From computational thinking to computational participation: Towards Achieving Excellence through Coding in elementary schools

Group members: Lisa Floyd (TVDSB), Yasmin Kafai (Pennsylvania), Steven Khan (Brock), Donna Kotsopoulos (WLU), Laura Morrison (UOIT), Immaculate Kizito Namukasa (UWO), Sowmya Somanath (UoC), Jessica Weber (WLU, WCDSB), Chris Yiu (Western).

Key words: Coding; computational thinking, digital making; mathematics; play; toys

Contributing authors: Namukasa, I. K., Kotsopoulos, D., Floyd, L., Weber, J., Kafai, Y., Khan, S., Yiu, C., Morrison, L., Somanath, S.

Introduction

Technology is interwoven in virtually all aspects of daily living including that of children. In addition to technology being embedded into pragmatic tools (e.g., communication devices, microwave ovens, cars, alarm clocks, etc.), computers and mobile devices for both personal and educational use are common in homes and in schools. In Canada, 83% of homes have Internet access and 97% of these homes report having high-speed access (Statistics Canada, 2013).

Access to the Internet through personal and mobile devices has made a global knowledge network widely accessible. This access has facilitated information sharing that enables users to be *both* consumers of the knowledge and also potential developers of the knowledge. Policy makers and curriculum leaders are not under estimating the implications and the potential influence of this duality of consumer/developer. There is a growing recognition across jurisdictions worldwide that to function, problem-solve, engage in digital innovation, and advance a society already heavily technology based, individuals will need to be able to participate in "computational thinking."

Computational thinking is "an approach to solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computing" (Wing, 2006; 2008, p. 3717). Simply put, computational thinking can also be viewed as a way of thinking algorithmically using design trees from computer science as a guiding structural and sometimes metaphorical framework (Shodiev, 2014). Hoyles and Noss (2015) define computational thinking as entailing abstraction (seeing a problem at different levels of detail), algorithmic thinking (the propensity to see tasks in terms of smaller connected discrete steps), decomposition (solving a problem involves solving a set of smaller problems) and pattern recognition (seeing a new problem as related to problems previously encountered). It involves concepts (loops, conditions, subroutines), practices (e.g., abstraction and debugging) and perspectives, some of which are shared with other subject areas taught in schools such as science, mathematics, social science, language arts and engineering (Lye and Koh. 2014; Kafai & Burke, 2013). Yadav, Zhou, Mayfield, Hambrusch, and Korb (2011) argue that computational thinking needs to be taught in disciplines outside of computer science and in schools right from kindergarten. The term computational thinking could be broadly thought of as computational learning, communicating, creating, and participation. DiSessa (2000) referred to this as computational literacy.

As Kafai (2015) explains, discussions about programming and computational thinking have resulted in *renewed* contemplations about "computational participation" of young children through the learning of coding starting in elementary school curricula. Examples of this can be seen in England where new curriculum has made the teaching of coding in elementary schools mandatory starting in grade one (Government of England, 2013). Finland will be introducing a mandatory elementary computational thinking curriculum starting in 2016, and Estonia has had curriculum in place starting in grade one for all students since 2013 (SITRA, 2015). The emphasis on the word *renewed* is important to note. The historical origins of children participating in computational thinking can be traced back more than thirty years to the visionary work of Seymour Papert. Papert who first proposed the idea of children engaging in coding, with others, developed the software LOGO to enable children to do so (Papert, 1980). Papert (1980) proposed that through learning to program, children could establish "an intimate contact with some of the deepest ideas of science, from mathematics and from the art of intellectual model building" (p. 5).

At a coding and math symposium, in a working group, our collective efforts focused on unpacking, through discussion and application, the implications of coding curriculum in an elementary school context. Parallel to these discussions was the constant consideration of equity, particularly related to gender, given the challenges facing young girls as identified by Kafai (2015). Our discussion and activities focused on what appeared to be potentially four pedagogical phases of learning to code or learning to think computationally: (1) unplugged, (2) making, (3) tinkering, and (4) remixing (or "hacking"). While our group concluded that coding and computational thinking was particularly well aligned to mathematical processes, the idea of thinking algorithmically extends naturally across disciplines and was also emphasized. Consequently, our exploration focused on the larger conceptualizations of these four pedagogical phases and their theoretical and practical implications of learning rather than the actual discipline-specific content.

