MINDSTORMS 2.0
CHILDREN, PROGRAMMING, AND COMPUTATIONAL PARTICIPATION

Yasmin B. Kafai[1] and Quinn Burke[2]

Abstract
This conceptual paper argues that recent developments in K-12 programming education signal a new agenda in which learning to code has shifted from being a predominantly individualistic approach to one that is decidedly culturally grounded in the social creation and sharing of digital media. While this approach toward computing was long-advocated by Seymour Papert—most famously in his book Mindstorms*—this paper posits that his vision is at last coming into fruition with the plethora of coding communities and making activities that are increasingly garnering more youth into the potential of code as a unique tool to make and share digitally. These developments connect to current call for computational thinking but re-frame it as computational participation to leverage social connectivity inherent in the digital world of the 21st century. Drawing from extensive examples of our and others' research, this paper highlight four dimensions of computational participation: (1) from writing code to creating applications, (2) from composing from scratch to remixing others' work, (3) from designing tools to facilitating communities, and last, (4) from screen to tangibles. Discussion turns to how communities, schools, and educators can help broadening computational participation.*

Keywords Constructionism, programming, computational thinking, connectivity

1. Introduction

More than thirty years ago Seymour Papert's *Mindstorms* [23] presented computers to the world as more than just calculating machines but the very substance of learning for children, allowing them to achieve unprecedented levels of creative and critical self-expression in and around schools. Logo programming arrived at schools as a tangible way to introduce and apply mathematics, but beyond the immediate subject matter, the Logo-based activities also promoted the development of communities of social learners and programming for creative expression as never witnessed before. By the mid-1990s, however, schools had largely turned away from programming based upon a lack of subject-matter integration and a dearth of qualified instructors [21]. Yet there was also the wider question of purpose. With the rise of pre-assembled multimedia packages characteristic of the glossy CD-ROMs of the early

[1] University of Pennsylvania, 3700 Walnut, Philadelphia, PA 19147, kafai@upenn.edu
[2] College of Charleston, 86 Wentworth Street, Charleston, SC 29403, burkeqq@cofc.edu

1990s, who wanted to toil over syntax typos and debugging problems by creating these applications oneself? This question alone seemingly negated the need to learn programming in school, compounded by the excitement generated by the Internet. Schools started teaching students how to best surf the web rather than how to delve into it and understand how it actually works.

But this is changing. We are now witnessing a comeback of computer programming in schools and a return to Papert's initial vision of computers as tools for creative expression and social connectivity. We've seen a newfound interest in bringing back learning and teaching programming on all K-12 levels and within online communities. One of the most prominent developments in bringing programming back has been a call for computational thinking that has been defined as all "aspects of designing systems, solving problems, and understanding human behaviors" that highlight the contributions of computer science (p. 6, [33]). The term of "computational thinking" was initially used by Papert [25], commenting on the inherent interdisciplinary nature of computing and its potential to engage learners in new ways of thinking. This focus on moving beyond the machinery itself and stressing the potential of computers to help children think algorithmically is now also championed by many media theorists—most notably Douglas Rushkoff [29], who has named coding as nothing less than the new literacy of the 21st century. Indeed, the three Rs (reading, writing, and arithmetic) long associated with schooling may very well have two new Rs to join their ranks—a fourth R for the aRts [6] and a fifth R for pRogramming [13]. After all, digital technologies are embedded in the way we live, work, play, socialize, learn, and teach. We not only participate in but also make the digital world that we live in, and this requires understanding the personal, social, cultural, and tangible connections to code. These connections have always been part of constructionist learning, but for computational thinking to leverage social connectivity inherent in the digital world of the 21st century, it needs to be re-framed as computational participation. In other words, we argue that Papert's "objects-to-think-with" need to become "objects-to-share-with" [13]. In the following sections we discuss constructionist foundations and perspectives on computational thinking and the needed shift to computational participation to realize Mindstorms 2.0.

