone in the program can access them, rarely does any student look into a journal that isn’t their own. In his, he writes:

[...]

my group decided to talk about a lack of representation of female/black/latino students in the computer science field, something that doesn’t affect me personally (considering that i’m not focused on cs) but has an effect on some people i know, more specifically my mom and my grandpa. they’re both black coders, my grandpa working to code for the navy and my mother working to code for various companies as a checker/tester for how their code will work. if we get this type of info out, I hope it goes out to let other cs minorities know that it is known that they aren’t alone.

His interests and work are connected not only to his own identity but also to his family, as well as to the others in his group.

In his commitment to his group, and in how he talks with others, he relies heavily on others:

I think it’s better to explore the people around you than to know what you can do because if you know what people around you can do to help you then there’s not really that [...] you’re only limited to what they know, than what you know. So it’s better to get to know people by exploring your group than it is to go in alone.

He notes Christopher and Joshua’s expertise (in CS) and interest (in games) and, like Carolina in some ways, steps back, fitting his voice into the work almost as if he provides the canvas for their colorful ideas. Where his earlier goals were about working and money, his investment in this particular project shifts his attention and anxieties towards the project:
Money was pushed back, definitely. I wasn’t thinking about money as much as I was thinking about having to go into work and to do the work with other people. I think that was much more important as I went through the program. I actually forgot about getting a check at all last week.

Where his early reflection cards talked about his desire to be louder or do more, the later ones say things like “worrying about time for work vs. time for other activities”.

By the end of the summer, though, it becomes less about even the work and how much he has learned, and more about the people. At the beginning, I had hoped that it would be Lange’s interest in digital animation would draw him into Scratch, but because Scratch is his first programming language but not his first animation editor, he finds it “very annoying, very finicky, and very unsatisfying.” In this last group project, Christopher and Joshua, excited about a different programming language called Processing (a text-based and somewhat more complicated interface), abandon Lange entirely to making the Scratch component of their project, so he moves from being a reluctant coder in the previous group to diving in wholeheartedly, creating a Scratch project about their design process. When talking to him about the project, he reveals his own anxiety that the whole project will not come together, and in the past few days we have project time, he works with laser-like focus, headphones on, distraction-free.

It is surprising, then, that in one of those “footnotes” in his journal, angled and barely readable, he writes: “I would much rather present the bond that I think we’ve grown for each other. we love jokes, haha.” The jokes his group loves, though, seem to be more like underlying commentary on their projects – always the assertion, the statement of what they did, the “this is what we’re working on” or “this is what I’m doing,” and then later, the follow-up, the reflexive “haha, hopefully I’m doing it right,” or “hopefully this works out.” As an outside observer, the humor seems to, perhaps, act as a deflection from the underlying anxiety about completing the project successfully.

Reflections

In the past week, we ask students to bring digital artifacts to their exit interviews, to help us learn more about which projects they loved and are proud of (Figure 4). Lange brings up a project I had forgotten about, one of the 1-h exercises that we did earlier on in the summer (as opposed to the weeklong projects), a silly animation with a jumping cat that says, “I’m Jesus,” over and over and over again. When asked, he says, a note of pride resonant in his voice:

All the coding on this one’s me. The art and the background are also me […] I think that surprised me because I don’t think I’m very fluent in coding […] so being able to do something as small but as silly as this, in basically forty minutes, is a little mind-numbing and a little mind-blowing.

Figure 4.
Scratch blocks for “hello world”
Carolina does not use the same words, but she also picks out an early project, an environmental game where a shark has to avoid trash in the ocean. When I ask why she brought in this one, she says:

It was when I was just a beginner and I was figuring things out […] I think that with this project I learned the most and I can go back to this project and clean it up a little bit. Make it better, now that I know more.

I laugh and ask whether she still has such strong feelings about STEM, but she tells me she likes Scratch now. “I’m hoping that I get to use it for a project at [the high school] and, like, impress some people. I can code, ho-ho!” She chuckles and swings back and forth in her seat, looking pleased with herself.

Carolina was excited to use Scratch in school at the end of the summer, but I did not know if she actually had the opportunity to do it. Months later, I contact Lange and Carolina asking to do follow-up conversations for the purposes of writing this portrait. Lange readily agrees, and we decide to meet on a cold drizzly Sunday afternoon, in the same room that our exit interviews were conducted in. He is prompt, showing up in a “I stand with Standing Rock Mni Wiconi” sweatshirt, dark jeans and Nike sneakers. After he tells me about his college applications and the schools he is considering, I explain to him a little bit about what a portrait is and why I might want to write one about him. He is easy-going and far more talkative than I ever saw him before, expounding at length about his opinions on how he learns best (interest and relevance is key). Finally, I ask him about CS again, asking what his relationship to computers is, particularly in light of the fact that both his mother and his grandfather were, in his words, “black coders”.

He thinks before responding, looking up at the sky as he taps a finger on his chin as he composes an answer:

I liked looking at [my mom and grandpa] coding, watching them code, and sometimes saying, “Oh, I’d like to code”. And they were like, “Well, you should do it”. And then, not doing it, because I felt like it wasn’t the thing for me. That’s something that happened to me a couple of times. Computers are a part of my life now because I talk to friends all the time and play games. I do all these things with computers. But, computer science has never really been an option I’ve thought about […] a natural fear of all these people who would be better than me, or look down upon me because I’m not as proficient as they are, would be a thing […] And it’s also been a great setback on how I work around things, and how I do things. I can’t do this, other people will look down upon me, because I can’t do this, because I haven’t practiced it. So, I haven’t learned it. And because I haven’t learned it, and because these people have, all of a sudden, I’m not as good as these people.

The fear and hesitation Lange describes resonate powerfully for me; when I entered college, I had worried that it would be too late for me to learn something new, and it was that initial CS course that confirmed for me my belief that I was just not as good at this as other people. He then tells me that he still feels like, although “it was very nice to be able to do things in Scratch,” he still has the feeling “that it’s too late to pick it up; too late to learn it. And I also feel like I would prefer doing something else, as well”.

But he made so many things in Scratch this summer, I exclaim. Backtracking a little in my thoughts, I ask what else helped him learn in the program, and this time he does not need to think before he responds, citing “other people,” such as Christopher and Joseph, the other 17-year-olds in his last project group:

They knew what they were doing. Watching them do things, it was sort of like, “Oh this is way too easy”. It wasn’t depressing, because I don’t think they had expectations for anyone else to be as good as they were, because they’ve been doing it for a long time. Knowing that, I think it was like, “I can do whatever I want now”, because I don’t have the expectation to be as much as they are, as I do when I join a class for people who are trying to learn it.
In most learning opportunities available to us, we are often grouped by skill level, but what Lange has hinted at seems to be that internal pressure to succeed weighs heavily on him, and freedom from expectations helps him to learn. Where Carolina found Henri’s presence in a group useful because she could rely on his expertise, Lange never talks about getting help from Christopher or Joseph, but rather they seem to exist as models for what might be possible.

Carolina and I also meet in that study room, and she looks the same as I remember. Frustratingly enough, she tells me that she has not used Scratch since, though not for lack of want:

I actually haven’t gotten to use it yet. I’m so sad, because I was like, “This is going to be a great tool. I’m going to wow a bunch of teachers with my wonderful Scratch skills”. They’ve all been like, “No. You can do a slide presentation or a Google doc”. I’ve said, “This would be really cool in a Scratch project. Can I do that?” But they always say no. It’s so frustrating, I don’t get it.

Though Scratch is something that many young people primarily use outside of school in their personal creative practices, Carolina’s imagination is bounded within the realm of the classroom, perhaps because that is where she was introduced to it, both through Hour of Code and this program.

As a teacher, I do not really care if students “master” CS and enter STEM fields; what is more important is that young people “build capacity to think with tech about social justice” (Lee and Soep, 2016, p. 490). When I ask her whether any of the things that happened in this program have helped her in school, she suddenly lights up as she tells me about an incident that happened in physics class:

I got to realize the process of making something that might not seem like a lot into something that you can actually change. I actually got to use it for something in physics, actually. Physics is very problematic this year [...] There was a question that was really misogynistic. “It’s so problematic. I wrote an essay on it [...] I know!” I wrote an essay on it. I got to do the organization myself because it was my own project. I gave it to the physics teacher and told him that it was really problematic. Even if it’s such a small question, it’s so much bigger than that. I told him to bring it to the board and he did and they did nothing about it. I’m going to bring it to all the physics teachers and try to get it changed.

Last year I would have thought that it was really messed up and talked about it with my friends and stuff, but I wouldn’t have actually done something about it, because I didn’t think that was possible. With [this program], I think I got to realize that if you really care about something enough, it can be changed. Anything can be changed. And it’s really important to do that no matter how small it is.

Discussion

Through this portrait of two learners’ experiences, I highlight some of the ways in which identity, community and competence can support young people’s motivation and their development in understanding themselves as civic actors. Below, I describe four key points for educators before noting some of the limitations of this method.

*Personalized learning opportunities can support reluctant learners’ motivation for mastering key concepts.* Where Deci and Ryan (2000) consider autonomy, relatedness and competence to be central in understanding how intrinsic motivation supports learning, identity or the “personalization” aspect of constructionist learning, seems to be a strong factor in Carolina and Lange’s motivation, in terms of their dedication to their Scratch
projects and the program more broadly. As Brennan (2013) writes, “Personalizing, as a constructionist aim, means that the design of learning experiences should consider how to engage an individual learner on multiple levels, including cognitive and affective” (p. 49). When Carolina talked about the difference between her experiences in Hour of Code and the initial “About Me” project over the summer, much of the difference in excitement was connected to how personal the project was, both in terms of what she was able to do with the code blocks as well as the visual elements and design of the overall project. Although the tutorials in Scratch were not dissimilar from the Hour of Code experience, in terms of user interactivity with the content, the clarity of Carolina’s vision of what she wanted to design enabled her to break down her project into its core components (a key aspect of computational thinking) and able to cherry-pick what she needed for her larger project aspirations (Barr et al., 2011). Her desire to understand more challenging computational concepts was deeply influenced by what she wanted to design.

Throughout the summer, Carolina’s interests and her identity served as key motivating factors for creating Scratch projects, which enabled her to practice the problem-solving practice enough to better understand how she might be able to translate this process into other areas of her life, such as her issue with the physics class. But although personalized learning has become a popular term of late, there are also a variety of understandings of its definition (Friedman, 2018). Although some definitions include tackling the same material with differentiated pacing or order, those definitions do not offer learners true opportunities for agency and expressiveness, which can be important, particularly for reluctant learners. Learner identity must be understood not only through learners’ own desires and experiences, but also the experiences of their families. Adult role models and mentors have often been thought of as inspiration and supports for pursuing interest-based learning (Ito et al., 2013), but a closer examination may be needed of learners’ relationships to potential mentors and what factors create the conditions for inspiration and/or fear. Although Lange had positive role models in his mother and grandfather’s work as black coders, his family’s prior experiences also contributed to his reluctance to pursue coding seriously. For Lange, his perceived lack of skill in coding, despite being exposed to computers at an early age, was connected to a “natural fear” that others would look down on him, which in turn led to his lack of engagement with computer programming, prior to the summer program. At the same time, he cited his family as inspiration for his commitment to his project on diversity in the CS field. As educators and designers, we need to take into account not only an individual’s passions but also the fears, concerns and prior experiences that they might bring into the classroom. Although Lange considers his fear to be a natural and obvious one, it is not one inherent to all learning environments, and we should strive to create environments where young people can find both the “humility and courage” (Lampert, 1990, p. 59) to explore new things, fail and try again.

Peers can support one another in a variety of ways, but more scaffolding is often needed to support young people in helping each other. Within interest-based learning, scholars have pointed to the value of not only adult mentors but also peers of all experience levels (Resnick, 2017). Certainly, Carolina and Lange pointed to the value of their interactions with more experienced learners (typically older boys–Henry, Christopher and Joseph–who had completed high school CS coursework), and both participants noted multiple helpful instances where, as the teacher, my observations differed from participants’ reflections, whether because students were quietly fixated on their own screens, or because the experienced young people mentioned focused on their own work. Those instances were a reminder that we as teachers often sit on the outside of learners’ experiences, However, though Carolina and Lange had positive experiences with receiving support from others, we
as facilitators struggled to create the same opportunities for supportive peer experiences among other participants. Sometimes, this was because the gaps between participants was too wide, whether these were differences in experience, age or maturity, and sometimes, more experienced learners were not necessarily interested in teaching, in that moment. Introductory computing experiences continue to be implemented unevenly nationwide (Blikstein, 2018), and we as educators will continue to encounter learners with wide disparities in their prior computing experiences. Therefore, scaffolding opportunities for learners to develop their own fluency, while also learning to support others, will continue to be an ongoing challenge.

The needs of group projects can be in tension with the desires of learners or educators. In this portrait, I highlight some of the ways it can be difficult for formal learning environments to support young people’s simultaneous development of computational fluency as well as civic capacity, largely because of how much schools are asked to accomplish. Although identity, community and competence can support young people’s motivation, these values can be in tension with one another. Lange was able to build community with his group, which facilitated his interest in pursuing his project and actively participating in the program, but it was challenging for students to develop competence in the same set of concepts, as group projects require so many different roles. Different learners may have different interests and strengths, and the deep and meaningful learning they may want to experience (e.g. in terms of learning to do high-quality research, or their desire to improve their artistic design skills), as individuals can also be in tension with the desires of the group project needs (e.g. deadlines, or the needs for specific kinds of research to fit the research questions) (Rubin et al., 2017). Because one of Lange’s group members came into the summer with prior CS experience, and the projects were on a short timeline with pressure to do well in front of the rest of the community, with presentations to real world stakeholders, the group divided the work by student self-identified strengths, instead of areas of potential growth. In formal academic contexts, with time constraints and state-required learning outcomes, teachers may need to consider what their priorities are for their students, given their particular context. In informal learning environments, with more opportunities for longer-term and self-directed engagement, educators, librarians and other youth workers may have an easier time supporting a diversity of learning goals and styles, or “epistemological pluralism” (Turkle and Papert, 1990), as well as an overall vision of Coding for All (Martin, 2017).