Unplugged

Unplugged activities can be implemented without the use of computers. Off-screen activities are used to inspire students to become interested in computer science as well as to enhance subject knowledge, often embedding and augmenting computer science concepts into the curriculum (Curzon, 2013; Thies & Vahrenhold, 2013). These constructivist exercises are often kinaesthetic in nature and make "abstract concepts both tangible and visible" (Curzon, 2013, p. 48). Unplugged activities are often collaborative and this may inspire a shift in students' perceptions about the nature of computer science (Nishida et al., 2009; Taub, Armoni, & Ben-Ari, 2012). Unplugged activities can be a powerful way to introduce children to computing concepts and to help them to not only improve on their understanding of concepts across subject areas, but to improve upon their problem solving skills (Curzon, McOwan, Plant, & Meagher, 2014).

By using unplugged activities, barriers such as learning a computer programming language or limited access to computers are potentially avoided, particularly for novices and younger learners (Curzon, 2013; Nishida et al., 2009). Novices benefit from unplugged activities as it "frees students from the implementation details required to produce a working computer program" (Lamagna, 2015, p. 52). Unplugged activities keep screen time for children to a minimum and encourage important active engagement between children and caregivers (LeMay, Costantino, O'Connor, & ContePitcher, 2014).

Lamagna (2015) suggests that "while the ability to program is another valuable skill, an unplugged approach allows students to focus on honing their problem solving skills when that is the primary objective" (p. 52). With unplugged activities, students are able to witness the process required to complete a task, allowing them to put computing into context (Curzon et al., 2014). Not only does incorporating unplugged activities help to make challenging computer science concepts more simple to understand, students enjoy working collaboratively and their motivation and interest in the content appears to increase (Curzon et al., 2014; Lamagna, 2015). Although limited research has been conducted on the effects of the use of unplugged activities on student understanding, Lambert and Guiffre (2009) found that CS Unplugged activities improved fourth grade students' interest in computer science, their confidence in mathematics, and their perceived cognitive skills.

A pioneering resource for unplugged activities that we explored is *Computer Science (CS) Unplugged* (www.csunplugged.org), which is "an educational method for introducing non-specialists to concepts of CS through hands-on activities that don't require the use of a computer" (Nishida et al., 2009, p. 231). *CS Unplugged* was first developed by Tim Bell and his colleagues at the University of Canterbury in New Zealand and the main goal was to attract students to study computer science in high school and post-secondary education (Taub, Armoni & Ben-Ari, 2012). Other websites include www.cs4fn.org, and teachinglondoncomputing.org, which provide resources for teachers who are looking to include unplugged computational thinking activities into their curricula (Curzon, 2013; Curzon et al., 2014).

During our discussions on unplugged computational thinking activities at the symposium, Lisa Floyd gave examples of sorting activities (an example of a computational algorithm) involving toy cars, dancing and classifying three-dimensional shapes based on their geometric properties (see Image 1). According to Nishida et al. (2009), unplugged activities can be taught in any location. However, if done in a room with computers, students might see the unplugged activity as being pre-amble to working on computers, reducing their focus on the task. Consequently, our group ventured outside to the courtyard, with sidewalk chalk and *a set of* plastic polyhedron and one sphere in hand, where we worked collaboratively on a three dimensional shape sorting activity using a simple if-then decision tree structure. The group reflected about what should be stressed for the learners in the sorting activity, as both mathematics and computational thinking concepts were involved.

This activity was originally designed for Lisa Floyd's ninth grade mathematics class as a review of three-dimensional shapes, before an investigation to determine how the formulas for volumes of pyramid, cone and a sphere are developed. This activity is what Resnick and colleagues (2005) have described as a "low floor" activity in that it only requires limited prior knowledge of mathematics or computer programming, and can lead to the development of more advanced mathematical and computer programing concepts such as the development of sorting algorithms.

Jessica Weber noted that the unplugged activity could be modified and also used in her grade two classes when sorting shapes by their properties. When teachers have a strong enough understanding of computational thinking, they can easily adjust their instruction by incorporating a form of computational thinking. When doing so effectively, lessons naturally incorporate a logical focus involving problem-solving skills. Adjusting questions to be more open-ended and inquiry-based is one way to make this strategy more effective. Rather than creating new unplugged activities from scratch, teachers can alter and remix their existing lessons (such as those related to sorting in math or Venn Diagrams) to incorporate coding ideas.

