Digit Exp Math Educ
DOI 10.1007/s40751-017-0031-2

MATHEMATICS AND PROGRAMMING

A Pedagogical Framework for Computational Thinking

Donna Kotsopoulos[1] · **Lisa Floyd**[2] · **Steven Khan**[3] · **Immaculate Kizito Namukasa**[4] · **Sowmya Somanath**[5] · **Jessica Weber**[6] · **Chris Yiu**[7]

© Springer International Publishing 2017

Abstract Our goal in this paper is to propose a Computational Thinking Pedagogical Framework (CTPF), developed from constructionism and social-constructivism theories. CTPF includes four pedagogical experiences: (1) unplugged, (2) tinkering, (3) making, and (4) remixing. Unplugged experiences focus on activities implemented without the use of computers. Tinkering experiences primarily involve activities that take things apart and engaging in changes and/or modifications to existing objects. Making experiences involve activities where constructing new objects is the primary focus. Remixing refers to those experiences that involve the appropriation of objects or components of objects for use in other objects or for other purposes. Objects can be digital, tangible, or even conceptual. These experiences reflect distinct yet overlapping CT experiences which are all proposed to be necessary in order for students to fully experience CT. In some cases, particularly for novices and depending on the concepts under exploration, a sequential approach to these experiences may be helpful.

Keywords Computational thinking · Experiences · Mathematics · Making · Pedagogical · Remixing · Tinkering · Unplugged

✉ Donna Kotsopoulos
dkotsopo@wlu.ca

[1] Faculty of Education and Faculty of Science (Department of Mathematics), Wilfrid Laurier University, 75 University Avenue, Waterloo, ON N2L 3C5, Canada

[2] Faculty of Education, Western University/Thames Valley District School Board, London, ON, Canada

[3] Faculty of Education, Brock University, St. Catharines, ON, Canada

[4] Faculty of Education, Western University, London, ON, Canada

[5] Department of Computer Science, University of Calgary, Calgary, AB, Canada

[6] Waterloo Catholic District School Board, Waterloo, ON, Canada

[7] Department of Computer Science, Western University, London, ON, Canada

Published online: 09 March 2017

A Pedagogical Framework for Computational Thinking

Technology is interwoven in virtually all aspects of daily living including that of children. In addition to technology being embedded into everyday 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 and developers of knowledge. The implications and the potential influence of this duality of consumer/developer are not being underestimated by policy makers and curriculum leaders. There is a growing recognition across many jurisdictions worldwide that to function, problem-solve, engage in digital innovation, and advance a society already heavily technology-based, individuals require some level of "computational thinking" (CT) and this requirement is often cumulative in that newer technologies require an understanding of older technologies.

CT is "an approach to solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computing" (Wing 2006, 2008, p. 3717). CT can also be viewed as thinking algorithmically by using principles from computer science as a guiding structural and sometimes metaphorical framework (Shodiev 2013). Hoyles and Noss (2015) define CT 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).

CT involves concepts (e.g., loops, conditions, subroutines) and practices (e.g., abstraction and debugging) primarily from computer science which are shared across other disciplines such as science, mathematics, social science, biology, language arts and engineering (Kafai and Burke 2013; Lye and Koh 2014). These disciplines are also simultaneously computational in that digital technology is increasingly necessary for each discipline (Weintrop et al. 2016). Despite the obvious relevance of CT to computer science, scholars argue that CT needs to be taught in disciplines outside of computer science beginning in kindergarten (Barr and Stephenson 2011; Yadav et al. 2011). CT is proposed to be "a powerful cognitive skill that can have a positive impact on other areas of children's intellectual growth" (Horn et al. 2012, p. 380); therefore, according to Wing (2006), it should be added "to every child's analytical ability" (p. 33).

The intense interest in CT has resulted in many jurisdictions worldwide proposing that CT is introduced either as a stand-alone course or a cross-curricular expectation. For example, England has made the teaching of "coding" (or computer programming) part of the national curriculum in elementary schools and mandatory starting in grade one (Berry 2013; Government of England 2013). Coding is one example of CT and arguably the most popularized example to date. Finland will be introducing a mandatory elementary CT curriculum starting in 2016, and Estonia has had curriculum in place starting in grade one for all students since 2013 (SITRA 2014). Similar trends are also being observed in some parts of Canada (e.g., British Columbia Government 2016; Province of Nova Scotia 2015) and the United States (e.g., The White House 2016).

The current focus on CT can be viewed as a *renewed* focus. The historical origins of children participating in CT can be traced back more than thirty years to the visionary work of the late Seymour Papert who developed the software LOGO to enable children to engage in computer programming (Papert 1980). It is worth contemplating why Papert's pioneering work was not enduring or widely adopted at the time (Pierce 2013). Quite simply, technology has evolved to a level where there is an inescapable daily dependence or interaction for most people. As one online article proposes in its title "the future will be built by those who know how to code" (SITRA 2014). Additionally, much simplified programming languages, utilizing block-based click and build components, have made participation in CT and coding much more accessible. It is this reality that suggests that the renewed focus on coding will have longevity and will increasingly see CT inscribed in education policy through mandatory curricula. Consequently, not including coding for all children can be potentially undermining the extent to which participation in such future building is limited (Kafai 2015).