Limitations
Because this portrait is of two particular learners in a singular experience, although it might be an authentic description of their experiences of engaging with computer programming activities, it may be challenging to extend beyond this portrait to other skeptical, reluctant and/or historically marginalized learners, given when and how this portrait was situated. Both Carolina and Lange came from well-educated families and participated in a fairly well-resourced public-school system; although Hour of Code and other CS introductory experiences felt commonplace to them, there are many other learners who live in low-resourced communities without any introductory CS experiences at all. As educators, we must strive to know more about the whole learner, as well as supporting learners in making connections in their learning across a variety of formal and informal contexts (Ito et al., 2013). Portraiture also affords greater understandings of the importance of context and the complexity of learners’ experiences and desires, and these understandings may be recognizable to other learners, educators and researchers, even when the contexts and the learners vary.
Conclusion
This portrait offers a glimpse into the ways in which reluctant, skeptical and/or marginalized learners’ identities influence their motivations and experiences in an introductory CS summer program. Prior to summer 2017, Carolina and Lange had been exposed to some CS, whether in the classroom or informally. Though computer programming was not brand new to them, they still had a lot of room to grow, particularly in understanding how programming could be powerful, as well as what they as programmers were capable of. As introductory computing offerings continue to expand, it is increasingly important to consider not only how the first experience is designed but also how that experience may connect to a learner’s future computing experiences. Important future research directions include exploring more of the qualities of introductory CS learning environments that can support skeptical, reluctant and/or historically marginalized learners, developing better understandings of how novice programmers connect personal interests to broader civic capacity, as well as creating design heuristics and core practices so that educators can better support youth in their creative digital pursuits.

Notes
1. Throughout this piece, all learners are identified by pseudonyms they have chosen.
2. https://code.org/

References
Blikstein, P. (2018), Pre-College Computer Science Education: A Survey of the Field, Google LLC, Mountain View, CA.


**Corresponding author**
Paulina Haduong can be contacted at: haduong@g.harvard.edu

For instructions on how to order reprints of this article, please visit our website: [www.emeraldgrouppublishing.com/licensing/reprints.htm](http://www.emeraldgrouppublishing.com/licensing/reprints.htm)
Or contact us for further details: permissions@emeraldinsight.com
Supporting women’s persistence in computing and technology
A case for compulsory critical coding?

Joanna Weidler-Lewis
Krause Innovation Studio, The Pennsylvania State University,
University Park, Pennsylvania, USA

Wendy DuBow
National Center for Women and Information Technology,
University of Colorado Boulder, Boulder, Colorado, USA

Alexis Kaminsky
Department of Consulting, Kaminsky Consulting, LLC, Albuquerque,
New Mexico, USA, and

Tim Weston
Alliance for Technology, Learning and Society,
University of Colorado Boulder, Boulder, Colorado, USA

Abstract

Purpose – This paper aims to investigate what factors influence women’s meaningful and equitable persistence in computing and technology fields. It draws on theories of learning and equity from the learning sciences to inform the understanding of women’s underrepresentation in computing as it investigates young women who showed an interest in computing in high school and followed up with them in their college and careers.

Design/methodology/approach – The mixed-methods approach compares data from quantitative surveys and qualitative focus groups and interviews. The sample comes from database of 1,500 young women who expressed interest in computing by applying for an award for high schoolers. These women were surveyed in 2013 and then again in 2016, with 511 women identifying themselves as high schoolers in 2013 and then having graduated and pursued college or careers in the second survey. The authors also conducted qualitative interviews and focus groups with 90 women from the same sample.

Findings – The findings show that multiple factors influence women’s persistence in computing, but the best predictor of women’s persistence is access to early computing and programming opportunities. However, access and opportunities must be evaluated within broader social and contextual factors.

Research limitations/implications – The main limitation is that the authors measure women’s persistence in computing according to their chosen major or profession. This study does not measure the impact of computational thinking in women’s everyday lives.

Practical implications – Educators and policymakers should consider efforts to make Computer Science-for-All a reality.

Originality/value – Few longitudinal studies of a large sample of women exist that follow women interested in computing from high school into college and careers particularly from a critical educational equity perspective.

Keywords Equity, STEM, Computing, Mixed-methods, Women, CS-for-All

Paper type Research paper

This research is supported by the National Science Foundation under CNS 1441071 and CNS 0813956. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.
Introduction

In 2016, then President Obama announced his vision for the “Computer Science (CS) for All Initiative” in which all students would have “hands-on computer science and math classes that make them job-ready on day one” (State of the Union Address). The CS-for-All initiative was part of a broader campaign to support and expand science, technology, engineering, and mathematics (STEM) education and employment opportunities for American youth. We are purposefully calling attention to this connection between CS and broader STEM activities as our research has been investigating what role early computing interest and opportunities for young women play in supporting their persistence in both computing and other technology fields. In this article, we compare the results of a quantitative survey measuring predictive factors for young women’s persistence in CS and non-CS college majors, with the findings from qualitative interviews and focus groups of the same population of young women. Specifically, we present research findings from surveys, interviews and focus groups that followed-up with young women in college and early careers who had shown an interest in computing during high school as demonstrated by applying for the National Center for Women and Information Technology (NCWIT) Aspirations in Computing (AiC) award.

The results of our qualitative and quantitative data differ. The qualitative data demonstrate that multiple factors influence women’s persistence in CS including not only women’s experiences but also how participation and persistence in computing is defined (Weidler-Lewis et al., 2017). Analysis of our interviews and focus groups suggests that we ought to have an expansive view of both CS participation and how to cultivate this participation, which would encompass more than engaging in coding as the only outcome measure of persistence. Our survey data, on the other hand, show that women who persist not only in CS (e.g., those fields that require computational thinking) but also other non-CS technology fields (i.e., computer graphic design) had access to programming opportunities as youth (Weston et al., 2019). Thus, we draw a narrower conclusion that prior experience with coding specifically – as opposed to other related technology experiences such as game design or web development – is the strongest predictor of women’s persistence in both computing and technology related majors. In this article, we analyze these seemingly disparate findings within a coherent narrative and what this means for supporting women’s persistence in computing and technology. We seek to understand what factors influence women’s meaningful participation and persistence in computing? And, how do our qualitative and quantitative findings on women’s persistence in computing complement and/or contradict each other?

Background

The underrepresentation of women in computing continues to be an unsolved yet highly investigated problem. According to the National Center for Educational Statistics, the percentage of women pursuing CS-related majors has declined over the past three decades from 37 per cent of bachelor’s degrees earned in 1985 to 18.7 per cent in 2016 [National Center for Women and Information Technology (NCWIT), 2018]. Although women enroll at lower rates than men, there is little difference in attrition rates by gender in college degree programs (Cohen and Deterding, 2009; Dee and Gershenson, 2017). Research has focused on what spurs interest in computing such as early exposure, access to rigorous computing opportunities and peer support (Google, 2014; Teague, 2002) and what hinders women’s participation in computing, including gender-bias, micro-aggressions and lack of community support (Camacho and Lord, 2011; Rosson et al., 2011; Smyth and Nosek, 2015). The problem of underrepresentation has persisted for so long that numerous efforts have
been made to synthesize all the research to date and possible solutions (Corbett and Hill, 2015; Kanny et al., 2014). Despite the enduring problem of the lack of women’s participation in computing, there have been multiple calls to increase their participation [National Research Council (NRC), 2011; National Center for Women and Information Technology (NCWIT), 2015; The White House, 2009]. Our research attempts to support this call.

NCWIT is a non-profit community organized to support the increased meaningful participation of women in computing. Multiple strategies are used to further their mission, including convening leaders to collaborate on how best to implement change; offering award programs and incentives for girls, young women and educators in computing; and providing research-informed resources for stakeholders wishing to change their computing environments for students and professionals alike. The research presented here comes from a mixed-methods study of one of their prominent award programs for high school young women: the Aspirations in Computing (AiC) award. The research was conducted over six years and includes AiC application survey data, data from survey instruments administered in 2013, 2016 and 2018, and data from interviews and focus groups of 90 women conducted between 2012 and 2017. The research team is interdisciplinary and brings multiple perspectives on education, learning, and the social factors that interact with women’s persistence in computing. Given our differences, and the longevity of the project, our thinking and our approach to analysis have evolved over time and this evolution is apparent in the presentation of this work below. Next, we present our current thinking on equity and women’s meaningful participation in computing before presenting the model used for analyzing persistence.

**Equity and computer science education**

As Social and Learning Scientists, we are mindful of the ways in which our research can disrupt or contribute to injustice and inequity in our society. For example, the way in which the underrepresentation of women in computing is often framed – and we invoked this framing by including Obama’s call for a STEM ready workforce – is that equity would be achieved if women were represented in 50 per cent of computing jobs. Although this is a goal that as researchers for NCWIT we hope is attained, we also recognize that it oversimplifies both what “equitable” participation entails and the ways in which computing as a discipline is tied to systems of power and embedded within complex sociopolitical contexts that are inherently fraught with competing economic and political interests (Vakil, 2018). Furthermore, we have shown that “successful” participation in STEM activities does not necessarily mean that young women do not still suffer gendered consequences despite their success (Weidler-Lewis et al., 2016), and pushing women through the “STEM pipeline” only to earn significantly less money than their white male counterparts is hardly equitable (Sengupta-Irving, 2015). Therefore, we must question how we define our goals for success and equity in STEM disciplines (Carlone et al., 2011).

Another way of framing equity in computing is to recognize that computational thinking and computer literacy are fundamental problem-solving skills that all students ought to have a right to develop (diSessa, 2001; Wing, 2006, 2008). This perspective varies in kind. For example, Soloway (1993) argues that programming and computational modeling represent core scientific practices that ought to be part of the curriculum, while other educators see computational literacy as any other form of literacy that should be recognized and valued as a way in which we make sense of our place in the world (Ito et al., 2013; New London Group, 1996). As Gutiérrez (2008) would argue, literacy tools empower youth and support their individual and collective pathways toward just and equitable futures. In this socio-critical tradition, Lee and Soep (2016) call for critical computational literacy that brings
together the power of computational thinking (Wing, 2006) with critical consciousness (Freire, 1993). From this perspective, youth have the opportunity for both self-determination supporting their pathways toward their chosen college and careers and recognizing themselves as cultural-historical actors with the ability to create a more just and equitable world. We as educators, then, simultaneously can have the goal of increasing participation from a purely numbers perspective such that women represent 50 per cent of computing majors and careers while also working to change the social practices of computing so that they are just and equitable for all.

As educators committed to equity, we attempt to understand learning holistically and not just analyze how people learn but also interrogate “for what,” “for whom,” and “with whom” learning takes place (Philip et al., 2017). With this in mind, we recognize that advocating for computer science education for all means we need to be vigilant in promoting equitable computer science education. We take learning to be a process of becoming in which identities are constructed through participation, and through participation communities are produced defining who belongs and is valued (Holland et al., 1998; Packer, 2010). An individual’s identity is shaped both by her goal-directed activity and the subsequent recognition by her community (Gee, 2008). Learning is a mutual process constituting both the individual and her community. Although it is important to focus on the process in its entirety, the research described below focuses mostly on the individual as our survey and interview protocol were based on Lent’s (2000) Social Cognitive Career Theory (SCCT).

SCCT holds that a variety of person, environmental and behavioral variables influence career choice. The model defines internal and external factors that support or inhibit career and education decisions, including interest, self-efficacy, outcome expectations, perceived social supports and intention to persist. One of the premises of the model is that students with high self-efficacy, or confidence in their computing understanding and skills, will be more likely to persist in computing in college and careers. Increased self-efficacy promotes favorable outcome expectations or the belief that actions will result in expected outcomes. Self-efficacy and outcome expectations alone and together support career interest and goals. Career choice is also influenced by contextual factors such as social support. The SCCT model has been used to study factors related to the under-representation in STEM (Fouad and Santana, 2017; Lent et al., 2008, 2011) and so we selected it as our initial survey model.

Methods and data
The research question is as follows:

**RQ1.** What factors influence women’s meaningful participation and persistence in computing?

To answer **RQ1**, we engaged in two distinct data collection phases:

1. a series of surveys, including the initial application survey; and
2. qualitative interviews and focus groups.

Using a “convergent parallel design” (Cresswell and Plano Clark, 2011), the results of each investigation are compared and triangulated to give a more complete understanding of women’s persistence in computing. In this section, we describe the sample for the two strands of research followed by a description of each strand. After presenting the findings for each, we integrate the results and draw connections to answer the question, ‘How do our
qualitative and quantitative findings on women’s persistence in computing complement and/or contradict each other?

Sample
The sample of women in this research comes from a database of winners and non-winners for the NCWIT AiC Award. Anyone who registered on the program website between 2009 and 2013 or had won the award in 2007 or 2008 (prior to the existence of a digital platform) were eligible to be included in the sample. The award program began in 2007 and has attracted thousands of young women with some affinity toward computing or technology. The database, however, also includes any young women who registered on the award’s website but did not submit an application for the award. As part of the registration and application process, young women complete an “application survey” in which they rate a list of 20 computer skills and activities according to how often they engage with each ranging from “Not at all,” “Only a little,” “Pretty much” and “A lot”. We have application survey data from nearly all survey respondents.