Several approaches and guidelines for teaching unplugged activities are outlined in research papers. Curzon (2013) states that "links from the activities to computational thinking concepts need to be explicitly drawn out" (p. 48). Embedding stories and having students discover ideas themselves help to give them the bigger picture (Curzon, 2013; Nishida et al., 2009). Activities should be designed to be student directed, kinaesthetic, easily implemented, game-based and with embedded challenges, as students tend to have higher motivation and see them as being more entertaining (Nishida et al., 2009). Nishida et al. (2009) also recommends teaching the computational thinking unplugged material within one lesson, rather than spreading it out over several days.

One potential challenge for the implementation of unplugged activities is the teacher's limited coding experience (Curzon et al., 2014). There is a "knowledge and skill gap, especially with respect to subject-based pedagogy" (Curzon et al., 2014, p. 89). However, by providing teachers with the basic concepts of computational thinking through unplugged activities, challenges associated with the need to learn a specific programing language can be minimized. In fact, workshops for teachers should be designed to be unplugged, so that the knowledge of computer science concept gap is narrowed and they are more confident with using the tools in their classroom (Curzon et al., 2014). Teaching mathematics concepts through computational thinking strategies such as unplugged activities requires support and an appropriate balance of methods. Relatively little research exists on the impact and implementation of unplugged activities. More research is necessary.

Image 1. Unplugging

Making

Making is a technical term when used to refer to activities of individuals or groups of people who, inspired by technology, play with things, make things, take things apart and want to make things that solve problems. Makers learn from what they do and share about what they do with other makers (Dougherty, 2012). In making they engage practices such as prototyping and testing products. The artefacts of digital making are physical. By combining making with programming these artefacts directly impact a digital environment (Strawhacker & Bers, 2015). Digital making is also referred to as tangible programming (or, physical computing, digital fabrication or graspable user interfaces, GIU). The artefacts that result from digital making are usually integrated with simple electronics thus extending engagement with computers programming to hardware and fabrication. Tangible objects involved in digital making include but are not limited to digital textiles (Lovell & Buechley 2011), digital games, programmable blocks (Kwon, Kim, Shim, & W. Lee, 2012), tangible avatars/digital puppets (Liu, Liu, Wang, Chen, & Su, 2012), reactive materials, robotics (Kazakoff, Sullivan, & Bers, 2013; Sullivan, Kazakoff, & Bers, 2013), or simply tangible algorithms. Digital making is productive and creative making.

Digital making is seen to teach computational thinking concepts such as sequencing, recursion and debugging (Przybylla & Romeike, 2014), and to be of use in learning other school disciplines such as, kindergarten curriculum, math (e.g., Scarlatos, 2006; Parker, 2012), science and literacy (e.g., Liu et al., 2012). Constructing computational artefacts "encourages students to combine multiple ideas into a cohesive process, organize their understanding in new ways, and 'debug' understandings in their instructions to produce something unexpected (Wilerson-Jerde & Hoda, 2014 p.102). Digital making is a tangible representation of computational thinking.

Digital making is a major part of the re-emergence of programing for kids and youth. The theme on tangible programming tools was present in the original movement on programming for kids through which children learned to program both "screen" turtles and "floor" turtles (Papert, 1980). Papert referred to these digital turtles as "objects-to-think-with … in which there is an intersection of cultural presence, embedded knowledge, and the possibility of personal identification"(p.11).

The tangible programming movement appears much aware of the need to go beyond conceptions of digital and electronics common in industrial applications of programming mainly limited to motion (Przybylla & Romeike, 2014; Resnick, 2007). Bowler (2014) refers to digital making as the connection between the virtual and the real when new intuitive interfaces that use sensors, micro-controllers, motion, sound, light and other actuators as well as other tangible materials to interact with the environment are created between computer objects and humans. To Resnic kits such as Cricket "that give the students power to create and control in the real world" (p.20) are about diversifying participation and creativity. Cricket kits also contain art and craft materials. Kids become makers of a variety of digital technology products. Robot kits such as Hummingbird controller by Carnegie Mellon University www.hummingbirdkit.com, LEGO WeDo robotics construction, and Cricket are making their way in schools, especially, in afterschool, camp, library and other informal activities. Several other Arduino microcontroller boards (www.arduino.cc) and Arduino open source software are behind several digital makers projects in schools and at maker shops in museum and in the community.