Our goal for this paper is to propose a CT Pedagogical Framework (CTPF), which is rooted in constructionism (Papert 1980, 1987; Papert and Harel 1991) and social-constructivism theories (Vygotsky 1978). CTPF, as we conceive of it, includes four pedagogical experiences: (1) unplugged, (2) tinkering, (3) making, and (4) remixing. The CTPF was developed by the authors and with the support of other scholars (see acknowledgment) in a working group focused on CT pedagogy at a symposium on CT (Namukasa et al. 2015). During the working group, the four experiences were identified in the available literature and then explored through activities.

The working group included novices and experts (teachers, graduate students, and researchers) in CT. Central to our discussions and explorations were the following questions: (1) Was a particular experience appropriate for a novice? (2) Were some of the experiences more cognitively or technologically demanding? And (3) How do these experiences relate to one another? The refinement the CTPK continued beyond the symposium through dialogue and our own individual explorations with each of the experiences and the framework. For some of us, the exploration occurred at the individual level. For others, the exploration happened in pre-services settings and others explored the experiences in authentic professional development contexts. Many of us explored the experiences and the framework with children in classrooms.

This further exploration inspired a revision in the ordering of the experiences, a clarification in role of using the experiences sequentially, and a reconsideration of the role of unplugged experiences. For example, our early thinking about the CTPF considered these experiences instead as "phases" which could occur sequentially (i.e., first unplugged, then tinkering, then making, and finally remixing). In some cases, particularly for novices and depending on the concepts under exploration, this may be the case. Unplugged experiences, as we explain shortly, may be useful preceding any of the other experiences.

Consequently, the proposed CTPF should instead reflect distinct yet overlapping CT experiences which we propose are *all* necessary in order for students to fully experience CT. We emphasize up front that our proposed framework is intended to contribute to the early discussions on CT and pedagogy. The need for empirical data about efficacy of the framework is necessary. The proposed CTPF is an outgrowth of the available CT pedagogy research – which is widely agreed to be in its infancy. While we believe that

the proposed CTPK has great potential, there may be in fact other experiences or other sequences of the experience that may be more powerful for learning CT and this is may be discovered with further research.

Our disciplinary lens used to exemplify the proposed CTPF is mathematics given the obvious and numerous connections between mathematical and CT, particularly in the areas of problem-solving, modeling, and analyzing (Gadanidis 2015; Sneider et al. 2014). This is not to suggest that CT is only relevant to mathematics education. CT can also be naturally extended across other curricula too (Horn et al. 2012). As noted earlier, some jurisdictions are taking a cross curricular approach because of the relevance of CT to other disciplines (Kafai and Burke 2013; Lye and Koh 2014). We propose that the CTPF may be useful for learning CT across the elementary and secondary levels of education and particularly useful for novice learners, including pre-service and in-service teachers.

Numerous frameworks for CT have been proposed but, to the best of our knowledge, frameworks that describe pedagogy have not been established yet. For example, Brennan and Resnick (2012) describe three "dimensions" of CT which include: computational concepts, computational practices, and computational perspectives. These dimensions capture the what, how, and the why of CT without fully addressing the actual teaching of CT. Similarly, Weintrop et al. (2016) propose a "taxonomy of practices" which include data practices, modeling and simulation practices, computational problem solving practices, and systems thinking practices. These practices describe CT in terms of actions. Resnick et al. (2005) describe "design principles of tasks" which assist greatly in thinking about why a particular task is CT and the appropriateness of a task to a learning goal or teaching objective. Resnick et al. (2009) also introduce the *practices* of tinkering and remixing, terms which we also use to describe pedagogical experiences. Our view of tinkering and remixing takes a distinctly teaching lens that focuses on learning rather than only the action.

Theoretical Perspectives

The proposed four pedagogical experiences are deeply rooted in *constructionism learning theory*, first proposed by Papert (1987), and Vygotsky's (1978) "zone of proximal development." Papert's theory of constructionism posits that learners construct internal representations to make sense of their environment and to develop knowledge. Learners build on existing knowledge through student-centered and inquiry-based active learning.

Papert and Harel (1991) makes clear that constructionism is more complex than simply "learning-by-making." Papert explains constructionism in his National Science Foundation proposal as follows: "We take a view of learning as a reconstruction rather than as a transmission of knowledge. Then we extend the idea of manipulative materials to the idea that learning is most effective when part of an activity the learner experiences as constructing a meaningful product" (Papert 1987, Abstract). The learner plays an active and primary role, rather than the teacher, in constructing their knowledge and this is the central consideration for teachers as they guide instructional activities in the classroom.

Papert and Harel (1991) perspective is fundamentally multi-modal in that internal representations are constructed from engagement with artifacts in the real world and that may include sound, text, imagines, motion, and so forth. We also take this broad view of artifacts in our definition of "objects" which is frequently used in describing and illustrating the proposed CTPF. As used in this context, object is meant to describe something that is digital, tangible, or even conceptual. For example, a block of computer programming is a digital object. A segment of a structure or building blocks are examples of tangible objects. A variable or formula can be viewed as a conceptual object. Some objects can be both tangible and digital (e.g., robots) or digital and conceptual (e.g., a formula in a computer program). In the next sections, examples of objects are provided to illustrate the four experiences in the CTPF.