2. Constructionist Perspectives on Computational Thinking

The elements of the personal, social, cultural, and tangible connections were dimensions Papert understood as essential for genuine and meaningful learning in the constructionist fashion. In terms of the *personal* perspective, he saw youth's engagement with Logo programming as a way to facilitate the construction of knowledge structures with computers from the ground up with what he termed "appropriation" so that learners could make knowledge their own and begin to personally identify with it [26]. By designing a program or game (or, on a more granular level, its procedures, algorithms and data structures) the personal knowledge becomes public and can then be shared with others. In the process of sharing these artifacts, the *social* dimensions of Constructionist learning are planted, hearkening back to Papert's fascination with the Brazilian samba schools in which entire communities not only gather around but celebrate a shared dance, utilizing such gatherings as a means to inculcate social norms and the learning by doing later expostulated by Lave and Wenger's [18] seminal work on

communities of practice and Gee's [8] notion of affinity groups. The *cultural* dimension of Constructionist learning has focused on the politics that determine how one way of knowing is valued over others. Turkle and Papert [30] argued that the more improvisational, bricoleur style favoring concrete thought could be just as advanced as abstract thought and observed that among the sciences, computer science, prized abstract thinking. It's not just the abstract design but the actual programming artifacts that represent the learning context for youth, bringing us to this fourth dimension of the *tangible*, in which knowledge does not exists as an abstraction but as an actual product to be tinkered with, shared, and re-imagined.

The framing of learning as personal, social, cultural and tangible is key in realizing that computational thinking needs to move beyond mere problem solving. While computational thinking has been an instrumental concept in extending the potential of computing beyond operating the devices themselves, we argue it should be conceived as *computational participation* to better emphasize that programmed projects like games, stories, and art are not only *objects-to-think-with* (to use one of Seymour Papert's ideas) but also very much *objects-to-share-with* that connect us to and with others [13]. We see computational participation as solving problems with others, designing intuitive systems for and with others, and learning about the cultural and social nature of human behavior through the concepts, practices, and perspectives of computer science [2]. Having kids program applications, work in groups, and remix code will not address all of the challenges that are associated with broadening participation in computing, but it represents a crucial beginning step. The commercial toy industry is developing programming applications that allow children to design artifacts that they and others find significant. The popularity of the new maker kits and smart toys illustrates that children not only want interactive toys that light up and make sounds, but also toys that place them at the helm and let them do the interactive designing. They want to determine (program) when the lights blink and when the buzzers sound, and educators can foster this spirit in the classroom.

Furthermore, by making programming a community effort, educators can make schools into places for sharing and collaboration, both formally and informally. Whether these communities center around making and sharing graphic art, developing and debugging video games, or composing interactive multimedia montages, these online spaces offer children the potential to create their own digital content and then share these creations online with hundreds of thousands of other fellow enthusiasts. It is here within these digitally-based "maker" communities [9] that we see come to fruition Papert's original vision. Schools, in particular, need to move beyond seeing programming as an individualistic act and begin to understand it as a communal practice that reflects how students today can participate in their communities. This of course is no small step for schools, where individual achievement far outweighs group dynamic in what qualifies for academic success. Yet if the past decade and the advent of Web 2.0 have taught us anything, it is the importance of collaboration in facilitating more creative and cost-effective solutions to problems. Having a chance to participate and collaborate in communities of programming is key to learning the fundamental concepts and practices of programming, which in turn offers people an unprecedented opportunity to participate in the wider digital public.

3. From Computational Thinking to Computational Participation

If we take seriously the idea of computational participation, then learning programming has shifted with the wider cultural perceptions and practices of what it means to socialize and produce in the 21st century, resulting in better learning opportunities and, by extension, better teaching opportunities. Here we outline four pathways for computational participation.

3.1 From Code to Applications

Learning to program is about writing code, developing algorithms, data and control structures that result in functional, if not always efficient, programs. But while much planning and problem solving went into the early K-12 courses on coding with Logo and other languages, there was also prompt criticism [27] that the empirical evidence was slim in demonstrating that what students were learning actually transferred to other subjects. Others noticed the near utter lack of integration of programming with the rest of the school curriculum [22]. In its initial foray into K-12 schools, programming largely existed as a stand-alone activity in which students would participate once or twice a week. And typically, these isolated moments of coding existed apart from classrooms within the computer lab after which the students were to return to their "normal" classes back down the hallway.

Today, the question is what are you making? Rather than coding exercises for learning about algorithms and data structures, children now learn programming to create specific applications, be they video games or interactive stories. As our own research suggests, often children are not even aware they were in fact programming with Scratch until we actually tell them, be it in post-workshop interviews or while they are on-task [14]. Children are engaged by the potential to create something real and tangible that can be shared with others, converting the learning of programming—at least initially—from the study of an abstract discipline to a way of making and being in the world digitally. Starting with the application also makes coursework much more accessible to younger learners as evident with studies around learning programming through video game design [12], and it allows for better integration with other school subjects such as language arts [13], science [4], as well as music and art [28]. By leading with a particular project within a particular subject matter, be it digital stories in an English class or fraction games in a math course, programming pedagogy engages children with the potential to create "real-world" applications. And the children are not the only ones more engaged. An application-centric approach allows educators to leverage their own content-based knowledge to create lessons that reinforce traditional subject matter while also giving their students the opportunity to design software that is meaningful and authentic beyond the classroom.