One might critique that the growing interest in programming is promoted by computer scientist with intent of feeding their "pipeline". Quite the contrary, the digital making movement largely arose and is growing as a social movement from the creative arts (Przybylla & Romeike, 2014). Artists, designers, hobbyists and makers, have especially used microcontrollers in creating interactive objects. A good example of the link between digital making and the creative arts is electronic textiles or wearable computing (see Berzowska electronic textiles at http://www.berzowska.com/; Ngai, Chan, Leong, & Ng, 2013). Computing textiles (e-textiles) combine skills of computer programming, electronics and sewing. Lilypad Arduino (http://lilypadarduino.org/) is a micro controller board for wearables and e-textiles. These kits contain pre-assembled circuits and electronics.

For education innovators and researchers digital making is a move further from text-based computing languages via visual, screen-based languages to tangible physical programming tools. A commonly and widely used and known computer programming software for children and novices is Scratch (see https://scratch.mit.edu/). It is a visual-graphic programing language. Other similar visual programming languages include Alice by Carnegie Melonie University, and Microworlds (Pierce, 2015). Such visual user interfaces are making programming more accessible. They are also making it possible to learn other subjects with programming. Gapminder world (http://www.gapminder.org/for-teachers/) is an example of a forum promoting digital visualization activities for teachers. Pencilcode.net is another example of use of visual user interfaces.

Making is multimodal and thus provides a context for understanding computational concepts and processes differently. Whereas visual programming tools eliminate the difficulty posed by syntax, tangible programming further eliminates any difficulty posed by the manipulation of a mouse, mouse pad screen or keyboard, especially for participants who normally would not be able to engage in formal programming. Tangible programming interfaces may be combined with visual graphical interfaces to form hybrid user interfaces and interactive graphical interfaces.

Digital making has been noted as an opportunity for making abstract programming concepts and ideas more physical, accessible for younger children and more amenable to social learning. In computer science courses digital making is also related to object-operated languages (Govender & Grayson, 2008)) developed to counteract the traditional procedural programming with its complex syntax and structures that must be learned first before a student learn to program and to use programing to solve problems (Goevender & Grayson, 2008).

While most research in this new area has been innovation-oriented, a few studies have been done on the influence of digital tangibles on learning. Tangible user interfaces (TUIs) have been observed to be as equally effective as visual user interfaces (VUIs) at learning programming languages and more helpful at early stages of learning a programming concept for kindergarteners (Kazakoff et al. 2013; Kwon, Kim, Shim, & W. Lee, 2012; Strawhacker & Bers, 2015) and for children with certain special needs (Farr, Yuill, & Raffle, 2010). TUIs have also been linked to improved interest levels of first year computer programming students (Corral, Balcells, Estévez, Moreno, & Ramos, 2014) and inservice and pre-service computer science teachers (Govender, & Grayson, 2008). Bers and Horn (2010) observe that young children as young as four years old, through tangible and visual programming, are able to understand the basics of programming and can build and program simple robotic manipulatives. Tangible programming interfaces were more appealing to children younger than 16 years in a museum setting and equally appealing to both boys and girls in contrast to visual-graphic programming which appealed more to boys (Horn, Crouser & Bers, 2011). In their study the levels of understanding about programming concepts gained through tangibles and graphical interfaces did not differ.

Kafai (2015) noted that gender issues are of significant concern with making. Messaging in making activities may send unintentionally messages for females. Some learners receive several messages and signals, from school and outside school settings that suggest they are not able to do computing. This is also true for mathematics and other STEM disciplines. The electronic textiles movement is one digital making context that has provided an avenue for participation for girls and women in particular (see examples at http://www.instructables.com/id/turn-signal-biking-jacket/, and http://www.design-research-lab.org/). The underlying message is that making activities need to be inclusive and need to engage a diverse range of learners and abilities.

It has been noted that tangible user interfaces do not only increase participation in introductory computing as well as address equity issues, but also make it possible to prompt reflective discussions on programming among learning. Students quickly get to understand basic computing concepts without stumbling on the syntax of programming languages, proceed to engage with hard ware when they make technology products, thus getting the opportunity to be able to reflect on the process, the role and limitations of programming.

Image 2. Making

Tinkering

When commenting about tinkering, Papert (1980) described learning to consist of building up a set of materials and tools that one can handle and manipulate (p. 173). *Lego* and other block building activities can be seen as classic examples of tinkering where the concepts of math and geometry can be explored in a creative way to students. The idea of tinkering here was to allow students to use materials, as tools, to represent computational thinking. To this end, in the working group we worked with several different tools including the *Arduino*, *Little Bits*, and several robots. Each of these tools is tangible (see Image 3).