Vygotsky (1978) defines zone of proximal development (ZPD) as "the distance between the actual development level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers" (p. 86). Consequently, learning is viewed as social, collaborative, and interactive. The four pedagogical experiences partially reflect a developmental continuum, or zones of proximal learning, whereby each pedagogical experience may reflect an increasingly more challenging cognitive demand level than the previous pedagogical experience for learners. This developmental perspective should not be interpreted as suggesting new learning can only occur within one particular CT experience or another. Nor should these experiences be viewed as rigidly sequential; the sequences of the experiences may vary or even be iterative.

New concepts can still, of course, be introduced during any aspect of the CTPF; however, the actual tasks associated with of some of the proposed experiences in the CTPF, such as making and remixing require, in our opinion, a higher level of foundational understanding and skill level of CT than the unplugged or tinkering experiences. For example, in the making experience, some understanding of computer programming or coding is necessary to write computer programs anew. Likewise, the essential skills necessary to remix were likely first learned through tinkering (e.g., applying simple modifications to existing computer programs). This is not to say that there may not be individuals who engage in computer programming, for example, without any prior knowledge. Indeed, the experiences are perceived as distinct but overlapping (see Fig. 1). Our point is that to develop CT, approaching learning sequentially through these four experiences could potentially be helpful – particularly for teachers and for students with limited CT background, but exposure to all four experiences is necessary to fully experience CT.

Less obvious about the proposed CTPF at the onset is the social aspect of Vygotsky's ZPD. To explain our view, across each of the experiences we outline shortly the opportunity for engagement with "other" either occurs frequently as part of the experience (e.g., building activities), is often a feature of the experience (see unplugged), or may be implicit in the experience. For example, with remixing, students appropriate (or "hack") on others' work and this is an opportunity to both learn and build knowledge. Consequently, social is an important aspect of the CTPF.

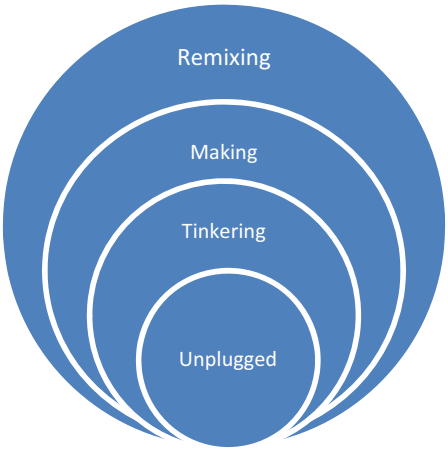

Fig. 1 Four pedagogical experiences

Four Pedagogical Experiences

Unplugged

Unplugged experiences focus on CT activities that are implemented without the use of computers. Barriers such as learning a computer programming language or limited access to computers are potentially avoided, particularly for novices and younger students (Curzon 2013; Nishida et al. 2009). Consequently, it is an accessible way to introduce children to CT (Curzon et al. 2014; Thies and Vahrenhold 2013). These experiences release "students from the implementation details required to produce a working computer program" (Lamagna 2015, p. 52).

Unplugged experiences are often first and foundational in learning CT because they require possibly the least amount of cognitive demand and technical knowledge. This is not to say that unplugged experiences cannot be cognitively demanding. Rather, the purpose of unplugged experiences are to introduce preliminary and overlapping concepts related to CT that can then be explored in a more sophisticated way, either conceptually or technologically, during the other experience. Like the other three experiences, unplugged experiences may also be repeated prior to any of the other experiences to introduce foundational concepts without the use of a computer. For example, a teacher may do an unplugged activity prior to making experiences in order to introduce a relevant new concept without the use of technology.

Unplugged experiences are often collaborative and kinesthetic (LeMay et al. 2014; Nishida et al. 2009; Taub et al. 2012). From a design perspective, unplugged experiences also have many of the foundational design principles outlined by Resnick et al. (2005) that include: multiple access points for students, the potential to build in complexity within the task, they are simple, and they support explorations.

With unplugged experiences, students are able to witness and experience the process required to complete a task, allowing them to put CT into a relatable context (Curzon et al. 2014). Not only does incorporating unplugged experiences 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). Lambert and Guiffre (2009) found that unplugged experiences improved fourth grade students' confidence in mathematics and their perceived cognitive skills.

According to Nishida et al. (2009), unplugged experiences 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. Curzon (2013) states that "links from the activities to CT concepts need to be explicitly drawn out" (p. 48). Unplugged experiences should be designed to be student-directed, kinesthetic, easily implemented, game-based, and with embedded challenges (Nishida et al. 2009). Nishida et al. (2009) also recommends teaching the unplugged experiences within one lesson, rather than spreading it out over several days.

Rather than creating new unplugged experiences, teachers can use existing lessons (i.e., such as those related to sorting or those using Venn Diagrams) to incorporate CT thinking. The challenge in using existing material for teachers is being able to identify the CT in the activity or being able to imbed explicitly CT within the task where it may not have previously existed.

One example of an unplugged activity is the sorting of shapes based on properties and attributes using a simple if-then decision tree structure. Sorting is an example of a computational algorithm whereby properties or attributes form the rule for placing an object in one collection versus another (Namukasa et al. 2015). This activity is 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 (Papert 1980; Resnick et al. 2005). Objects in an unplugged context can be tangible or conceptual, rather than digital or computer-based. Unplugged experiences can also be coupled with any of the other experiences described shortly.