Quantitative survey
As part of this longitudinal study, a series of three surveys were fielded between 2013 and 2018 (Table I). While the survey was revised with each administration, the same set of constructs was included in each administration to allow for case-level comparisons across time.

The initial survey administered in 2013 to the women in the database was based on the SCCT model as described above and sent via SurveyMonkey. NCWIT researchers developed the survey, and the five constructs of the SCCT model were represented by 34 items: 9 interest items, 7 confidence items, 5 intent to persist items, 7 perceived social supports/barriers items and 6 outcome expectation items. Survey items were organized in blocks aligning with the SCCT model. Regarding interest and confidence, items asked about designing computer games, trying new computer software, fixing or building computers, programming computers, inventing technology and finding technological solutions to world problems to name a few. Items designed to measure perceived social support asked questions about perceived family and peer support such as, “My family likes me to learn about technology” and “I believe people like me can do well learning computing”. A final open-ended question asked women about their college, military or work position they were holding at the time of the survey.

The second administration in 2016 of the survey went to the 1,500 respondents from the first survey administration. In the 2016 survey, 511 respondents were either in college or early career. The second survey contained the same questions as the first but included questions asking the women about their college major or occupation and title, if they were in the workforce. From this “where are you now” question, a dependent variable of “persister/non-persister” was identified. The levels of this variable included: CS-persister for those women who were pursuing or who had graduated with a CS or Computer Engineering degree; Tech-persister for those women who were pursuing or who had graduated with a technology-related degree (other than CS or Computer Engineering); all others respondents made up the Non-persister group. Table II shows the number of women in each group.

<table>
<thead>
<tr>
<th>Year</th>
<th>No. in sample</th>
<th>Respondents</th>
<th>Response rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>9,860 (all those who had registered on the website)</td>
<td>1,613 (1,500 usable)</td>
<td>16</td>
</tr>
<tr>
<td>2016</td>
<td>1,500 (all usable surveys from Survey #1)</td>
<td>885</td>
<td>59</td>
</tr>
<tr>
<td>2018</td>
<td>1,500 (all usable surveys from Survey #1)</td>
<td>795</td>
<td>53</td>
</tr>
</tbody>
</table>
Our survey data were analyzed to see which model of persistence was a better predictor of continued CS persistence in college or beyond. The first model was based on SCCT, and the second model is called the “domain” model. The domain model included the same items as the SCCT model, but they were rearranged to allow for a different analysis. In the SCCT model, one item from each subdomain (e.g. gaming, programming and inventing applications) was represented in each psycho-social SCCT category (e.g. interest and confidence). In the domain model, we grouped items by subdomains corresponding to a particular underlying skill including programming, game design and inventing new applications. See Table III for examples. We did this to learn which grouping method better fit the data while controlling for number of total parameters estimated by the model. Although the original SCCT model (Lent et al., 2000) includes “intent to persist,” we did not include this item block in our analysis because it was redundant with our persister variable mentioned above, which measured actual versus intended persistence.

We used the statistics software SPSS AMOS 24 to conduct confirmatory factor analysis (CFA). CFA tests the comparative fit of hypothesized model (e.g. SCCT) with alternative models (e.g. the domain model) to empirical survey data. We used the following fit indices: Chi-square, root mean square of approximation (RMSEA), comparative fit index (CFI) and sequence robust multi-array analysis (SRMA). Using the Maximum Likelihood estimation procedures, we followed the best practices for standards of model fit (Hu and Bentler, 1999; Mueller and Hancock, 2008). After checking the adequacy and assumptions of the data for modeling we found:

- an acceptably high ratio of persons to parameter for both the domain and SCCT models, 14:1;
- univariate and multivariate normality were within acceptable ranges for skewness and kurtosis for individual variables and multivariate normality; and
- the reliability of composites were adequate to good, ranging from 0.77 (gaming) to 0.85 (programming).

<table>
<thead>
<tr>
<th>Group name</th>
<th>CS-persister</th>
<th>Technology-persister</th>
<th>Non-persister</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of women</td>
<td>n = 177</td>
<td>n = 137</td>
<td>n = 181</td>
</tr>
<tr>
<td>Description</td>
<td>CS and Computer Engineering majors</td>
<td>Pursuing (non-CS) information technology related major or other engineering major</td>
<td>Students not in CS, engineering or information technology related majors</td>
</tr>
</tbody>
</table>

Table II. Three groups in the persister outcome variable

Table III. Example constructs from the SCCT and domain models

Regardless of whether or not you have actually tried it [...]
How interested are you in programming computers or other technologies (In other words, writing code)?
How interested are you in thinking of new technology inventions (For example, new apps or software, improved tablets or MP3 players)?
How interested are you in actually creating new technology inventions (For example, new apps or software, improved tablets or MP3 players)?
How interested are you in finding technological solutions to world problems?

How interested are you in programming computers or other technologies (In other words, writing code)?
How confident are you in your ability to program computers or other technologies (In other words, writing code)?
How much do you want to learn more about programming computers or other technologies?
Based on the data from the 20 items from the applications survey, we created four factor variables using Maximum Likelihood Extraction and Orthogonal rotation procedures; the four factors accounted for 43 per cent of total variance. These four factors were programming, technology work and community, multimedia and network. Composites for these factor variables were created with the “regression” method in SPSS. For our predictive models, we used multinomial regression to (simultaneously) predict the CS-persister and the technology-persister dependent variables. Comparisons for multinomial regression were between technology-persister and non-persister on one hand, and CS-persister and non-persister on the other. A more thorough description of our analysis is presented in Weston et al. (2019).

**Interviews and focus groups**

The interviews and focus groups of 90 women took place in two phases. The initial 64 women interviewed were recruited from the AiC award sample described above and interviewed either individually via telephone, videoconference or in person or via videoconference focus group between 2012 and 2015. We recruited women through both email and personal phone calls, and they were offered an incentive of an iTunes gift card to participate. Both the individual interviews and focus groups followed a similar semi-structured protocol that inquired about the women’s experiences and perceptions about computing and SCCT concepts and others such as belonging and identity relative to computing. We were also interested if winning or not winning the AiC award impacted their attitudes about computing. Our initial analysis showed that winners of the award were more likely to be persisters (39 out of 41, 95 per cent) than non-winners (11 out of 20, 55 per cent).

In our initial sample, award winners (in particular national winners as opposed to regional winners) and women who persisted had greater representation than non-award winners and non-persisters. In the second phase of the qualitative study, we increased our efforts to focus on these latter categories. We focused our recruitment on those women who:

- had returned the study surveys administered in 2013 and 2016;
- were not national winners or runners up for the award;
- were no longer pursuing computer science based on their responses to survey questions asking what they were currently studying or what field they were working in; and
- had not been interviewed for this project before.

In total, 281 women fit these criteria. We utilized stratified random sampling to select potential participants to contact, stratifying the sample by winner and application status (i.e. seeking non-winner and no application). We further stratified the sample by race and ethnicity to ensure representation from multiple racial and ethnic groups. We recruited from this stratified sample and contacted 59 women. We also noted that our original recruitment script and interview protocol may have turned away women who did not persist in computing, as they may have appeared to implicitly judge women who were no longer persisting in computing, so we reframed the study to emphasize the factors and experiences that influenced women’s educational and career choices more broadly instead of computing alone. The interview guide for non-persisters included questions about what women were currently doing and the factors influencing their choices, their high school experiences, the roles of gender and race/ethnicity in their educational and career decision-making and their sense of belonging in their chosen field. However, we retained the core questions about computer science and engineering experiences to facilitate comparisons between persisters
and non-persisters. Recruitment involved direct solicitation, including personalized emails and phone calls and the incentive was increased to a $75 Amazon gift-card. In Phase 2, we stopped recruiting after we had interviewed an additional 26 women.

All interviews and focus groups were recorded and transcribed. Transcripts were uploaded into Dedoose, a qualitative analysis program. We used a combination of a semi-emergent approach to content analysis (Cresswell, 1998) and grounded theory (Glaser and Strauss, 1967) to develop our coding scheme. Some codes were created based on SCCT (e.g. self-efficacy, outcome expectations) and those identified in reviews of the literature such as Kanny et al. (2014) review of the past 40 years of research on women in computing which identified five metanarratives: individual background characteristics; family influences and expectations; structural barriers and affordances in K-12 education; psychological factors, values and preferences; and perceptions of STEM fields. To this, we also added a sixth sub-code, post-secondary barriers and affordances, because the majority of women who participated in the interview component of our study were in college or working. We also remained open to constructs that emerged from the data themselves such as belonging, and we iteratively refined our codes.

Each transcript was coded by at least two researchers and all new sub-codes were reviewed by a second researcher to make sure they were consistently applied. During coding and analysis, the research team met regularly to define and refine codes and work toward inter-rater agreement. Coding disagreements were discussed by the research team. Often the disagreements were due to the different experiences and bodies of literature that individual researchers were familiar with. These discussions enriched our understandings of the data from multiple perspectives.

Findings
In this section, we present condensed findings from our two strands of research. A summary of our findings allows us to make comparisons between the qualitative and quantitative data that might otherwise be obscured by a comprehensive discussion of each study alone. For a more detailed description of the survey findings, see Weston et al. (2019) and a more detailed description of the qualitative research see DuBow et al. (2017). By presenting the findings from the two strands separately, we address the question, “How do our qualitative and quantitative findings on women’s persistence in computing complement and/or contradict each other?” Taken together, our two strands of research weave an argument for how best to understand the data used to answer the question, “What factors influence women’s meaningful participation and persistence in computing?”

Survey findings
As discussed above in our analysis we examined what prediction model was a better fit for our survey responses including the shortened SCCT model with the outcome variables removed and the domain model, each using the same 16 survey questions in different configurations. We tested these models and the original (longer) SCCT model and calculated fit-indices, factor loadings, and factor correlations. Items most associated with Programming showed strong factor loadings from 0.62 to 0.90. Those associated with Game Design ranged from 0.55 to 0.89. Three items constituting Inventing New Applications had factor loadings from 0.58 to 0.84. Finally, the four items making up Social Support had factor loadings from 0.47 to 0.86. Confirmatory Factor Analysis showed that the original SCCT model fit the data poorly according to cutoffs defined by Hu and Bentler (1999) with $c^2 = 2,472$, $df = 293$, $CMIN/df = 5.3$, $CFI = 0.77$, $RMSEA = 0.13$ and $SRMR = 0.077$. The shortened SCCT fit slightly better, but still fit poorly with $c^2 = 652$, $df = 84$, $CMIN/df = 10.5$, $CFI = 0.77$, $RMSEA = 0.13$ and $SRMR = 0.077$.
CFI = 0.81 and SRMR = 0.087. The domain model on the other hand met or nearly met model fit standards with $c2 = 242$, df = 84, CMIN/df = 2.9, CFI = 0.95, RMSEA = 0.061 and SRMR = 0.067.

Because the domain model was a better fit to our empirical data, we examined how well composites made from the domain model predicted computing persistence from high school to college. Given that our sample contains longitudinal data of women’s actual persistence, we are uniquely positioned to measure factors related to actual persistence instead of merely intent to persist. Furthermore, our large sample of women had a variety of computing experiences based on their responses to the initial application survey. We had the opportunity to examine if the composite variable from the best fitting model predicted the dependent variable for both CS-persisters and technology-persisters. To do so, we created a multinomial regression model that compared them.

The Programming composite variable significantly predicted CS persistence ($p < 0.001$), with one standard deviation unit increase corresponding to approximately four times more likely persistence (odds ratio equal = 4.12). Programming also significantly predicted persistence for technology-persisters ($p < 0.001$), with a lower odds ratio of 1.6. Other variables were significant predictors of persistence such as being AiC award winners and taking the CS Advanced Placement Exam. We were surprised to find that variables such as social support and game design did not predict persistence in any model. Although there are limitations to the survey that could explain the lack of predictive power for social supports that could be addressed in subsequent measures, we argue that it is significant that Programming engagement was the best predictor of persistence in both CS and the larger category of Technology persistence. Our findings suggest that high school girls who become involved with the more technical aspects of computing early on have a greater likelihood of pursuing CS or other tech-related majors in college.

Qualitative findings
Unlike our survey findings in which one factor, early programming experience, was clearly significant for women’s persistence, our qualitative data showed that multiple, even redundant factors influence individual women’s persistence rather than one factor alone. It is easy to identify where some women land on the spectrum of supporting and inhibiting factors. For example, when comparing the profiles of Joan, a persister, and Sophia, a non-persister (pseudonyms), two women interviewed during the first qualitative phase, you can see that one has multiple supports while the other does not. Joan’s persistence is attributable to having parents in the tech industry, living in the Pacific Northwest – one of the densest technology areas in the USA – having access to AP computer science classes and having friends accompanying her to tech-related afterschool activities. Sophia, on the other hand, grew up in an agricultural town in California’s Central Valley, where “they didn’t really do much about technology”. She was a first-generation college student, her community was lacking in “computer people” as she called them, and her high school only offered one computing class. Despite the fact that she “really liked” her computing class, she also liked animal science. This field would allow her to stay close to her family and she believed it would be easier for her to find a job after graduation.