Any programming is ultimately manifested in a physical sense (e.g. sound, lights, movement, etc.) rather than simply living inside a computer screen. This hands-on aspect allows for students to easily see the connection between changes in the program and in the physical model. Activities can be created that exercise the concept of algorithmic thinking such as constructing a program to guide a robot through an obstacle course or programming the Arduino to display specific patterns.

The tools vary in their applicable skill range. When looking at each tool, we are primarily concerned with two factors: the skill "floor" - the minimum skill required to begin using the tool, and the skill "ceiling" - the maximum potential of the tool (Gadanidis, J., Scucuglia, & Tolley, 2009; Resnick et al., 2005). The Little Bits are colour-coded magnetic blocks that snap together giving it a low skill floor, but without many opportunities to provide complexity it also has a low skill ceiling. The Arduino has a higher skill 'floor' since some programming knowledge is required to use the Arduino language, but also has a much higher skill ceiling as the language allows for more complex interactions.

While these examples of tinkering are physical, virtual tinkering can be just as effective. *Minecraft*, a block-building video game recently bought by Microsoft, still retains the aspects of tinkering despite not being tangible. Minecraft retains the same concept of block building like Lego, while also introducing the concept of circuits through a material known as *Redstone*. Students can create basic circuits to affect their surroundings using Redstone, allowing their creations to have a manifestation within their world (e.g. opening a door, turning on/off a light). An educational mini-game, *Circuit Madness*, developed by several students at Miami University, teaches students about the basics of circuits and logic gates using the Redstone material (Duncan, 2011).

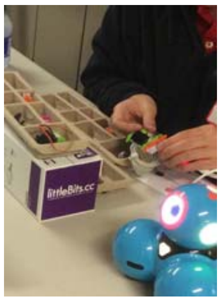

Image 3. Tinkering

Remixing (or "Hacking)

Remixing involves proficiency with examining code with a critical eye, as well as modifying and manipulating code to suit new purposes. Chris Yiu described how remixing is common in the computer science community in that programmers often use open-source code as a base and then modify it to suit their particular needs. Throughout our discussions, the notion of remixing was addressed. What skills and mindsets are required for remixing? Where does remixing fall in the potential pedagogical phases of learning to code or think computationally? Is credit given to the original creator? It was agreed that remixing requires sophisticated reasoning and problem solving skills, therefore should be introduced later in coding skill acquisition.

When introducing coding as part of computational thinking to students we need to develop this flexible, creative thinking and explicitly teach remixing as a means of developing solutions. We discussed remixing largely within the context of Scratch and emphasized the importance of community building in its success. According to Resnick et al. (2009), "community members are constantly borrowing, adapting, and building on one another's ideas, images, and programs. Over 15% of the projects . . . are remixes of other projects" (p. 65).

This ability for students to view, share, and connect with others builds capacity and creates a supportive environment for risk taking. Resnick et al. (2009) also shared how perceptions of remixing have evolved from negative notions of others "stealing" ideas, to a more open-minded view of a community of learners stating that, "Our goal is to create a culture in which Scratchers feel proud, not upset, when their projects are adapted and remixed by others" (p. 65).

Watters (2011) stated that this is a clear benefit of participating in the Scratch community in that it enables students to build upon others' work and give credit to others when they do so. When teaching students to code, it is clear that remixing is an essential element. Opportunities to analyze code, make connections and create new applications for existing code, are necessary aspects in the development of computational thinkers. An example of a site that encourages remixing is Projectspark.com, which involves game players not only in playing already created digital games but also in creating and playing the games in their new role as creators of games**.**

Conclusions

Our exploration of unplugging, making, tinkering, and remixing highlights several key features of coding. First, coding is a continuum of skills that builds first on fundamental kinaesthetic and experience-based concepts to more complex structures found in computer science. Coding is playful and is engaging. Coding is exploratory by nature and is well suited to project-based explorations in particular. Coding is social because solving-problems most efficiently and productively involves others and potentially others' code (i.e., remixing). Finally, our exploration highlighted to us that coding is possible – for all students when tasks are designed with low entry and high ceilings that facilitate multiple entry points (Resnick et al., 2005).