Another good example of an unplugged experience is found in this special volume. Gadanidis et al. (2016) describe an unplugged coin toss activity with grade one students that explores the Binomial Theorem (see paper and Fig. 6). In this activity, students are asked to record and compare the number of outcomes following each of the paths (1 to 6). The activity explores probability, estimation, and conjecturing. It provides concrete and kinesthetic experiences for the child and allows for foundational knowledge to be developed. It reflects Weintrop et al. (2016) data practices, modeling and simulation practices includes several of the design principles identified by Resnick et al. (2005), including: supportive of many levels of learning, supports collaboration, and makes a mathematical concept very simple and accessible.

Tinkering

Tinkering experiences primarily involve taking things apart and engaging in changes and/or modifications to existing objects. These objects can be building blocks, puzzles, digital or electronic simulations, programming code, and so forth. During tinkering, students are not constructing an object, digital or otherwise, but rather exploring changes to existing objects and then considering the implications of the changes. These experiences may require students to use some of the foundational concepts and skills learned during unplugged experiences, but new concepts and skills may also

being introduced. Like unplugged experiences, tinkering experiences can occur prior to any of the other experiences or in sequence following unplugged experiences if necessary for novice learners.

The goal of tinkering experiences is to provide a context for exploring incremental modifications, without the additional cognitively demanding challenge of actually building the object. Application, simulation, and problem-solving are the foci. These experiences ultimately explores the "what if" (e.g., What if I change this part of the code? What if I remove this part of the structure? What if I add this to the object?) moments of learning that yield insight, and often more questions (Sneider et al. 2014).

A good example of tinkering is modifying existing computer programming code. Any computer programming when executed or run is ultimately manifested in a physical sense (e.g. sound, lights, movement, etc.). This hands-on aspect of tinkering with computer programs or code allows for students to easily see the connection between changes in the program and the outcome and even allows students to immediately know an error has occurred when the program does not execute or run.

Widely used and known visual-graphic computer programming software for children and novices is *Scratch* (Lifelong Kindergarten Group at the MIT Media Lab 2016; Resnick et al. 2009; Smith and Neumann 2014; Watters 2011a). The concept of "tinkering" underpins this popular software and makes it highly accessible even to young students (Resnick et al. 2009). A block coding method is used whereby students combine programmed blocks (e.g., run, repeat, stop, move, etc.) together to construct an executable code. Scratch is intended to be highly interactive and accessible: "Just click. .. and it starts to execute its code immediately. You can even make changes. .. as it is running, so it is easy to experiment with new ideas incrementally and iteratively" (Resnick et al. 2009, p. 63). While children can use Scratch to "make" during making experiences described next, they can also easily explore pre-existing block codes to investigate CT concepts. Smith and Neumann (2014) found that Scratch supports students by drawing connections between physical actions on the screen and commands and helps them "reason about and predict . . . actions, leading to a more nuanced understanding of the attributes of transformations" (pp. 186–87).

Geometry is ideally explored through software such as Scratch during tinkering experiences. Teachers can select premade coding blocks from the extensive online library to allow students to explore concepts such as properties, attributes, rotations, reflections, and translations by making incremental or small changes to existing programs. One simple example of this is modifying lines of code to produce different outputs. For example, students are provided with the script for a square and are asked to make modifications to the square script to create a rectangle (see Image 1). Tinkering in this activity involves modeling and simulation practices (Weintrop et al. 2016) and supports exploration (Resnick et al. 2009; Resnick et al. 2005).

By tinkering with premade programs, students can focus on the lines of code where the mathematics is most evident. Students can hone in on the variables and math formulae, and through adjusting code, they can immediately discover what their adjustments will do. By doing so, students can appreciate that "coding offers new ways of experiencing, representing and investigating mathematics concepts and relationships" (Gadanidis 2015, p. 162). From a pedagogical perspective, the focus on the "what if" provides a context for conjecturing, problem-solving, generalizing,

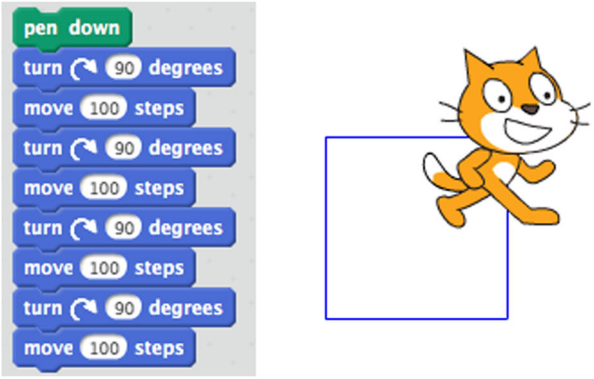

Image 1 Draw a rectangle using Scratch. Source: http://researchideas.ca/wmt/c6b1.html (Reproduced with permission)

and predicting – all which can lead to deeper mathematical understanding. Opportunities for facilitating learning are extensive during tinkering experiences.

We should make clear that our view of tinkering experiences align with many of the sharing practices defined as "remix" by Scratch (see https://wiki.scratch.mit.edu/wiki/Remix). This is different than our "remixing" described shortly. Scratch's remix, for the most part, involves sharing and tinkering with existing code and we see this as requiring a significantly lower cognitive demand level than our view of remixing or even making.

Making

Papert (1980) described learning to consist of building up a set of materials and tools that one can handle and manipulate (p. 173). Papert's views are the essence of making experiences. These experiences are different than tinkering in that objects are fully constructed anew rather than pre-existing. Making experiences, depending on the objects used, require deeper knowledge than tinkering where objects for the most part are already constructed or pre-existing.