3.2. From Tools to Communities

Having something meaningful and authentic to share beyond the classroom is requisite for this second shift. Happily, the past decade has seen the development of many admirable introductory programming languages that have made coding a more intuitive, personal process. Scratch, Kodu, and Alice are three of the many examples of introductory languages that have broadened the reach of code [17]. But

developers are now realizing that the tool alone is not enough. Every tool needs an audience and the opportunity to bring like-minded creators together via the Internet have reached an unprecedented level since the emergence of web 2.0. Tools like Scratch and Alice now have extensive online communities of millions of young users such that both sites are now migrating entirely online (Scratch just this past spring), tacitly highlighting the fact that the community of practice effectively has become the key tool for learning to code.

Today, the question is where are you posting? Young users—especially those entirely new to programming—are not selecting languages based upon syntax or compiling speeds but based on where their friends are at and where their own work can gain the most exposure [10]. Focusing on a community of learners builds upon essential insights from educational research that fruitful learning is not done in isolation but in conjunction with others. While early work by Webb examined some of the challenges as well as opportunities in having students program in small teams and there has been promising gains in promoting pair-programming activities among novices [7], programming in K-12 contexts is still mostly an individual activity. Designing better tools, we are learning, is not enough. The social context in which these programming tools are used and where programming artifacts are shared is equally, if not more, important. And while the industry seemingly has figured this out, schools need to recognize it.

3.3 "From Scratch" to "Remix"

In the past, not only did most programs have to be created from the ground-up or "from scratch" to illustrate programming competencies, but the expectation was that code was very much a proprietary commodity, to be built and ever-refined but certainly not freely shared. This mindset very much set the tone for early computing coursework as children were introduced to the potential of programming in terms of text-based functionality, akin to learning about fiction in a language arts course by being taught the meaning of nouns, verbs, and modifiers.

Today, the question is who are you following? Certainly this statement rings of Twitter feeds and Pinterest posts, but it also characterizes programming today where the repurposing and remixing of code has become standard practice. With nearly 25% of the 3 million project posted at the Scratch site as remixes, members use the practice as means to familiarize oneself with the software as well as to socialize and collaborate with others via their creations [20]. Of course, for schools, remix remains taboo. For while a growing number of media theorists and educators posit remix as a fundamental skill that schools need to address [11], school's traditional conception of "copying" sits directly at odds with the wider culture of remix that prevails in children's interactions outside of school. And while it is true that remixing on the most basic level requires mimics copying, we do know that selective remixing [16] can actually require a far more degree of sophistication in terms of where to modify selected coding segments. Practicing remix in classrooms also broaches the crucial social and political issues surrounding copyright and the open-source movement. Teaching children how to remix and when and how such appropriation is fair and mutually beneficial, schools have the opportunity to move beyond "Internet Safety 101" and ground children's understanding of web-based production in actual practice.

3.4 From Screens to Tangibles

The cover of the first edition of *Mindstorms* featured a girl with a large robotic turtle on the floor. The early Logo turtle was a physical object, not a screen object, programmed to move on the floor and, using a pen attached to its bottom, to draw designs and images on a piece of paper. Both the programmer and the onlooker could easily replicate the physical movement of the turtle with their own bodies. This might seem to be insignificant in terms of learning, but by being able to mimic the programmed movement of turtle, children could more easily comprehend the abstract input and execute commands. These tangible aspects of learning were ignored at the time, but today many educational researchers see cognition and even learning of programming as embodied [1].