The current focus on coding and computational thinking can be viewed as a renewed focus. Indeed, our working group and the broader symposium participants engaged in critical discussion about why this current movement might persist whereas the early and visionary of Seymour Papert and colleagues did not. The general consensus amongst the Symposium participants related to the initial comments in the introduction. Technology has evolved to a level where there is an inescapable daily dependence for most people. As one article proposes the future will be built by those who know to code (SITRA, 2015). Consequently, not including coding for all children can be potentially undermining the extent to which participation in such future building is limited. It is this reality that suggests that this renewed focus on coding will have longevity and coding will increasingly see itself inscribed in policy through mandatory curricula.

Key challenges going forward will be in training teachers. It was noted that the teachers at the Symposium appeared to be the atypical innovators that are supporting this bottom up movement of coding in schools – with limited resources, professional development, and infrastructure. This working group proposed the following recommendations for designing a coding course/module/session for teachers.

The course needs to:

- Be informed by research and theory and infused with computer science and teaching experience;
- Focus on coding skills that are key for learners to know;
- Make explicit differences between computational thinking and coding (or computer science);
- Explore coding as a tool, a concept, set of skills (e.g., soft, process and thinking skills), and as an application;
- Address learning outcomes of awareness and competency;
- Explore multiple environments/platforms such as block programming platforms (e.g., scratch);
- Include tangibles (e.g., little bits) and robotics (e.g., dash);
- Include task design discussions and opportunities;
- Include the pedagogical phases of unplugging, making, tinkering, and remixing.

The challenges of implementing such programming were noted. As Kafai (2015) pointed out, a key challenge will be in finding qualified personnel to teach the teachers. Currently, she proposes that a dearth exists in this skill base in most relevant units (i.e., Faculties of Education) at most post-secondary institutions. As Higginson (2015) noted in his response to Kafai regarding the "other 96% of teachers" who are not early technology adopters, the learning curve for deep understanding of coding and its potential can be substantial and this will have to be seriously attended to. As noted in the Executive Summary of the NCTM's *Principles to Action* (2014), "too many teacher of mathematics remain professionally isolated, without the benefits of collaborative structures and coaching, and with inadequate opportunities for professional development related to mathematics teaching and learning" (p.2). This phenomenon makes it difficult to 'scale up' innovation resulting in isolated "pockets of excellence" and the inability to effect systemic change.

The scaling-up, in thinking and practice, in moving to broad and deep computational participation discussed by Kafai (2015) is a reminder of the fact that innovations are not always shared equitably in terms of access, quality of activities and experiences, and in terms of legitimate community membership and greater than peripheral participation. In Ontario, the renewed vision for education articulated in *Achieving Excellence* (Ontario Ministry o f

Education, 2014) has articulated four goals – achieving excellence, increasing equity, supporting and promoting well-being and enhancing confidence in publicly funded education. Coding and digital-making activities perhaps have a significant potential to realize all of these goals. As Higginson (2015) recalled, an emphasis on social justice was always a part of Papert's thinking in his work. When talking about equity and computing Anderson (2009) and Margolis, Estrella, Goode, Holme and Nao (2011) encourage educators to challenge the mindsets about abilities of students of color, girls, students with disabilities and those from low-income communities. Non-productive mindsets continue to limit many kids from participating in computing courses in schools and at universities. Kafai & Burke (2013) refers to this as computational participation. It will be useful to also make these socio-political connections explicit as providing avenues for developing alliances with classroom teachers, educational leaders and policy-makers.

Our working group concluded with numerous potential research questions: (1) What is the coding continuum of learning from early childhood to post-secondary? (2) How can gender equity be facilitated? (3) When is coding play and when is it not? (4) How do we make coding skills and habits transparent without taking over the play? (5) What role do community partnerships play in supporting the development of coding in young children? (6) How do we engage educational leadership or policy makers so that more enabling conditions can be created for teachers in which to experiment, implement, innovate and create transformative computational participatory learning experiences? (7) How does the use of multiple representations assist with coding? (8) How does coding assist with making multiple representations of mathematical ideas more meaningful and relevant? (9) How do we get teachers to get thinking about overlaying coding on tasks that they are already working on? (10) How can we chunk the coding concepts so coding is not overwhelming? (11) How can coding instruction be differentiated? (12) Do we make the coding ideas explicit in disciplinary content? (13) In what ways could we helpfully work on carefully managed and well-designed tasks and lessons for coding and math?