In making experiences, students are required to problem-solve, make plans, select tools, reflect, communicate, and make connections across concepts. Often making involves practices such as prototyping and testing. The knowledge and understanding that is developed in this these pedagogical experiences could involve the development of foundational skills but for the most part, makes use of foundational skills instead. Students have the potential to learn as they construct and while they share what they do, what they have made, and how they have made it (Dougherty 2012). A "maker" or "maker space" are other popularized terms which refers to individuals or groups of people who build things and locations where this building occurs.

Making experiences can occur with any objects, including those that are explicitly intended for building such as *Lego* or even household objects. In these instances the making is unplugged. Or, making experiences can occur through computer programing. These experiences can also occur by digitizing objects or through digital fabrication often referred to as "digital making" (Strawhacker and Bers 2015). Digital making refers to the interactions between the virtual and the real world through the use of

intuitive interfaces that use sensors, micro-controllers, motion, sound, light, as well as other tangible materials to interact with the environment (Bowler 2014; O'Sullivan and Igoe 2004).

Sometimes, digital making is also referred to as "tangible programming" (or, physical computing, digital fabrication or graspable user interfaces). Digital making is simultaneously a tangible representation and digital of CT that moves beyond text-based computer programming and coding. Object-oriented programming languages are often used for digital making which counteract the traditional procedural, highly structured and complex, syntax-based computer programming; thus, they are more accessible for students (Govender and Grayson 2008). Object-oriented programming first creates objects through programming where the most abstract case is first developed and then can be subsequently tinkered with or remixed (see next section) to create objects where non-essential characteristics/behaviours can change to get variations. Object-oriented programming languages are proposed to facilitate learning CT concepts such as sequencing, recursion and debugging (Przybylla and Romeike 2014), and to be of use in learning mathematics (Parker 2012; Scarlatos 2006).

Digital making "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" (Wilkerson-Jerde 2014, p. 102). They are a major part of the re-emergence of programing for children and youth because they have the potential to make abstract programming concepts and ideas more physical and accessible for younger children and more amenable to social learning. Digital making also resonates with traditional mathematics manipulatives such as blocks, counters and fraction bars, which have shown to be supportive of young children's learning and creative expression (Strawhacker and Bers 2015, p. 298).

Digital tangibles, the product of digital making, include but are not limited to digital textiles (Lovell and Buechley 2011), digital games, programmable blocks (Kwon et al. 2012), tangible avatars/digital puppets (Liu et al. 2012), reactive materials, robotics (Kazakoff et al. 2013; Sullivan et al. 2013), or simply tangible algorithms. Tangible programming was the key focus of Papert's (1980) seminal contributions to the field. Through Papert's LOGO, 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).

While most research in this new area has been innovation-oriented, a few studies have investigated the influence of digital tangibles on learning. Digital tangibles have been observed to be as effective as graphical user interfaces for learning programming languages, and more helpful in the early stages of learning programming concepts for kindergarteners (Kazakoff et al. 2013; Kwon et al. 2012; Strawhacker and Bers 2015; Sullivan et al. 2013), and for children with certain special needs (Farr et al. 2010). Digital tangibles also promote deeper engagement and prolonged focus when exploring concepts and have been shown to be more appealing to children than solely coding (Horn et al. 2012, p. 379).

Digital tangibles have also been linked to improved interest levels of first year computer programming students (Corral et al. 2014) and in-service and pre-service computer science teachers (Govender and Grayson 2008). Bers and Horn (2010) observe that young children as young as four years old are able to understand the

basics of programming and can build and program simple robotic manipulatives through digital tangibles,. Digital tangibles were also 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 et al. 2012). In this study, however, the levels of understanding about programming concepts gained through tangibles and graphical interfaces did not differ.

Several tangible digitals are adaptable to teaching mathematical concepts such as counting, volume, surface area, location, movement, measurement, time, distances, angles, and so forth. The use of digital tangibles creates a context that is both exploratory and social. Students are able to program digital objects, such as robots, digital blocks or digital characters, to physically embody concepts. Sphero (2016), a ball-shaped robot, for instance, that can be programmed to move around a space (i.e., classroom, table-top, obstacle course, etc.) using several open source programming applications that utilize a visual-based block programming and are downloadable onto mobile devices such as tablets and cell phones (see Image 2). Programming movement for Sphero involves thinking about angles, lengths, times, distances, measurement, as well as proportional reasoning. Students could practice transformation geometry concepts by programming Sphero to navigate a maze marked on the floor, in manner similar to navigating a maze in a digital object on a screen.

Another example of digital making can be seen with the use of Arduino (2016) technology. According to Hughes et al. 2016 in this same special issue, Arduino "is a microcontroller-based digital circuit making kit along with a coding environment for writing programs that can be transferred to the microcontroller" (p. 6). An example of a making activity that has deep mathematical knowledge is programming patterns on a bicolour grid using the Arduino to illuminate LED lights (Hughes et al. 2016; Yiu 2016; see Image 3). The activity involves translating patterns using the Cartesian coordinate system. This would be an example of CT that involves computational problem solving practices, and systems thinking practices (i.e., incorporate loops) in order to make the LED lights illuminate (Weintrop et al. 2016).