This is not to say that all young women like Sophia will not persist. Of the first 64 women we interviewed, over three quarters of them persisted in computing. We focused on identifying themes related to women’s persistence including those women who unlike Sophia persisted in face of obstacles. We identified three general themes that contribute to women’s persistence:
sufficient exposure to learn computing skills, whether in school or out of school;
(2) sufficient community support, including teachers, parents, and peers; and
(3) respect and encouragement to feel they belong in computing and, thus, to develop a computing identity.

The third theme called into question our various views as researchers of what identity means. At a basic level, an identity refers to a particular kind of person in a given context (Gee, 2000), but the construction of identities occurs through participation (Lave and Packer, 2008). As we more closely examined women who were or were not persisting, we began to question both how we as researchers and the women themselves viewed “participation in computing,” recognizing the need to be explicit regarding how we defined participation in computing (Weidler-Lewis et al., 2017). Participation in computing often has been defined by a disciplinary perspective that values computational thinking as being the critical skill representing CS. We argued that this perspective unduly excludes some women (and likely men) from being seen as participating in a more expanded view of computing, namely a community of practice view of computing (Lave and Wenger, 1991). From this perspective, an individual is seen as a member of a community not necessarily because of an acquired skill, but rather because she is seen and sees herself as identifying with the community.

To demonstrate this, we use the example of two young women who would be classified as “not persisting in computing” according to the disciplinary perspective because they were not explicitly engaging in computational thinking; however, they were both sophisticated users of technology who many would argue should be seen as “persisting in computing,” including themselves. We also complicate our understanding of women’s persistence by questioning the roles teachers play in helping to alleviate the underrepresentation of women and in our efforts to support women in computing: Should we encourage women who want to teach the women to go into computing? If so, how do we classify their participation and persistence? After all, former President Obama believed we needed to make it “a priority to train an army of (STEM) teachers […] to make sure that all of us as a country are lifting up these subjects for the respect that they deserve” (April, 2013).

In the second phase of the qualitative interviews we focused our efforts on more deeply understanding non-persisters. Of the 26 women we interviewed during this phase, 23 were no longer pursuing computing based on a narrower definition of computing that only included computer science or computing engineering disciplines. If we consider a broader definition of participation in computing, such as web design and graphics, and if we include STEM teaching, seven more women from both phases of interviewing could be counted as persisting.

Again, a prevalent theme in the interviews was that early exposure matters. Those who persisted had early and continued access to computing education opportunities, 10 of the 26 interviews specifically mention access and exposure. Some of the non-persisters had little (e.g. one computer class in high school) or no computing classes at all. Other women commented on how their chosen field was more readily accessible and offered greater opportunity to succeed. For example, one woman expressed how she was “too engaged” in her International Baccalaureate science classes to even notice what access, if any, she had to computing. One biology major stated that her only option for computing was an elective:

I wanted to do computer science because that would be my next elective choice. However, by that time it was already booked by the other students. So I think […] I don’t know what I chose after that. But had I taken that course I might have changed my career path […] But I didn’t get that head start and I felt that I would have been behind in computer science and it wasn’t enough to turn me as I already had an initial interest in biology.
Others shared their concerns about being “behind” in computer science compared to peers with more experience. For example, a finances major told us, “There just wasn’t really very much open to me. And I think I might have chosen a career more in that area had I had more opportunities when I was younger. My first experience was computer science at a high school level”. She compared her experience to women in the AiC awardee community with whom she did not believe she could compete:

I mean they’ve probably computed before they spoke their first word! I mean, that’s you know, an exaggeration, but they’ve been doing it forever […] that just wasn’t something that was in my high school.

This second set of interviews provided further evidence that multiple factors enable and hinder women’s persistence. For example, having family support and positive role models support persistence while lacking in either hinders persistence. Interest and self-efficacy impact how women understand themselves in relation to coding. For example, one woman chose Web design over coding because, “I hated coding! I hated every second of it. Like coding and me just did not get along”. Another IT help desk worker said, “I couldn’t understand like what coding was, like how it all came about just because there was, again, if you missed a comma or a period like the whole thing could not work”. One last notable finding is that women did express their concern with being a woman in a male-dominated profession. For example, several women shared they were the only women in their computer science class, or that they felt their teacher was sexist: “It was like being a girl in a boy’s club”.

*Comparing quantitative and qualitative findings*

An initial take-away from comparing the quantitative and qualitative findings is that early exposure and access matter for persistence. Although the quantitative and qualitative findings are consistent, we ought to consider what the qualitative findings bring to bear on the quantitative survey, lest we conclude that solving problems of exposure and access will remedy all issues of equity.

The quantitative findings suggest that young women who are involved with the more technical aspects of computing have a greater chance of pursing both CS and other tech-related majors in college. It is important to note two limitations to this finding. First, we have not ruled out that exposure to non-technical aspects of computing may contribute to young women ultimately choosing to engage in more technical aspects of computing. Second, we have no evidence regarding the ways in which computational thinking is valued in the daily lives of those women we classify as non-persisters. From a socio-critical literacy perspective, we ought to value the choices women make in self-determining their lives (Gutiérrez, 2008). Additionally, we need to value those who are contributing to the success of others, such as the woman who became a technology teacher for K-8 students. As she said, “They needed someone. There aren’t a lot of people who are interested in teaching in grade school who also know stuff about computer science”.

Our interview and focus group data remind us that teaching and learning in STEM is a historicized and relational practice (DiGiacomo and Gutiérrez, 2016). Computing has been seen as a white, male discipline and this is not easily ignored nor rectified. We heard stories of a computing teacher who “was so disrespectful to women. He just always feels like their opinions are wrong and doesn’t pay attention to them”. One young woman told us:

What I experienced of the field in computer science in general, turned out to be a very, like, straight man’s field. I am not straight, I am not a man. So, it was very awkward for me to be in that environment.
Women of color are confronted with additional obstacles, such as a Hispanic woman who felt she always needed to “prove” herself in her computing experiences in ways her white peers did not.

Creating access to computing alone does not ensure that these opportunities do not act as “gate-keepers” to more advanced opportunities. Women expressed how introductory classes in particular are often designed to weed out weaker students, presenting a challenge she called “traumatic”. Describing her own experience, one woman said:

A lot of times in engineering they would skip all of the foundational stuff that maybe I really needed to go over and they would go right into the more difficult stuff that I already knew I couldn’t understand . . . it moved very quickly.

It is important to remember that access and preparation are not the same thing.

Discussion and conclusion

At one level, our findings are neither novel nor unexpected in that they mirror many of the findings from the previous research discussed above including access, support, and bias. On the other hand, our findings are unique given both the sample size and longevity of the research. Longitudinal research on women’s interest and persistence in computing is sparse. This is due in part to the fact that the CS-for-all-initiative is relatively new, so before, many women’s early computing experiences were in informal settings that present challenges to following-up with students. Given the national presence of NCWIT, we were in a privileged position of having access to hundreds of women allowing us to make claims from a larger, more diverse population than smaller studies.

Comparing across the qualitative and quantitative data, it is evident that several factors influence persistence in computing, but common to both is that early programming opportunities are significant factors in supporting persistence. This is consistent with other research on access and exposure (Google, 2014), as well as prior work that studied middle-school girls’ persistence over time (Friend, 2016). Although we do not want to diminish the importance of other factors such as peer and family support, or how compounded factors work together to support or hinder persistence as in Joan and Sophia’s cases (DuBow et al., 2017), we believe this finding is compelling to focus on for several reasons. The most important reason is that among the multiple factors we identified, creating opportunities for women to engage in programming is something that educators have the power to control and commit. Unlike social supports, or contextual factors such as poverty, minority status, or geographic location, educators and policymakers can have great influence in the lives of all students by establishing computing as compulsory across the K12 curriculum. Although this would not be without its challenges, before we discuss those challenges, we present three ways in which compulsory coding potentially would have impacted the women in our studies.

First, if all students were required to engage in computational literacy courses in the same way all students are required to take English and Math classes for example, girls and boys would be equally represented in computer science classes. So, women in our study who were the “lone” female in class would not be subjected to this same experience. However, we know that women are underrepresented in other STEM disciplines beyond computing and as they progress in their education and careers, the “leaky pipeline” results in fewer women than men. While K12 educators cannot control college environments and thus women’s persistence over time, we can establish a level playing field from which all students can start, and from there, continue to investigate the causes and factors that lead to attrition.
Second, if we establish a baseline criterion for all students to meet with respect to computational literacy, we can begin to equalize students’ experiences toward this criterion. As our qualitative results showed, some women felt “behind” compared to their peers because they did not have the same level of access to computing experiences. We know that currently there is no federal policy dictating the minimum requirements for high school graduation and that it varies from state to state (and in some cases from district to district) (US Department of Education, National Center for Education Statistics, 2008), as only 15 states have policies to provide CS education to high school students and only 6 states provide access to K-12 students (Code.org, 2018). Rectifying this imbalance would alleviate the situation where women lamented that classes were unavailable.

A reasonable objection to implementing compulsory coding in K12 schools is that by merely providing access, it does not follow that interest will increase. Although this argument is not without merit, the same argument is not made with other disciplinary domains such as math or literacy. It does not follow that as not all students will become mathematicians, not all students need a basic level of math understanding. When it comes to persistence, our survey results demonstrate that interest is not a better predictor than access to early programming experiences. Furthermore, early programming experience leads to not only greater persistence in computing fields specifically, but other technology fields as well. Therefore, while interest in programming specifically may not increase with access to programming, a greater sense of identification with the community of computing is likely to occur. We argue this is what occurred for several of the women who were no longer persisting in the narrow definition of computing but were successful in technology or teaching fields. For example, the majority of women in our study who said, “coding and me did not get along” or “I didn’t like coding” ended up pursuing other technology majors such as computer graphic design. The third reason for compulsory coding is that it opens up the opportunity for students to define themselves in relation to this way of thinking and their broader place in the world in similar ways to how developing socio-critical literacy skills empowers students to define what is meaningful and significant in their lives.

Our perspective entails that computational literacy should not be considered simply as a means to an end resulting in a coding or a software engineering job. Rather, we ought to acknowledge both the different ways in which computational skills play a role in individuals’ lives and individuals’ rights to determine what constitutes socially valued outcomes for such literacy. Although Obama invoked “job-readiness” in his call for CS-for-All, he did not prescribe particular career paths and instead connected computer science to multiple professions including teaching, professional football, car mechanics, and nursing (2013, 2016). From an equity perspective, compulsory coding would support not only women’s persistence in computing and technology as our data suggest but it could also empower all students to envision themselves as makers, creators, and innovators of their futures. Approaches to teaching and learning computational literacy should accommodate such visions if we truly want to implement CS-for-All.

To be clear, we are not advocating for a sterile implementation of a programming or one-size-fits-all computational literacy curriculum. As critical educators we believe that all learning and schooling takes place in broader societal contexts with challenging structural inequalities, and if we do not take this into consideration, disparities in achievement will persist. Our findings corroborated that structural factors matter. They also highlighted how it may take only one “sexist” teacher to turn a woman away from pursuing CS. With this in mind, we should build on the work that has begun to look explicitly at how equity emerges in CS classrooms (Fields and Enyedy, 2013; Lewis and Shah, 2015) and models for equitable CS education (Vakil, 2018). We should encourage teacher education programs to support
equitable learning practices (Darling-Hammond, 2008) and address the shortage of qualified CS teachers (Ladner and Israel, 2016). Finally, we can look to other disciplines that grapple with similar access and achievement problems such as mathematics education (Lubienski, 2008), and explore how to make disciplinary practices meaningful in students’ lives (Moll et al., 1992) and encourage disciplinary engagement (Nasir and Hand, 2008).

To ameliorate the complex problem of women’s underrepresentation and lack of persistence in computing, we must have complex solutions that embrace the totality of what CS-for-All entails. As the CS-for-All initiative gains traction, we will have more opportunity to investigate how best to support women’s persistence and look specifically at how early coding opportunities contribute not only to women’s persistence in computing and technologies fields, but also how these opportunities help to foster critical computational literacy in other areas of all students’ lives beyond their career choice. Until then, we must recognize our role as researchers, educators, and policy-makers in making equity in computing possible.

References


Corbett, C. and Hill, C. (2015), Solving the Equation: The Variables for Women’s Success in Engineering and Computing, American Association of University Women, Washington, DC.


Further reading


Corresponding author
Joanna Weidler-Lewis can be contacted at: jrw96@psu.edu
Developing a Scratch-based coding achievement test

Ömer Demir and Süleyman Sadi Seferoğlu
Department of Computer Education and Instructional Technology, Hacettepe University, Ankara, Turkey

Abstract

Purpose – The lack of a reliable and valid measurement tool for coding achievement emerges as a major problem in Turkey. Therefore, the purpose of this study is to develop a Scratch-based coding achievement test.

Design/methodology/approach – Initially, an item pool with 31 items was created. The item pool was classified within the framework of Bayman and Mayer’s (1988) types of coding knowledge to support content validity of the test. Then the item pool was applied to 186 volunteer undergraduates at Hacettepe University during the spring semester of the 2017-2018 academic year. Subsequently, the item analysis was conducted for construct validity of the test.

Findings – In all, 13 items were discarded from the test, leaving a total of 18 items. Out of the 18-item version of the coding achievement test, 4, 5 and 9 items measured syntactic, conceptual and strategic knowledge, respectively, among the types of coding knowledge. Furthermore, average item discrimination index (0.531), average item difficulty index (0.541) and Cronbach Alpha reliability coefficient (0.801) of the test were calculated.

Practical implications – Scratch users, especially those who are taking introductory courses at Turkish universities, could benefit from a reliable and valid coding achievement test developed in this study.