References

Anderson, N. (2009). Equity and information communication technology (ICT) in education. New York: Peter Lang.

Barr, V. and C. Stephenson, Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? ACM Inroads, 2011. 2(1): p. 48-54.

Bers, M. U., & Horn, M. S. (2010). Tangible programming in early childhood. In I. R. Berson and M. J. Berson (Eds.), High Tech Tots: Childhood in a digital world (pp. 49-69). Charlotte, North Carolina: IAP.

Bowler, L. (2014). Creativity through "maker" experiences and design thinking in the education of librarians. *Knowledge Quest, 42*(5), 58-61.

Corral, J. M. R., Balcells, A. C., Estévez, A. M., Moreno, G. J., & Ramos, M. J. F. (2014). A game-based approach to the teaching of object-oriented programming languages. *Computers & Education, 73*, 83-92.

Curzon, P. (2013). cs4fn and computational thinking unplugged. *WiPSE '13 Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, 47-50.

Curzon, P., McOwan, P., Plant, N., & Meagher, L. (2014). Introducing teachers to computational thinking using unplugged storytelling. *WiPSCE'14 Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 89-92.

diSessa, A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: MIT Press.

Dougherty, D. (2012). The maker movement. *Innovations*, 7(3), 11-14.

Duncan, S. C. (2011). Minecraft beyond construction and survival. *Well Played: a journal on video games, value and meaning, 1*(1), 1-22.

Farr, W., Yuill, N., & Raffle, H. (2010). Social benefits of a tangible user interface for children with autistic spectrum conditions. *Autism, 14*(3), 237-252.

Gadanidis, G., J., H., Scucuglia, R., & Tolley, S. (2009). Low floor, high ceiling: Performing mathematics across grades 2-8. *Paper presented at the annual meeting of the North American Chapter of the International group for the Psychology of Mathematics Education*.

Govender, I., & Grayson, D. J. (2008). Pre-service and in-service teachers' experiences of learning to program in an object-oriented language. *Computers & Education, 51*(2), 874-885.

Government of England. (2013). *National curriculum in England: Computing programmes of study*. Retrieved June 29, 2015, from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-studyhttp://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study

Higginson, W. (2015). Response to Kafai. *Math + Coding Conference, Western University, June 2015.*

Horn, M. S. (2009). Tangible computer programming: Exploring the use of emerging technology in classrooms and science museums. Unpublished Ph.D. dissertation.

Horn M.S., Crouser, R. J. & Bers, M.U (2011). Tangible interaction and learning: The case for a hybrid approach. *Personal and Ubiquitous Computing, 16*(4), 379-389.

Hoyles, C., & Noss, R. (2015). *Revisiting programming to enhance mathematics learning*. Paper presented at the Math + Coding Symposium. Western University.

Kafai, Y. B. (2015). *Connected code: A new agenda for K-12 programming in classrooms, clubs, and communities*. Paper presented at the Math + Coding Symposium, Western University, London, Ontario, Canada.

Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan, 95*(1), 61.

Kazakoff, E. R., Sullivan, A., & Bers, M. U. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal, 41*(4), 245-255.

Kwon, D. -Y, H. -S Kim, J. -K Shim, and W. -G Lee. 2012. Algorithmic bricks: A tangible robot programming tool for elementary school students. *IEEE Transactions on Education* 55, (4) (11): 474-479

Lamagna, E. (2015). Algorithmic thinking unplugged. *Journal of Computing Sciences in Colleges, 30*(6), 45-52.

Lambert, L., & Guiffre, H. (2009). Computer science outreach in an elementary school. *Journal of Computing Sciences in Colleges*, *24*(3), 118-124.

LeMay, S., Costantino, T., O'Connor, S., & ContePitcher, E. (2014). Screen time for children. *IDC'14 Proceedings of the 2014 conference on Interaction design and children*, 217-220.

Liu, C., Liu, K., Wang, P., Chen, G., & Su, M. (2012). Applying tangible story avatars to enhance children's collaborative storytelling. *British Journal of Educational Technology, 43*(1), 39-51.

Lovell, E. & Buechley, L. (2011). LilyPond: An Online Community for Sharing E-Textile Projects. In Proceedings of C&C, pp. 365-366.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51-61.

Margolis, J. (2011). Stuck in the shallow end: Education, race, and computing (1st MIT Press pbk. ed.). Cambridge, Mass: MIT Press.