Image 2 Sphero coded to complete a path marked on the floor

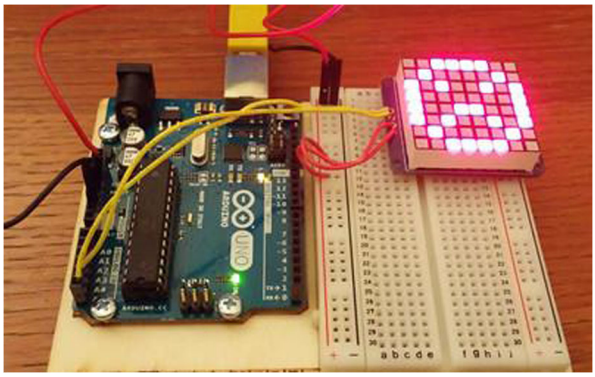

Image 3 Arduino board programming patterns. Source: http://researchideas.ca/mc/article-1-title-recent-issue/arduino-math-patterns-on-an-led-matrix/ (reproduced with permission)

As Hughes et al. (2016) point out, making with the Ardunio has predominantly a very "high" floor and more research is necessary to understand and develop tools that allow for both a "low" and "high" floor. This is consistent with our view of making experiences as having a higher cognitive demand level than the unplugged or tinkering experiences. This also illustrates that sometimes the digital tool used may be too cognitively demanding for concepts that may be more simply understood through the selection of other tools and experiences.

Remixing

Remixing experiences refer to the appropriation of objects or components of objects for use in other objects or for other purposes. Remixing is sometimes referred to as "hacking" or "digital sampling." Remixing, as used in this context, is more than just sharing an object or sharing and making a minor modification where the objects produced are "largely derivative, uninteresting, and of poor quality" (Dasgupta et al. 2016, p. 1439). Remixing experiences involve sharing (intentionally or through "hacking") an object and modifying or adapting it in some way and/or embedding it within another object to use it for substantially different purposes. Remixing requires a significant level of proficiency to identify a useable object and then adapt and modify it to suit new purposes. To remix, the student must be able see useable objects within other objects. In our view, this is the most cognitively demanding task and one that suggests substantial advancement along a zone of proximal development. Consequently, remixing experiences may fall sequentially after the other three or various combinations or iterations of the other three experiences in the CTPF.

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. 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). Perceptions of remixing have evolved from negative notions of others "stealing" ideas, to a more open-minded view of a community of learners that "feel proud, not upset, when their projects are adapted and remixed by others" (Resnick et al. 2009, p. 65). Online coding communities offer a

clear benefit for students in that it enables them to build upon others' work and also provides a forum for giving credit to others when they do so (Watters 2011b). Remixing offers important opportunities for engaging in discussions with students about the ethical issues associated with appropriation, sampling, hybridity and the responsible use of digital technologies (Colton 2016). These opportunities for students can serve to interrupt strong utilitarian views regarding computational thinking and broaden critical discussions about the consequences of widespread computation. These opportunities can also help build digital citizenship.

Dasgupta et al. (2016) tested the belief that remixing, particularly the appropriation of programmable media in Scratch promotes learning (Lifelong Kindergarten Group at the MIT Media Lab 2016). Using data from over a million Scratch user accounts they carefully show that "users who remix more often have larger repertoires of programming commands [and that] exposure to computational thinking concepts through remixing is [positively] associated with increased likelihood of using those concepts" (p. 1438). They suggest that designing deliberately constructed remixing experiences together with increased structure in an informal learning environment such as Scratch might register larger effects. They also note that "remixing may be more effective at promoting engagement with some concepts, like loops, than others, like operators and data" (p.1446). Consistent with our view of remixing as developmentally more complex for students, these researchers acknowledge that, "remixing requires prolonged engagement, and remains extremely difficult in informal learning environments. .. and that enormous work remains to realize its potential" (p.1446).

Davis (2014) provides a powerful unplugged example of remixes in mathematics education by using conceptual analogy. Working with teachers on the topic of mathematical operations, Davis had teachers use their prior experiences and familiarity with addition, subtraction, multiplication and division "grids" to generate an exponentiation grid from which a rich conversation arose around the properties of the operation. This creative remixing, and not mere re-using of a popular artefact illustrates the way in which remixing opens up spaces for recursive elaboration.

Historically, there is evidence to suggest that mathematicians routinely use remixing in advancing the discipline. The development of the Hindu-Arabic numeral system and the concept of angle (Matos 1990, 1991) are two historical examples. More recently, the work of mathematician James Maynard on prime numbers provides a good example of remixing in mathematics (Freiberger 2016). Freiberger (2016) notes in describing the process undertaken by Maynard that:

> This kind of thing is very symptomatic of modern mathematics . . . You're quite often tying together lots of different areas of mathematics, and maybe draw inspiration from physics and engineering as well, which normally you wouldn't have thought as associated at all. If you say 'prime numbers' nobody thinks music and nobody thinks cuboids with points in them. But it turns out that all of these things can be related to each other, and this combination of different techniques has proved very useful in mathematics. (Paragraph 14)

It is likely that part of the underlying reason that such approaches have become "symptomatic of modern mathematics" lies in the availability of rich computational environments for doing mathematics.