Thus today, the question is what are you holding? Certainly this manifests itself most immediately in the ubiquitous presence of cellular devices in people's hands—devices which digital media researcher S. Craig Watkins [32] describes as the veritable "Grand Central Station" for youth under the age of 21 as these mobile devices increasingly represent the way youth connect online. But just as there is growing market for programmable toys for children as young as five years old, there has been a veritable renaissance of construction kits in recent years that have expanded far upon the original Lego Robotics kits, the LilyPad Arduino [3], and many other kits and materials suggest rich contexts for computing and can lead to computational activities that move beyond the screen into arts and humanities. This is especially important in K–12 classrooms, where early programming activities occurred exclusively in mathematics and science classes. As social sciences, humanities, and media arts evolve in the digital age, these construction kits represent the new "stuff" of teaching and learning. By broadening activities, practices, and perceptions of computing, computational participation can be ensured for all students. Now that technologies have become more widely distributed and affordable, it is time to develop and promote materials that jump the traditional gender divide that has been associated with robotics. We can no longer be content with youth online communities that simply socialize. Computational participation means connecting through making, which leads to deeper, richer, and ultimately healthier connections among youth.

4. Broadening Computational Participation

These are new directions for the design of activities, tools, and communities in K-12 educational computing efforts that are aimed at broadening participation in and perceptions of computing on a considerably larger scale than previous attempts of integration. Some might argue that bringing programming into K–12 schools will fail now as it failed thirty years ago. Those who persist in seeing programming as a skill meant for the exclusive few also usually persist in seeing it as an inherently asocial activity. We are not arguing that all the standard staples of traditional computing courses should be dismissed. The merits of writing code to learn about the nature of algorithms, and developing data structures, compilers and general architecture remain crucially important and they are not easily learned on their own. But we want to make the case that K-12 educational computing can take the road that K-12 language arts, mathematics, and science education have taken long ago and leverage with

success the insights gained from youth digital media and networked cultures in conjunction with learning theory that initially promoted the idea of children programmers against all conventional wisdom.

In terms of broadening computational participation, while there have been admirable efforts to attract a wider range of students to computing in terms of gender, race, and ethnicity [19], at this time, it is imperative to schools to broaden their underlying conception of what computers can potentially be as tools for creative expression. Schools play a crucial role here because despite the growth and promise of these online DIY youth communities, the percent of children who actually participate in these online groups still remains overwhelmingly small. As aptly pointed out of children's digital media use [10], the number of kids who move from socializing with digital media to creating new content with it to building new communities around such media grows exponentially smaller with each level. This is not to say however that pathways do not exist. But such pathways must be made more explicit, and here schools play a crucial role in broadening participation, fostering increased collaboration among youth, and perhaps most importantly, developing teachers who can foster the curiosity and perseverance within children to make and share their own unique digital content. Visitors to the majority of U.S. elementary, middle, and high schools (public or private) are unlikely to see any formal teaching related to computers as computational, creative devices. As education historian Larry Cuban [5] notes, schools persist in *teaching computers* rather than *teaching with computers*, and the results of this approach are readily apparent in the fact that "computers" at many schools denotes a class that students attend once or twice a week in a computer lab far removed from their core curricula coursework. The utter lack of broad opportunities to use computers creatively coupled with the persistent perception of computers as monitored screens have meant that only a narrow subset of children within schools are aware of computers as generative devices—and often this perception comes not from teachers but from parents and family members who already work within the computing field.

In terms of realizing the collaborative dimension of computational participation, schools' track record here has never been very good. The institutional model of schooling prizes individual achievement well above cooperative and productive group activity, and to a certain degree this institutional model is at ends with the scrappy, DIY ethic characteristic of these online spaces where youth are programming and sharing at will. Afterschool programs have consistently been the places at schools where the most interesting learning with technology has taken place over the past three decades. Whether helping students make video games in a Scratch gamers club, construct collaboratively programmable robots in a Lego Robotics club, or design interactive light-up masks with the Lilypad Arduino, afterschool clubs historically have had the latitude to allow children to work at their own pace in spaces that are more open and social than traditional classrooms [15]. Schools ought to take a note of the structure of their own peripheral technology programs and explore the potential of integrating such increased flexibility in school-day classrooms. Of course this is no easy feat—particularly in some of this country's neediest schools where assessment-driven curricula dominate teaching. Nonetheless, afterschool programs' success in getting children to collaborate more frequently and more genuinely stems from more than just flexibility. Such programs tends to focus on children creating particular *products,* and centering the learning around these products (as opposed to exams or worksheets) offers students a common means

around which to make and share, which in an of itself can serve as a rite of passage in such clubhouses [14].