Margolis, J., Estrella, R., Goode, J., Jellison-Holme, J., & Nao, K. (2008). *Stuck in the Shallow End: Education, Race, & Computing*. MIT Press: Cambridge, MA.

NCTM (2014). *Principles to Actions: Ensuring Mathematical Success for All.* NCTM: Reston, VA.

Ngai, G., Chan, S. C., Leong, H. V., & Ng, V. T. (2013). Designing I*CATch: A multipurpose, education-friendly construction kit for physical and wearable computing. *ACM Transactions on Computing Education, 13*(2), 30.

Nishida, T., Kanemune, S., Idosaka, Y., Namiki, M., Bell, T., & Kuno, Y. (2009). A CS unplugged design pattern. *SIGCSE, 41*(1), 231-235.

Ontario Ministry o f Education. (2014). *Achieving Excellence: A renewed vision for education in Ontario*. Toronto, Ontario: Author.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Parker, T. (2012). ALICE in the real world. *Mathematics Teaching in the Middle School, 17*(7), 410.

Pierce, M. (2013, May). Coding for middle schoolers: next-generation programming languages for children are taking up where Logo left off and teaching young students how to code to learn. *T H E Journal* [Technological Horizons In Education], *40*(5), 20+.

Przybylla, M., & Romeike, R. (2014). Physical computing and its scope - towards a constructionist computer science curriculum with physical computing. *Informatics in Education, 13*(2), 225-240.

Resnick, M. (2007). Sowing the seeds for a more creative society. *Learning & Leading with Technology*, 35(4), 18-22.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM., 52*(11), 60-67.

Resnick, M., Myers, B., Nakakoji, K., Shneiderman, B., Pausch, R., Selker, T., et al. (2005). *Design principles for tools to support creative thinking. National Science Foundation workshop on Creativity Support Tools*. Washington DC.

Scarlatos, L. L. (2006). Tangible math. *Interactive Technology and Smart Education, 3*(4), 293-309.

Shodiev, H. (2014). *Computational thinking and simulations in teaching science and mathematics* Paper presented at the AMMCS-2013 Interdisciplinary Conference Series: Applied Mathematics, Modelling, and Computational Science.

SITRA. (2015). *Future will be built by those who know how to code*. Retrieved June 29, 2015, from http://www.sitra.fi/en/artikkelit/well-being/future-will-be-built-those-who-know-how-codehttp://www.sitra.fi/en/artikkelit/well-being/future-will-be-built-those-who-know-how-code

Statistics Canada. (2013). *Canadian internet use survey, 2012*. Retrieved June 29, 2015, from http://www.statcan.gc.ca/daily-quotidien/131126/dq131126d-eng.htmhttp://www.statcan.gc.ca/daily-quotidien/131126/dq131126d-eng.htm

Strawhacker, A., & Bers, M. U. (2015). "I want my robot to look for food": Comparing kindergartner's programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education, 25*(3), 293-319.

Sullivan, A., Kazakoff, E. R., & Bers, M. U. (2013). The wheels on the bot go round and round: Robotics curriculum in pre-kindergarten. *Journal of Information Technology Education: Innovations in Practice, 12, 203-219.*

Taub, T., Armoni, M., & Ben-Ari, M. (2012). CS Unplugged and middle-wchool students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education (TOCE), 12*(2), 8.

Thies, R., & Vahrenhold, J. (2013). On plugging "unplugged" into CS classes. *SIGCSE '13 Proceeding of the 44th ACM technical symposium on computer science education*, 365-270.

Watters, A. (2011). *Scratch: Teaching the difference between creating and remixing* [Electronic Version]. Retrieved July 7, 2015, from http://ww2.kqed.org/mindshift/2011/08/11/scratch-teaching-kids-about-programming-teaching-kids-about-remixing/http://ww2.kqed.org/mindshift/2011/08/11/scratch-teaching-kids-about-programming-teaching-kids-about-remixing/

Wilkerson-Jerde, M. (2014). Construction, categorization, and consensus: Student generated computational artifacts as a context for disciplinary reflection. *Educational Technology Research and Development, 62*(1), 99-121.

Wing, J. M. (2006). Computational thinking and thinking about computing. *Communications of the ACM., 49*, 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of The Royal Society A, 366*, 3717–3725.

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011). Introducing computational thinking in education courses. *SIGCSE, 11*, 465-470.