CTPF Example

We use the concept of a "variable" to provide a threaded example of the four experiences in the CTPF. By definition, a variable is able to be changed or adapted in ways that shift the outcome of a formula or process. One example of an unplugged experience that uses variable could be any non-computerized game (e.g., board game) where rolling dice determines the movement of the players. While the lower and upper limits of the variable are fixed to the values on the dice, the amount nevertheless has the potential to be varied with each roll. Similarly, games that involve pulling cards or landing on a specific square on a board game that then have some unknown or random instruction or value that shapes the play are also examples of unplugged experiences that involve the use of variables. The cards or the spot on the game board are tangible objects that are representing the concept of variable. The extent to which the child, or the player, recognizes the variable in the particular experience or object depends on the degree to which the action which represents the variable is made explicit.

The same concept of variable is easily illustrated in a tinkering experience were students change a component of prewritten code to modify the output. For example, prewritten code may involve moving a character by a certain amount of spaces towards an end object. The number of steps can be manipulated in the code by changing the value of the variable within the coding structure. Linking the variable to measurement, angles, number sense, and so forth, additionally connects the learning to mathematical content. Tinkering experiences can also simultaneously be unplugged. For example, a preexisting structure that moves water, sand or marbles, vehicles, for example, can have the directional path altered by changing switches. The direction of the switch is the variable component of the path. A making experience would involve building some sort of code or other tangible object where a choice mechanism that can be manipulated is included. A switch component in electronics that is activated differently depending on the prior event is an example in a making experience.

Finally, developing complex code through remixing might involve copying or hacking codes or components of a structure explicitly for their use of variables (or choice inducing structures) which could also be further manipulated. In this final example, the variable is likely embedded in some other coding structure that makes the overall "hack" a more efficient process than rewriting the code containing that variable. Earlier we provided examples of how formulae, which usually include variables, are appropriated in mathematics to solve problems. In remixing, the learner (or "hacker") must understand both the utility of the variable and the current and future context in which the variable performs the change in order to increase the utility of use. They must be able to "see" the variable and its outcomes and this in our view represents a higher level of cognitive challenge.

These examples of the concept of variable developed using the CTPF represents a sequential approach and also illustrates instances were some experiences are simultaneously unplugged experiences. The CTPF is useful particularly for novices to CT in that it potentially provides a starting pedagogical structure. It may be the case that the starting point for some teachers may be at the tinkering stage. Relating the concept back to unplugged experiences, without actually engaging in those experiences may also be useful. Likewise circling back and forth between these experiences or simultaneously incorporating unplugged versions of the experiences may also be useful and necessary.

We propose that this will and may vary depending on the concept, the level of knowledge of the students, and likely the level of pedagogical comfort of the teacher. The degree to which such instances occur and how such a shuffle would impact teaching and learning would be an important area of further research.

Conclusions

In this paper we propose and introduced the CTPF that consists of four experiences: (1) unplugged, (2) tinkering, (3) making, and (4) remixing. Constructionism (Papert 1987), and Vygotsky's and social-cultural theory, specifically Vygotsky's (1978) zone of proximal development underscore our proposed framework. CTPF is not intended to be prescriptive or necessarily sequential and, while we focus our examples in mathematics education, we propose the framework can be applied more broadly to include other disciplines.

The proposed CTPF is intended to provide a preliminary lens for structuring teaching and learning for students by considering how to teach CT rather than what to teach or practices associated with CT that have been outlined by others (Brennan and Resnick 2012; Resnick et al. 2009; Resnick et al. 2005; Weintrop et al. 2016). Across all of the proposed experiences, teachers should ideally construct lessons and learning in such a way that supports a low floor and high ceiling (Papert 1980) but also with what Resnick et al. (2009) call "'wide' walls'". .. supporting many different types of projects so people with many different interests and learning styles can all become engaged" (p. 63).

A significant challenge going forward will be the training of teachers, both in-service and pre-service (Kafai 2015). The proposed CTPF may be tremendously useful for teachers and learners who may have limited understanding of CT. Given that this interest in CT is a relatively new resurgence, it is not likely that teachers encountered CT education/training during their initial teacher training (Curzon et al. 2014). Consequently, there is a "knowledge and skill gap, especially with respect to subject-based pedagogy" (Curzon et al. 2014, p. 89). It may be that in-service and pre-service teachers are at the exact same starting point in terms of knowledge and implementation and, consequently, some synergies in teacher development may be possible.

The proposed CTPF may also be a useful structure for supporting teacher development. Curzon et al. (2014) suggest that workshops for teachers should also begin with unplugged experiences so that the knowledge of computer science concept gap is narrowed and they are more confident with using the tools in their classroom. A sequential approach to the CTPF, beginning with unplugged experiences and working towards remixing experiences, may be particularly useful for novices. However, a sequential approach is not essential; rather, ensuring exposure to *all* four experiences in the CTPF may be more optimal in terms of supporting the development of CT.

Given the preliminary nature of our assertions about the CTPF, there are numerous opportunities for further research. In addition to focused studies related to mathematics and CT, cross disciplinary research pertaining to each of the four experiences is necessary in order to understand the implications for teaching and learning beyond traditional applications in computer science. Research is needed to explore the full CTPF and its utility for learning of CT. Such research could explore the efficacy of the

framework in relation to the development of teacher- and student-level CT knowledge and teacher CT professional development.

Acknowledgements This paper was based on the Namukasa et al. (2015) working group report from the Math + Coding Symposium, Western University, London, Canada. We would like to acknowledge the early contributions of Yasmin B. Kafai and Laura Morrison, and the feedback from George Gadanidis. This research was funded by a Social Sciences and Humanities Research Council grant to George Gadanidis, Donna Kotsopoulos, and Immaculate Kizito Namukasa.