Last, with the comeback of programming, the need for well-qualified educators who can take the lead in broadening participation and collaboration in their classrooms. Simply put, we need teachers that can implement computing curricula just as we need counselors that steer students into such courses. After reviewing the situation in the recent report, *Running on Empty* [31], the Computer Science Teachers Association has been instrumental in promoting the adaptation of certification standards for teachers. This cannot just be a transient corps of volunteers but will need to include professionals that develop a deep understanding of learning trajectories of students and pedagogical strategies to support learning. But more fundamentally in attracting more teachers to the potential of CS, schools need to stop perceiving computing entirely through the lens of math and science. At the introductory school level, computing education should be more about communication than computation. Connected teaching recognizes that students are motivated not by the technology but by the things that they make with the computer and the people with whom they get to share what they make. These ideas are present in Papert's vision of schooling, and as the participants in the youth-based DIY circles can attest, coding is not just about making the machine compute but also about communicating with the larger world. There are many reasons for these concerted efforts but all of them have one overarching goal: to achieve equity and diversity because computational participation cannot be achieved, if only a select few join the clubhouse. After thirty years of only marginal success in broadening participation in computing, we know that this does not happen on its own just as we know technology education in K-12 schools does not naturally evolve. It takes real pushing.

5. Conclusion

Becoming and being today's digital native involves not only browsing the web, using technology to communicate, and participating in gaming networks but also knowing how things are made, contributing through making, and understanding social and ethical ramifications—all ideas which were realized early on in *Mindstorms* [23] and *The Children's Machine* [24] While only few of us will become computer scientists that will write the code and design the systems that undergird our daily life, we all need to understand code to be able to constructively, creatively, and critically examine digital designs and decisions. All of us are users of digital technologies for functional, political, and ultimately personal purposes. On a functional level, a basic understanding of code allows for an understanding of the design and functionalities that underlie all aspects of interfaces, technologies, and systems we encounter daily. On a political level, understanding code empowers and provides everyone with resources to examine and question the design decisions that populate their screens. Finally, on a personal level, everyone needs and uses code in some ways for expressive purposes to better communicate, interact with others, and build relationships.

We need to be able to constructively, creatively, and critically examine designs and decisions that went into making them. Capturing these multiple purposes of literacy, education activist Paulo Freire coined

the saying that "reading the word is reading the world." We see reading and writing the code is as much about reading the world as it is about understanding, changing, and re-making the digital world in which we live. In *Mindstorms*, Seymour Papert envisioned the computer as a protean machine, a universal device to construct objects-to-think-with in intellectually rich, personally meaningful, culturally diverse and socially connected ways. In our vision, it is this fourth component—connectivity—that emerges as absolutely crucial, if children are to truly learn coding as a fundamental skill. In Mindstorm 2.0 objects-to-share-with represent the new imperative and new standard when it comes to learning and making with digital media. It is here, in the moment of exchange, that the computer may very well deliver on its protean promise for living and learning in a networked world.

6. Acknowledgments

The writings of this paper has been supported by a collaborative NSF award to the first author (CIS-0855868) together with Mitchel Resnick and Yochai Benkler. The views expressed are those of the author and do not represent the views of the National Science Foundation, the University of Pennsylvania or the College of Charleston.

References

[1] M. Berland and V. Lee, "Collaborative strategic board games as a site for distributed computational thinking," *International Journal of Game-Based Learning*, vol. 1, no. 2, pp. 65–81. 2011.

[2] K. Brennan and M. Resnick. "New Frameworks for Studying and Assessing the Development of Computational Thinking." Paper presented at 2012 Annual Meeting of the American Educational Research Association, Vancouver, April 2012.

[3] L. Buechley, K. Peppler, M. Eisenberg, and Y.B. Kafai (Eds.), *Textile Messages: Dispatches from the World of Electronic Textiles and Education*. New York: Peter Lang, 2013.

[4] C.C. Ching and Y.B. Kafai, "Peer pedagogy: Student collaboration and reflection in learning through design," *Teachers College Record*, vol. 110, no. 12, pp. 2601-2632, 2008.

[5] L. Cuban, *Oversold and underused: Computers in the classroom*. Cambridge, MA. Harvard University Press, 2001.

[6] J. Darby and J. Catterall, "The fourth R: The arts and learning," *Teachers College Record*, vol. 96, no. 2, pp. 299-328, 1994.

[7] J. Denner and L. Werner, "Computer programming in middle school: How pairs respond to challenges," *J. Education Computing Res.*, vol. 37, no. 2, pp. 131-150, 2007.

[8] J.P. Gee, *What Video Games have to Teach Us about Learning and Literacy*. New York: Palgrave Macmillan, 2003.