Originality/value – This paper has theoretical and practical value, as it provides detailed developmental stages of a reliable and valid Scratch-based coding achievement test.

Keywords Validity and reliability, Scratch, Coding achievement test, Coding education, Measuring coding skills, Types of coding knowledge

Paper type Research paper

Introduction

As first put forward by Perlis (1962), and later by Papert (1980), coding education has long been an important area of research and pedagogical development. The spread of coding education gained significant momentum particularly after Wing’s (2006) introduction of the concept of computational thinking (CT). Today, coding education is regarded as a requirement for digital natives (Prensky, 2001) to attain CT skills (Kanbul and Uzunboylu, 2017; Wing, 2006).

Aho (2012) defines CT as thinking processes containing the use of computational steps and algorithms in solving problems, CT is a skill that can be associated with Computational Thinker, a standard identified by International Society for Technology in Education (ISTE) in 2016 as one of the seven characteristics that should be possessed digital natives. Coding education can also be indirectly associated with the Knowledge Constructor standard of ISTE, 2016. This standard includes individuals’ presentations of information and innovative products by using digital tools. It can be argued that coding education makes it possible for users to achieve both of these standards.

Coding education is being included in more primary and secondary education curricula because coding education can make important contributions to a country’s development
The developments in coding education have positively contributed to the coding education in Turkey and, as a result, a course titled *Information Technologies and Software* has been incorporated in the curricula of Grades 5 and above since 2012-2013 academic year.

Coding education has become readily accessible for teachers in recent years. Tools such as *Scratch* (Scratch.mit.edu, 2018), *Alice* (Alice.org, 2018), *Code.org* (Code.org, 2018) and others, along with global coding activities such as *Bebras* (Bebras.org, 2018) and *the hour of code* (TheHourofCode.org, 2018), played important roles in the process. Not only has coding become available but its positive academic effects have also become more evident. For example, numerous scholars have linked coding activities to increased student motivation (Akpinar and Altun, 2014; Calder, 2010; Kelleher *et al.*, 2007; Kert and Ugras, 2009). Coding education is also connected to the development of students’ higher-order thinking skills such as analysing, assessing, relating, interpreting, problem-solving, critical thinking and algorithmic thinking (Akpinar and Altun, 2014; Kert and Ugras, 2009). A meta-analysis study examining the effect of coding indicates that coding mostly has a positive effect on problem-solving and other higher-order cognitive skills (Liao and Bright, 1991). This is because coding intensely includes mathematical and logical thinking skills.

In the rest of the introduction, we examine block-based coding languages, then, types of coding knowledge, followed by our argument for validated coding achievement tests. We also discuss the perceived shortcomings of existing coding achievement tests and articulate the rationale for our study and how it builds on prior scholarship in this area.

**Block-based coding languages**

Block-based coding languages are based on the idea that beginners should be able to master basic coding concepts more easily. This way, beginners can get used to professional coding languages more quickly. In block-based coding languages, coders do not write codes and do not need to be familiar with the confusing syntax of professional coding languages. They just drag and drop code blocks to create a programme, which is not intimidating for beginners. *Scratch* (Scratch.mit.edu, 2018), *Blockly* (Blockly-games.appspot.com, 2018) and *Snap* (Snap.berkeley.edu, 2018) are good examples of block-based coding language tools.

*Scratch* is the most well-known among these block-based languages. *Scratch* was developed by the Lifelong Kindergarten Group at the MIT Media Lab in 2007. *Scratch* is free and easy to learn. Thus, *Scratch* is an appropriate tool to teach coding to the beginners. *Scratch* is used not only by children but also by freshmen and sophomores at universities to ease the difficulties in learning coding. Therefore, it has millions of users all over the world. As of 20 February 2019, the number of registered users in the official *Scratch* website is 35,584,584 (Scratch.mit.edu, 2018). It should be noted that *Scratch* can also be used offline, without registering in the official *Scratch* website, further increasing the number of *Scratch* users. *Scratch* is widely used in Turkey, where the authors have based their work. There are 484,137 *Scratchers* in Turkey, which make up 1.44 per cent of *Scratchers* all around the world, as of 20 February 2019. *Scratch* is second only to the USA in *Scratch* usage, according to *Scratchers* worldwide map. Furthermore, nine *Scratch* days were celebrated all around Turkey in 2018, which demonstrates a strong commitment to coding education (Scratch.mit.edu, 2018). Nevertheless, it is unknown how many university students in Turkey are using *Scratch* and learning introductory programming through *Scratch*.

**Types of coding knowledge**

Because coding is a unique knowledge domain including several sub-knowledge types, scholars have developed a distinct coding knowledge classification. According to
Bayman and Mayer (1988), there are three types of knowledge acquisition during the learning of coding. These are syntactic, conceptual and strategic knowledge:

1. **Syntactic knowledge** is the knowledge containing the syntax of a coding language.
2. **Conceptual knowledge** is the knowledge containing coding concepts and principles.
3. **Strategic knowledge** is the knowledge containing problem-solving by coding.

*Repeat-Until* loop writing in the Pascal coding language can be given as an example of syntactic knowledge. This technical knowledge is required for writing the code to be run but it can be said that it is not enough to solve a problem. Conceptual knowledge is the knowledge containing the structural rules of coding and the functions of these rules for problem solving. Knowing definitions of concepts related to coding can be given as an example of conceptual knowledge. Finally, strategic knowledge can be identified as knowledge of finding solution to a problem encountered with, through coding. A problem is recognized, identified and solved with strategic knowledge. In this process, syntactic and conceptual knowledge is also used (Bayman and Mayer, 1988).

**The need for validated achievement tests**

We argue that measurement is necessary to prove if coding education is effective. A reliable and valid measurement tool is needed to meet this need. In the learning sciences, the requirement for validated achievement tests are taken for granted. However, when it comes to the field of information and computer science, this is not always the case. Few achievement tests in these fields are validated (Parker et al., 2016; Yadav et al., 2015). Therefore, we stress that validated achievement tests are needed and should be developed. Poorly constructed assessment tools might measure outcomes that teachers, administrators and stakeholders did not intend to measure or measure the intended outcome unreliably. These faulty outcomes might bring about failing to detect performance improvement caused by a novel approach or incorrectly supporting an ineffective intervention (Rhue and Zumbo, 2008; Schoenfeld, 2006). As a result of this, for example, a proficient student in a given subject might be categorized as incompetent or vice versa. In sum, we feel that such assessment tools would fill an important gap in the research literature of information and computer science pedagogy.

**Existing coding achievement tests**

There are numerous tools (also called concept inventory) in the literature to measure coding achievement in other domains, and we draw on these for inspiration. For instance, the work by Tew and Guzdial (2011) is particularly relevant. They validated the FCS1 coding achievement tool independent of coding language for introductory computer science students. However, it mostly consists of difficult questions; therefore, it may not function well as a formative assessment. Furthermore, it reports that its “full-scale reliability testing” should be done in a future work. In another study, Parker et al. (2016) extended this work and developed a 27-item and language-independent SCS1 assessment tool for introductory computer science students based on FCS1 assessment tool. Both of these tools draw on item response theory. It is worth noting that these two tools (FCS1 and SCS1) have considerably low Cronbach’s alpha reliability values indicating that their outcomes might not be stable. Furthermore, they are very difficult assessment tools in terms of item difficulty index. For instance, SCS1 has 22 difficult questions out of 27 with no easy questions. This makes them too challenging given that the audience is only introductory computer science students. In particular, the SCS1 has too many questions having *fair* discrimination level implying that...
Another coding achievement test based on the work by Tew and Guzdial (2011) is developed by Lee and Ko (2015). This 24-item test is validated with adult participants. Unfortunately, the reliability level of the test is not reported and there are questions regarding the construct validity of the test. There are several other concept inventories developed in English language in the related literature, which are digital logic concept inventory (Herman et al., 2010), computer science concept inventory for introductory programming (Caceffo et al., 2016) and Delphi concept inventory (Veerasmay et al., 2016).

Because preferred coding language is an important factor in related achievement tests, we also examined other Scratch-based tests that have been developed. Weintrop and Wilensky (2015) developed a 28-item commutative assessment tool based on FCS1 for high school students to compare conceptual understanding in text- and block-based forms. Commutative assessment means that each question can be displayed in either a text- or block-based form. However, psychometric details of reliability and validity of the assessment tool used in the paper were not available. As for solely Scratch-based assessment, Roman-Gonzalez et al. (2017) developed a 28-item assessment tool of CT for students from fifth to tenth grades. They focus on cognitive factors underlying CT and prove the criterion validity of the CT test using tests of mental ability and problem solving. As the CT test covers some of the instructional objectives in common with this study, our item pool draws on select items from this CT test.

Because translating items in an achievement test into another language weakens validity assertion owing to cultural compatibility issues, Turkish literature was also searched for coding achievement tests. Demir (2015) developed a 31-item test for vocational high school students. Demir’s test seems to be partially language-independent focusing more on algorithmic thinking. Sayginer (2017) also developed a test for university students consisting of 18 items, 4 of which are open-ended questions, whereas the rest are multiple-choice ones. Sayginer uses the Pascal coding language syntax and English coding commands in the test. Murat Cinar (personal communication on 4 February 2018) revised Sayginer’s test for high school students by adding seven questions to it, making it language-independent, and using Turkish coding commands. In sum, we used select items of Sayginer’s test for our own item pool in this study because of the fact that it was developed for university students.

Justification and objective
Coding achievement tests that are currently in use in both English and Turkish literature have reliability and validity problems. Because these tests have been developed as independent of coding languages, they predominantly include algorithmic thinking and logic. In other words, their main purpose is to measure these thought processes rather than the coding skill in a specific coding language, failing to satisfy the purpose of the test developed and validated within the scope of this study. Thus, it is not known whether those scoring high in a platform-neutral assessment would also be able to score high in a platform-based assessment, indicating that the two are measuring different constructs. Indeed, platform-neutral assessment should be more common in K-12 because enhancing higher-order thinking skills through coding education is the main objective as for children. Rather, we argue that the prime purpose in universities is to teach how to code properly, so platform-based assessment should be taken into consideration. On the other hand, when looking at the literature on coding education, it is evident that Scratch is the most commonly used language as a coding educational tool, not the professional ones. This is because professional coding languages are perceived to be user-hostile and have complex syntax for
coding learning students (Catlak et al., 2015). Thus, it can be said that a valid and reliable Scratch-based achievement test on coding education is needed in Turkish literature. Therefore, we developed a Turkish multiple-choice coding achievement test in the Scratch coding language to test this supposition.

**Method**

In this study, classical test theory (CTT) is used to validate a coding achievement test. CTT is a collection of psychometric theory predicting results of testing such as discriminativeness and difficulty of items in the test (Novick, 1966). In this paper, information about participants, implementation process and preparation of the data for the analysis are presented below.

**Participants**

The sample for this research consists of 186 volunteer undergraduates taking four different courses at the department of Computer Education and Instructional Technology (CEIT) at Hacettepe University during the spring semester of the 2017-2018 academic year. Characteristics of the sample are given in Table I.

According to the data presented in Table I, there are no significant differences by gender and class level. Nevertheless, when it comes to Scratch experience, a significant number of the participants choose having some experience choice (68.3 per cent). Having many students with some Scratch experience as participant is desirable because it would be unfitting to validate an achievement test through a coding language that they have never seen before.

**Implementation process**

The coding achievement test was applied in an online environment via Google Forms. In this form, questions were sorted by intuitive difficulty from simple to difficult. The data-collection process was carried out in a computer laboratory with 40 computers at the department where the authors of the study are employed. It was stressed that participation was voluntary and students would not be graded for the test. All invited students chose to participate in the study. Students were given 60 min to complete the test. In the process of administration of the test, it was emphasized that the students should do their best, as the test was a part of an academic study. In addition, the students were instructed that they should not give random answers; instead, they should leave the questions “not responded” if they did not know the answer. The authors were present in the computer laboratory.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Sub-variables</th>
<th>Frequency (f)</th>
<th>(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender</td>
<td>Female</td>
<td>86</td>
<td>46.2</td>
</tr>
<tr>
<td></td>
<td>Male</td>
<td>100</td>
<td>53.8</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>186</td>
<td>100</td>
</tr>
<tr>
<td>Class level</td>
<td>Freshman</td>
<td>49</td>
<td>26.3</td>
</tr>
<tr>
<td></td>
<td>Sophomore</td>
<td>33</td>
<td>17.7</td>
</tr>
<tr>
<td></td>
<td>Junior</td>
<td>63</td>
<td>33.9</td>
</tr>
<tr>
<td></td>
<td>Senior</td>
<td>41</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>186</td>
<td>100</td>
</tr>
<tr>
<td>Scratch experience level</td>
<td>Have no experience</td>
<td>39</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Have some experience</td>
<td>127</td>
<td>68.3</td>
</tr>
<tr>
<td></td>
<td>Have quite/much experience</td>
<td>20</td>
<td>10.7</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>186</td>
<td>100</td>
</tr>
</tbody>
</table>

Table I. Characteristics of the participants
throughout the implementation process so as to prevent participants from searching the Internet for the correct answers of questions that they find difficult in the test and other likely undesirable situations. The authors of this paper are faculty staff of the department where all the participants are students. In the academic term during which the data were collected, only senior students took a course that authors of the paper taught.