References

Arduino (2016). Arduino. Retrieved August 14, 2016, from https://www.arduino.cc/.

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

Berry, M. (2013). *Computing in the national curriculum. A guide for primary teachers*. Bedford: Computing at School.

Bers, M. U., & Horn, M. S. (2010). Tangible programming in early childhood. In R. Berson & M. J. Berson (Eds.), *High tech tots: Childhood in a digital world* (pp. 49–69). Charlotte: IAP.

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

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking. Paper presented at the American Educational Research Association*. Canada: British Columbia.

British Columbia Government. (2016). $6 million to help connect students with coding, new curriculum and computers. Retrieved August 11, 2016, from https://news.gov.bc.ca/releases/2016PREM0065-000994.

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

Colton, J. S. (2016). Revisiting digital sampling rhetorics with an ethics of care. *Computers and Composition, 40*, 19–31.

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.

Dasgupta, S., Hale, W., Monroy-Hernandez, A., & Hill, B. M. (2016). *Remixing as a pathway to computational thinking*. Paper presented at the Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing.

Davis, B. (2014). Toward a more power-full school mathematics. *For the Learning of Mathematics, 34*(1), 12–17.

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

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.

Freiberger, M. (2016). Primes without 7 [Electronic Version]. *+plus magazine*. Retrieved August 11, 2016, from https://plus.maths.org/content/missing-7s.

Gadanidis, G. (2015). Coding as a Trojan horse for mathematics education reform. *Journal of Computers in Mathematics and Science Teaching, 34*(2), 155–173.

Gadanidis, G., Hughes, J. M., Minniti, L., & White, B. J. G. (2016). Computational thinking, grade 1 students and the binomial theorem [electronic version]. *Digital Experiences in Mathematics Education*. doi:10.1007/s40751-016-0019-3.

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-study.

Horn, M. S., Crouser, R. J., & Bers, M. U. (2012). 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, Math + Coding Symposium*. Western University: Western University. London.

Hughes, J., Gadanidis, G., & Yiu, C. (2016). Digital making in elementary mathematics education [electronic version]. *Digital Experiences in Mathematics Education*. doi:10.1007/s40751-016-0020-x.

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.

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., Kim, H.-S., Shim, J.-K., & Lee, W.-G. (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.

Lifelong Kindergarten Group at the MIT Media Lab. (2016). Scratch. Retrieved August 11, 2016, from https://scratch.mit.edu/.

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*. New York: Paper presented at the Proceedings of the 8th ACM conference on Creativity and Cognition.

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.

Matos, J. (1990). The historical development of the concept of angle. *The Mathematics Educator, 1*(1), 4–11.

Matos, J. (1991). The historical development of the concept of angle (2). *The Mathematics Educator, 2*(1), 18–24.

Namukasa, I. K., Kotsopoulos, D., Floyd, L., Weber, J., Kafai, Y. B., Khan, S., et al. (2015). From computational thinking to computational participation: Towards achieving excellence through coding in elementary schools. In G. Gadanidis (Ed.), *Math + coding symposium*. London: Western University.

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

O'Sullivan, D., & Igoe, T. (2004). *Physical computing: Sensing and controlling the physical world with computers*. Boston: Thomson.

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

Papert, S. (1987). Constructionism: A new opportunity for elementary science education. Retrieved August 1, 2016, 2016, from http://nsf.gov/awardsearch/showAward?AWD_ID=8751190.

Papert, S., & Harel, I. (1991). *Constructionism*: Ablex publishing corporation.

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

Pierce, M. (2013). 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+.

Province of Nova Scotia. (2015). Minister announces coding as a priority during education day. Retrieved August 11, 2016, from http://novascotia.ca/news/release/?id=20151021002.

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

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.

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

Shodiev, H. (2013). *Computational thinking and simulation in teaching science and mathematics*. Toronto: Paper presented at the Association for Computer Studies Educators Conference.

SITRA. (2014). 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-code.

Smith, C. P., & Neumann, M. D. (2014). Scratch it out! Enhancing geometrical understanding. *Teaching Children Mathematics, 21*(3), 185–188.

Sneider, C., Stephenson, C., Schafer, B., & Flick, L. (2014). Exploring the science framework and NGSS: Computational thinking in the science classroom. *The Science Teacher, 38*(3), 10–15.

Sphero. (2016). Retrieved August 11, 2016, from http://www.sphero.com/about.

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.

The White House. (2016). Computer science for all. Retrieved August 11, 2016, from https://www.whitehouse.gov/blog/2016/01/30/computer-science-all

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.

Vygotsky, L. S. (1978). *Mind in society*. Cambridge: Harvard University Press.

Watters, A. (2011a). 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/.

Watters, A. (2011b). Scratch: Teaching the difference between creating and remixing 2015, from http://ww2.kqed.org/mindshift/2011/08/11/scratch-teaching-kids-about-programming-teaching-kids-about-remixing/.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., et al. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127–147.

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.

Yiu, C. (2016). *Using an Arduino - coding a bicolour LED grid to create math patterns [electronic version]* (p. 1). Math + Coding 'Zine: Exploring Math Through Code Retrieved August 16, 2016, from http://researchideas.ca/mc/article-1-title-recent-issue/arduino-math-patterns-on-an-led-matrix/.