[9] Honey, M., and D. Kanter, eds., *Design, Make, Play: Growing the Next Generation of STEM Innovators*. New York: Routledge, 2013.

[10] M. Ito, S. Baumer, M. Bittanti, M., d. boyd, R. Cody, B. Herr, H.A. Horst, P.G. Lange, D. Mahendran, K. Martinez, C.J. Pascoe, D. Perkel, L. Robinson, C. Sims, and L. Tripp, *Hanging Out, Messing Around, Geeking Out: Living and Learning with New Media.* Cambridge, MA: MIT Press, 2009.

[11] H. Jenkins, K. Clinton, R., Purushotma, A. Robison, and M. Weigel, "Confronting the challenges of participation culture: Media education for the 21st century," The John D. & Catherine T. MacArthur Foundation, Chicago, IL, 2006.

[12] Y.B. Kafai, *Minds in Play: Computer Game Design as a Context for Children's Learning*. Mahwah, NJ: Lawrence Erlbaum, 1995.

[13] Y.B. Kafai and Q. Burke, *Connected Code: Why Children Need to Learn Programming*, Cambridge, MA: MIT Press, in press.

[14] Y.B. Kafai, D.A. Fields, and Q. Burke, "Entering the clubhouse: Case studies of youth programmers joining the online Scratch communities," *Journal of End User Development*, vol. 1, no. 1, pp. 21-35, 2010.
[15] Y.B. Kafai, K.A. Peppler, and R. Chapman, Eds. *The Computer Clubhouse. Creativity and Constructionism in Youth Communities*. New York: Teachers College Press 2009.
[16] Y.B. Kafai, R. Roque, D.A. Fields, Q. Burke, and A. Monroy-Hernandez, "Collaborative agency in youth online and offline creative production in Scratch," *Research and Practice in Technology Enhanced Learning*, vol. 10, no. 10, pp. 63-87, 2012.
[17] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Computing Surv*eys, vol. 37, no. 2, pp. 88-137, 2005.
[18] J. Lave and E. Wenger, *Situated Learning: Legitimate Peripheral Participation*. Cambridge, England: Cambridge University Press, 1991.
[19] J. McGrath-Cohoon and W. Asprey, Eds. *Women and Information Technology: Research on Under-representation*. Cambridge, MA: MIT Press, 2006.
[20] A.Monroy- Hernández and M. Resnick, "Empowering kids to create and share programmable media," *Interactions*, vol. 15, no.2, pp. 50-53, 2008.
[21] R. Noss and C. Hoyles, *Windows on Mathematical Meanings: Learning Cultures and Computers*. Norwell, MA: Kluwer Academic Publishers, 1996.
[22] D.B. Palumbo, "Programming language/problem-solving research: A review of relevant issues," *Review of Educational Research*, vol. 60, no. 1, pp. 65–89, 1990.
[23] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, 1980.
[24] S. Papert, *The Children's Machine: Rethinking School in the Age of the Computer*. New York: Basic Books, 1993.
[25] S. Papert, "An exploration in the space of mathematics educations." *International Journal of Computers for Mathematical Learning*, vol. 1, no. 1, pp. 95-123, 1996.
[26] S. Papert and I. Harel, *Situating Constructionsim*. Norwood, NJ: Ablex Publishing, 1991.
[27] R.D. Pea and D.M. Kurland, *Logo Programming and the Development of Planning Skills*. Center for Children & Technology. New York, NY: Bank Street College, 1983.
[28] K. Peppler, K. *New Creativity Paradigm: Arts Education in the Digital Age*. New York, NY: Peter Lang Publishers, 2014.
[29] D. Rushkoff, *Program or Be Programmed*, New York: Soft Skull Press, 2010.
[30] S. Turkle and S. Papert, "Epistemological Pluralism: Styles and Voices within the Computer Culture." *Signs* 16, no. 1 (1990): 128-57.
[31] C. Wilson, L.A. Sudol, C. Stephenson, and M. Stehlik, "Running on empty: The failure to teach K-12 computer science in the digital age," New York, NY: The Association for Computing, 2010.
[32] S. C. Watkins, *The Young and the Digital: What the Migration to Social Network Sites, Games, and Anytime, Anywhere Media Means for Our Future.* Boston, MA: Beacon Press, 2010.
[33] J. Wing "Computational Thinking." *Communications of the ACM,* vol. 49, no. 3, pp. 33–35, 2006.