Preparation of the data for the analysis
SPSS 17 and MS Excel 2016 programmes were used in the analysis of the data. First of all, data of the research were examined in terms of random answers and unanswered questions. As a result of this examination, three participants out of 189 left more than half of the questions unanswered. As these participants did not most probably make necessary effort for the test, the data of these three participants were excluded from the sample. Then, one point was given for each question they answered correctly. No point was given for the questions they answered incorrectly or left unanswered. Participants could have maximum 18 points and minimum zero point in the test. In this process, wrong answers did not decrease from the total point received. After scoring process of the questions was completed, total scores of the participants were calculated. The participants were sorted by the number of correct answers, from the highest achiever to the lowest achiever. Subsequent to preparing data for analysis, validity of the test was discussed below.

Findings
Findings of this study are presented in three sections: content validity, construct validity and a short paragraph of reliability.

Validity
Validity is a vital property of achievement tests. Validity might be defined as the extent to which all amassed evidence corroborates the purported purpose of the test (AERA-APA-NCME, 2014). Three types of validity are traditionally mentioned in achievement tests. These are content, construct and criterion (also called concurrent) validity (Messick, 1989). Proving the criterion validity of the coding achievement test is outside the scope of this study.

Content validity
Content validity of a test refers to the content-representativeness of all items included in assessment tools (Martella et al., 1999). It is basically the extent to which a test measures what it purports to measure. Therefore, content validity is the starting point for the development of achievement tests. Statistical calculations are generally not used for determining content validity (Allen and Yen, 2002). It was expressed that it is important to prepare instructional objectives, create a table of specifications and ask expert opinion for test items to ensure content validity (Guler, 2012). In this way, it can be asserted that the test really measures coding achievement.

Writing of instructional objectives
Instructional objectives were determined in the first stage of the coding achievement test to be developed. In this context, 11 instructional objectives were determined in different levels of Bloom’s (1956) revised cognitive taxonomy (Anderson et al., 2001). Instructional objectives were checked by two experts in terms of instructional objective writing rules. One of the experts is a doctoral student in the field of curriculum and teaching, and the other one
is a professor in the field of *computer education and instructional technology*. These experts expressed that instructional objectives were largely incomprehensible and a bit general. Therefore, instructional objectives were rewritten in accordance with feedback taken. The number of items measuring instructional objectives and the classification of objectives according to Bloom’s revised taxonomy are presented in Table II.

It should be noted in Table II that an instructional objective in the coding achievement test can be measured with 11 items (fourth objective) and also with three items (for instance seventh objective). The reason of measurement of some instructional objectives with a large number of items is that these objectives are the basic ones and therefore it is very difficult to write questions measuring other objectives without referring to these objectives. In addition to this, instructional objectives are classified by three experts according to Bloom’s revised taxonomy (Anderson et al., 2001) in Table II. If more than one expert picks the same level, corresponding instructional objective is placed in that level. As a result of this classification, it is seen that eight objectives measure low-level knowledge (remembering = 1, understanding = 2, applying = 5), whereas three objectives measure high-level knowledge (analysing = 2, creating = 1). In accordance with that, it can be said that the coding achievement test measures low-level knowledge in general. However, there are five objectives at the *applying* level, which is regarded as the highest of the low-level knowledge.

**Creation of the item pool**

In the process of creation of the item pool for the coding achievement test developed, the researchers benefited from several similar studies in the related literature (M Cinar 2018, personal communication, 4 February; Roman-Gonzalez et al., 2017; Sayginer, 2017). Coding achievement tests (M Cinar 2018, personal communication, 4 February; Sayginer, 2017) and CT test (Roman-Gonzalez et al., 2017) were reached in addition to others mentioned in the

<table>
<thead>
<tr>
<th>Instructional objectives</th>
<th>The number of items</th>
<th>Experts’ decision</th>
<th>Final decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Scratch coding language, undergraduate students</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Define basic coding concepts and processes</td>
<td>6</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>2. Include variables in the code she/he writes</td>
<td>9</td>
<td>AP</td>
<td>AP</td>
</tr>
<tr>
<td>3. Write a code receiving an input from user and giving an output to the user</td>
<td>7</td>
<td>AP</td>
<td>U</td>
</tr>
<tr>
<td>4. Write a code containing arithmetic and logical operators</td>
<td>11</td>
<td>AP</td>
<td>A</td>
</tr>
<tr>
<td>5. Write a code containing conditional statements</td>
<td>8</td>
<td>U</td>
<td>C</td>
</tr>
<tr>
<td>6. Write a code containing loops</td>
<td>8</td>
<td>U</td>
<td>C</td>
</tr>
<tr>
<td>7. Edit appearance of objects by using codes</td>
<td>3</td>
<td>A</td>
<td>AP</td>
</tr>
<tr>
<td>8. Edit location and movement of objects by using codes</td>
<td>6</td>
<td>A</td>
<td>AP</td>
</tr>
<tr>
<td>9. Draw with objects by using codes</td>
<td>3</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>10. Place advanced ready-to-use code blocks</td>
<td>3</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>11. Place advanced ready-to-use code blocks</td>
<td>4</td>
<td>E</td>
<td>C</td>
</tr>
</tbody>
</table>

**Notes:** R = remembering (one objective); U = understanding (two objectives); AP = applying (five objectives); A = analysing (two objectives); E = evaluating (zero objective); C = creating (one objective); Number of items measuring corresponding instructional objective in the test

| Table II. Number of items measuring instructional objectives and classification of the objectives according to Bloom’s revised taxonomy |
related works section of this study. Table III shows the sources of the items of the coding achievement test.

As seen in Table III, 11 items added to the item pool in the first stage were inspired from Roman-Gonzalez et al. (2017) (1 item), personal communication with Murat Cinar on 4 February 2018 (1 item) and Sayginer (2017) (9 items). These items were transformed into the Scratch language, as they were created in an algorithmic structure by considering different coding languages or independently from coding languages. Then, 20 more items were written by the researchers, considering the coverage of instructional objectives and types of coding knowledge. Thus, the total number of items in the item pool reached to 31. Items were mainly inspired by codes of prevalent Scratch games freely available on the Scratch website. Distractors were formed based on the common misconceptions among introductory coding learners.

**Expert opinions**

One way of ensuring content validity is to take opinions of experts in the field (Popham, 2000). The experts studying in the related field can be consulted to determine the content-representativeness of a test by taking views and comments from them. That is, it makes sure that the test measures the characteristics to be measured adequately and appropriately (Seker and Gencdogan, 2006). Therefore, after the item pool was created, the 31 items were sent to seven experts to ensure content validity of the test. These experts were asked whether the test measured coding achievement. They were also asked to identify any possible logical, conceptual, syntactic and measurement errors in the questions. Characteristics of the experts are given in Table IV.

As it can be seen in Table IV, most of the experts are from Computer Education and Instructional Technology field and recently passed PhD qualification exam. As shown in the last column, a different number of experts were consulted for different purposes. For instance, seven experts were consulted for content validity of the test, whereas three experts were consulted for checking online version of it. Necessary corrections were applied to the test in line with feedback from these experts. For example, experts expressed that some questions had more than one correct answer and some questions did not have any correct answers in some cases. Online version of the coding achievement test published at Google Form was sent to five experts more for enhancing of clarity, grammar and appearance. Required corrections were also applied in accordance with feedback received from these experts. For instance, one of the experts suggested to equalize the length of choices of test items and to create choices as either text only or image only.

<table>
<thead>
<tr>
<th>Item number</th>
<th>Similar item number in the original test</th>
<th>Sources the of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>First stage (31 items)</td>
<td>Second stage (18 items)</td>
<td>Sources the of items</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>24</td>
<td>17</td>
<td>9</td>
</tr>
</tbody>
</table>

**Table III. Sources of items in the test**

Note: E = item excluded from the test
Test items were also classified according to Bayman and Mayer’s (1988) types of coding knowledge. For this purpose, the test was sent to three more experts and these experts were asked to classify the test items for their level to measure syntactic, conceptual and strategic knowledge. Classification of the coding achievement test according to Bayman and Mayer’s (1988) types of coding knowledge are given in Table V.

The item pool of the coding achievement test contains seven questions measuring syntactic knowledge, six questions measuring conceptual knowledge and 18 questions measuring strategic knowledge (Table V). The majority of questions in the item pool measure strategic knowledge. As also shown in the same table, 18-item version of the coding achievement test includes four questions measuring syntactic coding knowledge, five questions measuring conceptual coding knowledge and nine questions measuring strategic coding knowledge. Accordingly, it was decided that the coding achievement test measures strategic knowledge at most out of the types of coding knowledge, after that it measures conceptual and syntactic knowledge, respectively. Considering the significance of the strategic coding knowledge, it can be said that this is what was already wanted. As a result, the test covers all three types of coding knowledge; therefore, content validity of the coding achievement test is ensured.

**Construct validity**

Construct validity is a key factor for validity of tests. Unlike content validity, it draws on the outcomes of tests so that the actual impact of tests on students can be observed. According to Martella et al. (1999), construct validity refers to the attitudes, abilities or characteristics of individuals which cannot be directly seen, yet are implied on their observable behaviours. Item analysis and exploratory factor analysis EFA should be carried out to show construct validity of a test (Turgut, 1992).

**Item analysis**

Item difficulty index and item discrimination index are calculated in item analyses of a test. Item difficulty index shows the correct answer rate of each item. Its value can range between zero and one. When the item difficulty index is close to zero, it means that the item is difficult, and when the index is close to one, it means the item is easy. Index around 0.5 means average item difficulty (Thorndike et al., 1991).
Item discrimination index is the extent of an item to discriminate the most successful one-fourth test takers from the least successful one-fourth. Value of item discrimination index can range between negative one and positive one. When value of index is close to zero, it means that item discrimination is low and when it is close to positive one, it means that item discrimination is high (McCowan and McCowan, 1999). It is worth mentioning that the index close to negative one is a definitely undesirable situation because negative item discrimination index means that more low-achievers answer the corresponding item correctly than high-achievers do (Ebel and Frisbie, 1986). Item analysis results of the coding achievement test developed within the scope of this study are shown in Table VI.

In Table VI, \( C_{high} \) column shows the number of correct answers given by the highest-achieving quarter (27 per cent to be exact) among participants, whereas \( C_{low} \) column shows that of the lowest-achieving quarter. When Table VI is further examined, average item difficulty index of 31-item version of the coding achievement test is 0.399. Accordingly, 31-item version of the test consists of difficult questions. Average item discrimination index of 31-item version of the test was calculated as 0.360. Based on this value, 31-item version of

<table>
<thead>
<tr>
<th>First stage (31 items)</th>
<th>Second stage (18 items)</th>
<th>Experts’ decision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Expert 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>CON</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>CON</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>CON</td>
</tr>
<tr>
<td>4</td>
<td>Excluded</td>
<td>CON</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>CON</td>
</tr>
<tr>
<td>6</td>
<td>*5</td>
<td>CON</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>STR</td>
</tr>
<tr>
<td>8</td>
<td>Excluded</td>
<td>SYN</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>SYN</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>SYN</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>SYN</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>STR</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>SYN</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>SYN</td>
</tr>
<tr>
<td>15</td>
<td>*13</td>
<td>SYN</td>
</tr>
<tr>
<td>16</td>
<td>Excluded</td>
<td>STR</td>
</tr>
<tr>
<td>17</td>
<td>Excluded</td>
<td>STR</td>
</tr>
<tr>
<td>18</td>
<td>Excluded</td>
<td>STR</td>
</tr>
<tr>
<td>19</td>
<td>14</td>
<td>STR</td>
</tr>
<tr>
<td>20</td>
<td>15</td>
<td>STR</td>
</tr>
<tr>
<td>21</td>
<td>Excluded</td>
<td>SYN</td>
</tr>
<tr>
<td>22</td>
<td>*16</td>
<td>STR</td>
</tr>
<tr>
<td>23</td>
<td>Excluded</td>
<td>STR</td>
</tr>
<tr>
<td>24</td>
<td>17</td>
<td>STR</td>
</tr>
<tr>
<td>25</td>
<td>Excluded</td>
<td>STR</td>
</tr>
<tr>
<td>26</td>
<td>18</td>
<td>STR</td>
</tr>
<tr>
<td>27</td>
<td>Excluded</td>
<td>SYN</td>
</tr>
<tr>
<td>28</td>
<td>Excluded</td>
<td>SYN</td>
</tr>
<tr>
<td>29</td>
<td>Excluded</td>
<td>SYN</td>
</tr>
<tr>
<td>30</td>
<td>Excluded</td>
<td>SYN</td>
</tr>
<tr>
<td>31</td>
<td>Excluded</td>
<td>SYN</td>
</tr>
</tbody>
</table>

Table V. Classification of the items according to the types of coding knowledge

Notes: *See Appendix for these sample items; SYN = syntactic knowledge; CON = conceptual knowledge; STR = strategic knowledge
the test consists of highly discriminative questions. Some poor items were excluded from the test based on the unsatisfactory item difficulty and discrimination indexes relative to each other to obtain a medium-difficult test and further increase discriminativeness of the test. That is to say, items whose difficulty index is not between 0.200 and 0.825 and whose item discrimination index is lower than 0.30 were excluded. In this context, according to these criteria 13 items in total (specifically items 4, 8, 16, 17, 18, 21, 23, 25, 27, 28, 29, 30 and 31) were excluded from the test. The reason why question seven was not excluded, although its item statistics did not meet criteria mentioned above, is that this question is important in terms of the content validity of the test. Indeed, question seven was the unique item whose only aim is to measure the skill of defining, initializing variables and assigning value to them.

Item analyses were carried out again with the same sample after the aforementioned items were excluded from the test. As a result of the excluded items, it was observed that

### Table VI. Analysis of item difficulty and item discrimination index

<table>
<thead>
<tr>
<th>Item no.</th>
<th>*C&lt;sub&gt;high&lt;/sub&gt;</th>
<th>*C&lt;sub&gt;low&lt;/sub&gt;</th>
<th>**p</th>
<th>**d</th>
<th>Item no.</th>
<th>C&lt;sub&gt;high&lt;/sub&gt;</th>
<th>C&lt;sub&gt;low&lt;/sub&gt;</th>
<th>p</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>47</td>
<td>31</td>
<td>0.806</td>
<td>0.32</td>
<td>1</td>
<td>48</td>
<td>28</td>
<td>0.806</td>
<td>0.40</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>20</td>
<td>0.677</td>
<td>0.52</td>
<td>2</td>
<td>47</td>
<td>13</td>
<td>0.677</td>
<td>0.68</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
<td>24</td>
<td>0.790</td>
<td>0.44</td>
<td>3</td>
<td>46</td>
<td>27</td>
<td>0.790</td>
<td>0.38</td>
</tr>
<tr>
<td>4</td>
<td>49</td>
<td>34</td>
<td>0.844</td>
<td>0.30</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>39</td>
<td>20</td>
<td>0.645</td>
<td>0.38</td>
<td>4</td>
<td>41</td>
<td>14</td>
<td>0.645</td>
<td>0.54</td>
</tr>
<tr>
<td>6</td>
<td>47</td>
<td>21</td>
<td>0.799</td>
<td>0.52</td>
<td>5</td>
<td>50</td>
<td>18</td>
<td>0.769</td>
<td>0.64</td>
</tr>
<tr>
<td>7</td>
<td>47</td>
<td>34</td>
<td>0.876</td>
<td>0.26</td>
<td>6</td>
<td>48</td>
<td>33</td>
<td>0.876</td>
<td>0.30</td>
</tr>
<tr>
<td>8</td>
<td>39</td>
<td>25</td>
<td>0.661</td>
<td>0.28</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>41</td>
<td>16</td>
<td>0.591</td>
<td>0.50</td>
<td>7</td>
<td>41</td>
<td>10</td>
<td>0.591</td>
<td>0.62</td>
</tr>
<tr>
<td>10</td>
<td>29</td>
<td>5</td>
<td>0.317</td>
<td>0.48</td>
<td>8</td>
<td>30</td>
<td>5</td>
<td>0.317</td>
<td>0.50</td>
</tr>
<tr>
<td>11</td>
<td>46</td>
<td>16</td>
<td>0.677</td>
<td>0.60</td>
<td>9</td>
<td>46</td>
<td>17</td>
<td>0.677</td>
<td>0.58</td>
</tr>
<tr>
<td>12</td>
<td>48</td>
<td>15</td>
<td>0.677</td>
<td>0.66</td>
<td>10</td>
<td>48</td>
<td>16</td>
<td>0.677</td>
<td>0.64</td>
</tr>
<tr>
<td>13</td>
<td>49</td>
<td>20</td>
<td>0.774</td>
<td>0.58</td>
<td>11</td>
<td>49</td>
<td>23</td>
<td>0.774</td>
<td>0.52</td>
</tr>
<tr>
<td>14</td>
<td>37</td>
<td>7</td>
<td>0.392</td>
<td>0.60</td>
<td>12</td>
<td>37</td>
<td>4</td>
<td>0.392</td>
<td>0.66</td>
</tr>
<tr>
<td>15</td>
<td>31</td>
<td>7</td>
<td>0.285</td>
<td>0.48</td>
<td>13</td>
<td>33</td>
<td>5</td>
<td>0.285</td>
<td>0.56</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>6</td>
<td>0.167</td>
<td>0.14</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>1</td>
<td>0.091</td>
<td>0.14</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>9</td>
<td>3</td>
<td>0.102</td>
<td>0.12</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>32</td>
<td>2</td>
<td>0.247</td>
<td>0.60</td>
<td>14</td>
<td>32</td>
<td>4</td>
<td>0.247</td>
<td>0.56</td>
</tr>
<tr>
<td>20</td>
<td>31</td>
<td>2</td>
<td>0.280</td>
<td>0.58</td>
<td>15</td>
<td>32</td>
<td>2</td>
<td>0.280</td>
<td>0.60</td>
</tr>
<tr>
<td>21</td>
<td>11</td>
<td>3</td>
<td>0.086</td>
<td>0.16</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>26</td>
<td>8</td>
<td>0.226</td>
<td>0.36</td>
<td>16</td>
<td>24</td>
<td>7</td>
<td>0.226</td>
<td>0.34</td>
</tr>
<tr>
<td>23</td>
<td>13</td>
<td>2</td>
<td>0.124</td>
<td>0.22</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>40</td>
<td>12</td>
<td>0.489</td>
<td>0.56</td>
<td>17</td>
<td>42</td>
<td>13</td>
<td>0.489</td>
<td>0.58</td>
</tr>
<tr>
<td>25</td>
<td>4</td>
<td>0</td>
<td>0.043</td>
<td>0.08</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>24</td>
<td>0</td>
<td>0.210</td>
<td>0.48</td>
<td>18</td>
<td>24</td>
<td>1</td>
<td>0.210</td>
<td>0.46</td>
</tr>
<tr>
<td>27</td>
<td>19</td>
<td>2</td>
<td>0.145</td>
<td>0.34</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>8</td>
<td>2</td>
<td>0.113</td>
<td>0.12</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>7</td>
<td>2</td>
<td>0.102</td>
<td>0.10</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>1</td>
<td>0.038</td>
<td>0.08</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>12</td>
<td>4</td>
<td>0.113</td>
<td>0.16</td>
<td></td>
<td>Excluded</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average: 29.129 11.129 0.399 0.360  Average: 39.889 13.333 0.541 0.531

Notes: The items excluded from the last version of the test are shown in italic; *C<sub>high</sub> = C<sub>low</sub> = 50; C<sub>high</sub> = the number of correct answers in high group; C<sub>low</sub> = the number of correct answers in low group; **p = item difficulty index; d = item discrimination index

Scratch-based coding
item statistics of the 18-item version of the test improved substantially. The average item discrimination index of the 18-item version of the test was calculated as 0.531. According to this result, it was understood that the test was highly discriminative. Table VII shows how the item discrimination index should be interpreted (Tekin, 2003) and how many questions take part in each level.

Nine items that should be excluded from the test draw attention in the 31-item version of the test (Table VII). After the exclusion, the fact that there are 15 items with very good level by item discrimination index in 18-item version of the test is another finding that draws attention. On the other hand, average item difficulty index of the coding achievement test rose from 0.399 to 0.541. Thompson and Levitov (1985) pointed out that optimal difficulty level of five-option tests should be around 0.60. Because the average item difficulty index of the coding achievement test was close to 0.6, it was concluded that the test was in the middle-difficulty level.

The average of low 27 per cent group and that of high 27 per cent group were also compared to support construct validity of the test by showing that it discriminated the ones who did know coding and the ones who did not know coding; in other words, by showing that it was discriminative. For this, the data were sorted from the ones who answered the highest number of questions correctly to the ones who answered the lowest number of questions correctly. The number of participants in both high and low 27 per cent groups was 50. First of all, it was checked whether the data showed a normal distribution or not. It was seen, as a result of Kolmogorov–Smirnov test of normality (Massey, 1951), that the data did not show a normal distribution. For this reason, Mann-Whitney U test (Mann and Whitney, 1947) was consulted. As a result of the test, it was noted that high group has a higher average than low group in a statistically significant way ($U = 0.000; p = 0.000$). This constitutes a support for discriminativeness and the construct validity of the test.

**Exploratory factor analysis (EFA)**

EFA is consulted frequently to ensure construct validity of a measurement tool. EFA is a statistical technique, which turns a large number of interrelated items into a smaller number of significant factors. Through EFA, it can be determined what the questions measure (Dooley, 1995). Before starting EFA, two conditions are usually checked. These are Kaiser–Meyer–Olkin (KMO) test value (Kaiser, 1974) and the results of Bartlett’s Sphericity Test (Snedecor and Cochran, 1989). KMO test enables us to understand the adequacy of the sample size. A high KMO value indicates that factors can easily be extracted from the data set. As a rule, KMO value is expected to be above 0.5 (Kalayci, 2010). In this study, as the KMO value was calculated as 0.766, it can be said that the sample size is adequate.

On the other hand, Bartlett’s sphericity test is used to understand whether the data come from a normal distribution. This test should give a statistically significant value (Can, 2014). In this study, as Bartlett’s sphericity test gave a significant value ($p = 0.000; p < 0.05$), it was reported that the sample comes from a normal distribution.

<table>
<thead>
<tr>
<th>Interval</th>
<th>Item discrimination index</th>
<th>Comment</th>
<th>Frequency (f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.40 and above</td>
<td>Very good</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>Between 0.30 and 0.39</td>
<td>Pretty good</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Between 0.20 and 0.29</td>
<td>Should be reviewed</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0.19 and below</td>
<td>Should be excluded from the test</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table VII.**

Classification of the items by item discrimination index

394
As a result of EFA, there are different criteria to determine the number of factors of the measurement tool. For example, the eigenvalue of a factor should be above one, it should give more than 5 per cent variance explained and the point where the slope of scree plot flattens should be identified. The number of factors can also be decided by researchers (Kalayci, 2010). In general, the percentage of variance explained by a measurement tool is expected to be higher than the percentage of variance that cannot be explained (Secer, 2013).

Principal component analysis was used as a factor extraction method in EFA process. The first factor that explains the most variance among variables is extracted in principal component analysis. After that, the second factor is extracted to explain the remaining most variance. This situation continues in this way (Kalayci, 2010). Table VIII shows eigenvalue, variance explained and cumulative variance explained of the coding achievement test.

In Table VIII, there are six factors whose eigenvalue exceeds 1, and there are seven factors whose explained variance is over 5 per cent. However, scree plot should also be examined to determine the number of factors (Cokluk et al., 2010). Scree plot is given in Figure 1.

When the scree plot in Figure 1 is examined, it is evident that slope flattens after second factor. This flattening in the slope means that contribution of the factors to variance explained decreased after second factor (Cokluk et al., 2010). In this case, it is seen that the test consists of two factors. These two factors explain 33.783 per cent of the total variance. Though this percentage may seem low, it is difficult to reach a high-level explained variance in social sciences where information is often less precise (Hair et al., 1998).

After the number of factors of the coding achievement test was determined, the items that created these factors were determined. For this purpose, factor loads of items were examined. Varimax transformation method was used at this point for factors to become evident. The purpose of factor transformation is to obtain factors that can be named and interpreted more easily (Kalayci, 2010). Factor loads of test items and the distribution of these items according to factors are given in Table IX.

According to data presented in Table IX, whereas the highest factor load in the test is 0.803, the lowest factor load is obtained as 0.000. Factor loads are expected to be greater than

<table>
<thead>
<tr>
<th>Factors</th>
<th>Eigenvalue</th>
<th>Variance explained</th>
<th>Cumulative variance explained</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.214</td>
<td>23.411</td>
<td>23.411</td>
</tr>
<tr>
<td>2</td>
<td>1.867</td>
<td>10.372</td>
<td>33.783</td>
</tr>
<tr>
<td>3</td>
<td>1.208</td>
<td>6.711</td>
<td>40.494</td>
</tr>
<tr>
<td>4</td>
<td>1.206</td>
<td>6.697</td>
<td>47.191</td>
</tr>
<tr>
<td>5</td>
<td>1.095</td>
<td>6.086</td>
<td>53.277</td>
</tr>
<tr>
<td>6</td>
<td>1.032</td>
<td>5.736</td>
<td>59.013</td>
</tr>
<tr>
<td>7</td>
<td>0.923</td>
<td>5.129</td>
<td>64.142</td>
</tr>
<tr>
<td>8</td>
<td>0.871</td>
<td>4.836</td>
<td>68.978</td>
</tr>
<tr>
<td>9</td>
<td>0.810</td>
<td>4.500</td>
<td>73.479</td>
</tr>
<tr>
<td>10</td>
<td>0.772</td>
<td>4.288</td>
<td>77.767</td>
</tr>
<tr>
<td>11</td>
<td>0.714</td>
<td>3.966</td>
<td>81.733</td>
</tr>
<tr>
<td>12</td>
<td>0.640</td>
<td>3.558</td>
<td>85.291</td>
</tr>
<tr>
<td>13</td>
<td>0.554</td>
<td>3.075</td>
<td>88.366</td>
</tr>
<tr>
<td>14</td>
<td>0.532</td>
<td>2.954</td>
<td>91.320</td>
</tr>
</tbody>
</table>

Table VIII. Eigenvalue, variance explained and cumulative variance explained of the test.
0.30 (Hair et al., 1998). If an item takes a greater factor load as absolute value under a certain factor, it means that the item is more closely related to that factor. This further means that the items having high factor loads under a certain factor are somehow measuring a similar construct; therefore, they should form a factor together. The coding achievement test is composed of two factors. Each factor of the coding achievement test consists of nine items. The first factor consists of items 1, 2, 3, 4, 5, 6, 7, 9 and 11, whereas the second factor consists of items 8, 10, 12, 13, 14, 15, 16, 17 and 18. Items included in these factors were examined by researchers to be able to name them. After the examination, the first factor was named as basic coding skill and the second was named as advanced coding skill. Consequently, it is seen that construct validity of the coding achievement test was ensured.

**Reliability**

Reliability is an indication of how much a measurement tool is free of random errors (Nunnally, 1978). Namely, it is the degree to which measurement scores would be consistent over multiple testing (Popham, 2014). There are different methods that can be used to prove the reliability of a test. KR-20, KR-21 and Cronbach’s alpha coefficients are three of the most commonly used methods (Rivera, 2007). KR-20 and KR-21 are used in dichotomous data in which the correct answer is given one point and the wrong answer is given zero point. Cronbach’s alpha reliability coefficient built on KR-20 can also be used in dichotomous data (Atilgan, 2013). Within the scope of this study, Cronbach’s alpha reliability coefficient was benefited. Reliability coefficient is a number changing between zero and one. Reliability of the test increases, as this coefficient gets closer to one (Popham, 2014). Cronbach’s alpha reliability coefficient is interpreted as reliable if it is higher than 0.7 (Nunnally, 1978). In this test, standardized Cronbach’s alpha coefficient was obtained as 0.801. Therefore, it can be
said that the coding achievement test is quite reliable and it measures coding achievement in a consistent manner.

**Discussion**

Coding education is increasingly popular and is becoming more common at various educational levels. We argue that it is important to conduct measurement of coding skill to determine the efficacy of formal education in this domain. There is need for a validated data collection tool to inform this measurement. Tools that lack rigorous validation may poorly categorise/mark students by achievement level and therefore misinform decision-makers about the effectiveness of an intervention, which could squander precious resources and hamper educational gains (Rhue and Zumbo, 2008; Tew, 2010). Unfortunately, few tools are psychometrically validated in the field of information and computer science (Parker et al., 2016). Therefore, we developed and validated a Scratch-based coding achievement test in this study.

In summary, content validity of the test was given. When the classification of instructional objectives of the coding achievement test was examined according to Bloom’s (1956) revised taxonomy (Anderson et al., 2001), it was shown that 8 out of 11 objectives measured low-level knowledge and 3 measured high-level knowledge. According to this result, it can be said that the coding achievement test generally measures low-level knowledge. However, the fact that five out of eight low-level objectives take place in applying stage, the highest stage of the low-level stages, is worth consideration. In addition, when the questions in 18-item version of the coding achievement test were classified according to Bayman and Mayer’s (1988) types of coding knowledge, it was seen that there were four syntactic, five conceptual and nine strategic questions. Accordingly, the coding achievement test mostly measures strategic knowledge. As a result, content validity of the coding achievement test is ensured. In addition to content validity, a summary of construct validity of the coding achievement test was given in Table X.

<table>
<thead>
<tr>
<th>Items</th>
<th>Factor 1 (basic coding skill)</th>
<th>Factor 2 (advanced coding skill)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.507</td>
<td>0.031</td>
</tr>
<tr>
<td>2</td>
<td>0.803</td>
<td>0.000</td>
</tr>
<tr>
<td>3</td>
<td>0.389</td>
<td>0.164</td>
</tr>
<tr>
<td>4</td>
<td>0.738</td>
<td>−0.124</td>
</tr>
<tr>
<td>5</td>
<td>0.725</td>
<td>0.125</td>
</tr>
<tr>
<td>6</td>
<td>0.360</td>
<td>0.171</td>
</tr>
<tr>
<td>7</td>
<td>0.466</td>
<td>0.231</td>
</tr>
<tr>
<td>8</td>
<td>0.253</td>
<td>0.338</td>
</tr>
<tr>
<td>9</td>
<td>0.412</td>
<td>0.289</td>
</tr>
<tr>
<td>10</td>
<td>0.326</td>
<td>0.517</td>
</tr>
<tr>
<td>11</td>
<td>0.451</td>
<td>0.379</td>
</tr>
<tr>
<td>12</td>
<td>0.300</td>
<td>0.466</td>
</tr>
<tr>
<td>13</td>
<td>0.103</td>
<td>0.502</td>
</tr>
<tr>
<td>14</td>
<td>0.169</td>
<td>0.570</td>
</tr>
<tr>
<td>15</td>
<td>0.142</td>
<td>0.582</td>
</tr>
<tr>
<td>16</td>
<td>−0.055</td>
<td>0.564</td>
</tr>
<tr>
<td>17</td>
<td>−0.022</td>
<td>0.610</td>
</tr>
<tr>
<td>18</td>
<td>0.083</td>
<td>0.580</td>
</tr>
</tbody>
</table>

**Table IX.** Factor loads of the items and distribution of the items by factor
When the data in Table X are examined, it is seen that average item difficulty index of the 18-item coding achievement test is 0.541, and the average item discrimination index of it is 0.531. EFA results showed that the coding achievement test was in two-factor structure, and the coding achievement test explains 33.783% of the total variance. Factors in the test were named as basic coding skills and advanced coding skills. According to these results, construct validity of the coding achievement test meets expectations. The Cronbach’s alpha reliability coefficient of the test was calculated as 0.801, which is sufficiently high according to Nunnally (1978).

**Limitations**

There are six limitations of this study. First, the sample of study seems to be small and is homogeneous in terms of department of participants. This paper calls for further studies with larger samples and participants from different departments to cement the validity of Scratch-based coding achievement test. Second, the language of the Scratch was used as English in the test assuming that university level students must have a proficient level of English that is required to understand Scratch code blocks. Scratch is an introductory coding language and its main aim is to help young students get used to professional coding languages more easily. English coding commands are very prevalent in professional coding languages, so it was thought that using English as language of Scratch would help students ease the process of transition between block-based coding languages and professional ones. However, it was observed that some students looked up some words. Therefore, students’ English level might have affected their test scores. Third, because participants are university students, authors decided to administer a crowded item pool to be able to more easily exclude items from the final version in case psychometric problems occur with items. Nevertheless, some students started to show signs of testing fatigue after question 25, which might have diminished the quality of their effort. This might further give rise to exclusion of questions after question 25 from the final version of the test because of too high item difficulty index or the future use of randomized question delivery. In fact, many questions in the test include stories to make questions more related to real life, but they also make them more time-consuming to understand and solve. Further, questions towards the end of the test were intuitively more difficult. Fourth, Scratch-based coding achievement test was only applied in a context where students were explicitly informed that they would not be graded based on the results of the test. This might have affected their efforts exerted in the test. Fifth, as authors of this paper could not find any other Scratch-based coding achievement test in Turkish

<table>
<thead>
<tr>
<th>Variables</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Turkish</td>
</tr>
<tr>
<td>The number of questions in the test</td>
<td>18</td>
</tr>
<tr>
<td>The number of people in the study group</td>
<td>186</td>
</tr>
<tr>
<td>Average</td>
<td>9.731</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>3.831</td>
</tr>
<tr>
<td>The number of factors</td>
<td>2 (nine items each)</td>
</tr>
<tr>
<td>Total variance explained</td>
<td>33.783%</td>
</tr>
<tr>
<td>Average item difficulty index</td>
<td>0.541</td>
</tr>
<tr>
<td>Average item discrimination index</td>
<td>0.531</td>
</tr>
<tr>
<td>Cronbach’s alpha reliability coefficient</td>
<td>0.801</td>
</tr>
</tbody>
</table>

Table X.
Summary of the findings for the coding achievement test
literature, corroborating the criterion validity of the test was outside the scope of this paper. Finally, the coding achievement test developed in this study mostly measures the low-level knowledge (8 out of 11) according to Bloom’s (1956) revised cognitive taxonomy (Anderson et al., 2001).

**Conclusion and implications**

This inquiry sought to develop a valid and reliable multiple-choice coding achievement test in Scratch language, which is an important step toward rigorous assessment of coding education. Through multiple analyses, we argue that the resulting test is a valid and useful approach to measurement that builds on and extends research in this area. Kane (2013) points out that measurement tools themselves cannot be validated, but rather their outcomes relative to a population (valid for whom) and a purpose (valid for what). According to this approach, the coding achievement test is valid for university students at the department of CEIT in Turkey and for measuring coding skill in Scratch language. We hope that other scholars will be able to build on this model and further refine our approach.

The coding achievement test might also be used with children, as the test was developed by using Scratch. The coding achievement test may be regarded as a suitable tool to measure coding achievement particularly at a high school level. This makes the potential audience for our testing approach millions of students. In addition, reliability and validity analysis of the test can be conducted again after transforming the questions in the coding achievement test into different coding languages. Because Messick (1989) stressed that validity was a changing property and validation was an on-going process, once an achievement test is validated, it needs to be replicated with the purpose of identifying problems of it and extending its validity to different contexts (Parker et al., 2016). Therefore, suitability of the test for students in different age groups and contexts should be explored in future research. Hence, we argue that re-validation studies showing that the test is also suitable for students in different age groups, departments, languages and cultures are needed.

Scratch is used by millions of people worldwide. Even if a fraction of these users are learning Scratch for formal-education purposes, it creates a strong case for a valid assessment tool. To fulfill this need, educators can make use of Scratch-based coding achievement test developed and validated in this study to measure students’ coding learning. This way, outcomes of different coding education approaches can be measured; thereby, informed decisions regarding the use of novel approaches to teaching coding can be made. Nevertheless, Scratch-based coding achievement test developed in this study has a few shortcomings. We hope that future academic efforts on developing a coding achievement test and corroborating already-developed coding achievement tests’ psychometric properties will concentrate on addressing limitations of this study to be able to produce a more robust achievement tool.

**References**


One sample question for each type of coding knowledge is given below.

The sample question for **Conceptual** knowledge type:

Question 5. The constructs that enable the branching out of the flow of a program according to the indicated condition or conditions are called .............

a) Conditional expression  
b) Operator  
c) Data  
d) Variable  
e) Counter  

The sample question for **Syntactic** knowledge type:

Question 13. Below is given a code block containing a single line placed within a loop

Which one of the followings is true of the code block given above?

a) The object constantly goes to x and y coordinates of the mouse pointer.  
b) The object goes to x and y coordinates of the mouse pointer only at the beginning of the program.  
c) The object goes in between the two objects called “mouse x” and “mouse y”.  
d) The object goes to the x coordinate of the “mouse x” object and y coordinate of the “mouse y” object.  
e) The mouse pointer constantly goes to the x and y coordinates of the object.

The sample question for **Strategic** knowledge type:

Question 16. In a classical ping-pong (table tennis) game, the goal is to send the ball coming towards you to across the table by touching/hitting it. The player who cannot touch/hit the ball loses the game. In the game below, the ball initially moves randomly. According to the coordinate system, “y” coordinate increases upward, while it decreases downward. In the game, the player on the left is controlled by a human being, while the player on the right is controlled by the program. A screenshot of the game is given below.
According to the information above, which one of the followings can be the code of the player on the right?

a)  

b)  

c)  

d)  

e)  

About the authors
Ömer Demir has been a Research Assistant at the Department of Computer Education and Instructional Technology (CEIT) of Faculty of Education at Hacettepe University in Ankara, Turkey, since 2012. He received a bachelor’s degree at the Department of CEIT at Boğaziçi University in Istanbul, Turkey, in 2011. He received a master’s degree at the Department of CEIT at Hacettepe University in Ankara, Turkey, in 2015. He has been a PhD student at the Department of CEIT at Hacettepe University in Ankara, Turkey, since 2015. He is professionally interested in studying Scratch-based coding.
游戏化学习、编码教育、e-learning 准备性、量表的发展和适应。

Ömer Demir 是相应的作者，可以联系：omerdemir@hacettepe.edu.tr

Süleyman Sadi Seferoğlu 是计算机教育与教育技术创新学院的教授，位于哈塞普大学安卡拉，土耳其。他在安卡拉的本科课程中获得了广播电视和教育科学学位。他于 1989 年、1993 年和 1996 年在哥伦比亚大学获得了硕士学位和博士学位。在教育中使用技术、e-learning、m-learning、技术政策、教学材料设计开发、教师教育、专业发展、数字鸿沟、网络欺凌、网络成瘾、社交媒体、儿童和媒体是他的专业兴趣。

For instructions on how to order reprints of this article, please visit our website: www.emeraldgrouppublishing.com/licensing/reprints.htm
Or contact us for further details: permissions@emeraldinsight.com
Bringing together over 1,600 eBooks, the Emerald eBook collections are a cost-effective way of instantly expanding library holdings and increasing usage through an award winning platform alongside journals.

Visit emeraldinsight.com
eCase Collection

Real world business scenarios for the classroom

An annual single subscription provides unlimited access for students and faculty to a growing digital library of over 1500 real world case studies on a wide range of industries spanning the business and management disciplines from across the globe.

Visit emeraldinsight.com

Get eCase Collections for your library
Recommend eCases to your librarian today. Find out more: emeraldpublishing.com/ecasecollections
Number 5/6
Learning to code, coding to learn: youth and computational thinking
Guest Editors: Hong Huang and Eric M. Meyers
253 Editorial advisory board
254 Guest editorial
266 Generating gameworlds with computers: the case for procedural creativity
Matthew Jason Wells and Jason Boyd
285 Unfold studio: supporting critical literacies of text and code
Chris Proctor and Paulo Blikstein
308 Teachers’ goals predict computational thinking gains in robotics
Eben B. Witherspoon and Christian D. Schunn
327 Youth perspectives on their development in a coding community
Ricarose Roque and Natalie Rusk
349 “I like computers. I hate coding”: a portrait of two teens’ experiences
Paulina Haduong
366 Supporting women’s persistence in computing and technology: a case for compulsory critical coding?
Joanna Weidler-Lewis, Wendy DuBow, Alexis Kaminsky and Tim Weston
383 Developing a Scratch-based coding achievement test
Ömer Demir and Süleyman Sadi Seferoğlu