Journal of Pedagogical Research
Volume 7, Issue 2, 2023
https://doi.org/10.33902/JPR.202318546

Research Article

Unplugged activities as a catalyst when teaching introductory programming

Bhagya Munasinghe[1], Tim Bell[2] and Anthony Robins[3]

[1]*Wayamba University of Sri Lanka, Sri Lanka & University of Canterbury, New Zealand (ORCID: 0000-0003-2596-3519)*
[2]*University of Canterbury, New Zealand (ORCID: 0000-0002-0148-0857)*
[3]*Department of Computer Science, University of Otago, New Zealand (ORCID: 0000-0003-1473-1683)*

An unplugged approach to teaching enables students to explore Computational Thinking without using a computer. It might appear that if students are to learn programming, they should focus on computer-based work; however, it appears that using "unplugged" activities before engaging in computer-based coding (programming) activities for each unit of work leads to better outcomes for students in the same amount of time. In this paper we explore why this could be the case, by reviewing literature that reports on these experiences, and also using different theoretical lenses (Notional Machines, Semantic Profiles, and the Zone of Proximal Development) to analyse how the combination of experiences can engage students. We also explore how the approach integrates with mathematics education.

Keywords: Computational thinking; CS Unplugged; Programming

Article History: Submitted 16 October 2022; Revised 16 March 2023; Published online 10 June 2023

1. Introduction

"Unplugged" is a pedagogical approach for introducing students to computer science without having to involve computers (Bell & Vahrenhold 2018). Its genesis in the 1990s was around introducing primary school students to computer science, and included giving these students deeper mathematical experiences that relate to advanced ideas from Computer Science (CS), such as graph algorithms and tractability, as well as fundamental ideas, such as data representation, sorting algorithms and data compression (Bell, Rosamond & Casey 2012). The unplugged approach (and the resources shared through the "CS Unplugged" website) have since become widely used to support Computational Thinking (CT) teaching at all levels of K-12.

Unplugged activities are generally presented as games, magic tricks and puzzles, but are designed to exercise the kind of thinking that computer scientists engage in, such as finding algorithms and data representations that will work effectively on a computational device, and that will work well for the person who is using the software on that device (Bell, Rosamond & Casey 2012, Bell & Vahrenhold 2018). For example, the speed of different algorithms for the same task is explored through searching algorithms (sequential search, binary search and hash tables) that have been adapted to a format that is engaging for young students, but still demonstrates the same performance issues and benefits that would be observed if implemented in a computer program.

Address of Corresponding Author

Tim Bell, PhD, Department of Computer Science and Software Engineering, University of Canterbury, New Zealand.

✉ tim.bell@canterbury.ac.nz

How to cite: Munasinghe, B., Bell, T., & Robins, A. (2023). Unplugged activities as a catalyst when teaching introductory programming. *Journal of Pedagogical Research, 7*(2), 56-71. https://doi.org/10.33902/JPR.202318546

One form of this is cards with values face-down on the table, where the students know that the cards are in increasing order of value, but they must try to minimise how many they look under to find a particular value. This naturally leads them to using a variation of binary search, and opens the possibility of exploring its performance, for example, if the number of cards was doubled (it takes almost no extra time), which can be extrapolated to searching thousands or even millions of items very efficiently. Other activities uncover everyday applications of computer science ideas, such as how check digits work in product codes, leading to the general idea of error detection and correction.

These unplugged activities can be a prelude to programming, since they are well defined and students understand the mechanisms through engaging with them physically. This has led to "plugging it in" activities, which provide students with computer programming challenges based on the unplugged activities they have been doing.

Another style of unplugged activity is directed at understanding elements of programming, such as sequence, loops or variables. These ones also lead directly to programming activities, but the focus of the learning is on programming language elements, rather than the bigger picture of issues addressed in computer science. This kind of activity is explored further in Section 3 below.

The "unplugged" approach was originally designed in the 1990s for outreach in a classroom when it was unlikely that computers would be available, or where time was limited and would be wasted getting students set up on devices. In many countries now students have much better access to computers, and the "unplugged" approach is being used for teaching curriculum material rather than just for outreach. In this situation, the teacher has more regular time with students, and is better able to develop concepts over a longer period, whereas outreach is more focussed on creating interest in a short period.

Despite the improved availability of computers, unplugged teaching has remained popular - for example, Falkner et al. (2019) report that "Unplugged programming" is used by a similar number of teachers to those who use block-based and text-based programming, even in high schools. This may be partly because teachers with little computing background are more comfortable starting off with familiar materials instead of having to grapple with the perceived - and often real - challenges of working with software, but it also appears to reflect positive learning experiences using this approach. The unplugged approach addressed a need because it could be used in any context, without teachers having to choose software and get it installed, and it saved them worrying that students might get into a situation that the teacher couldn't help them out of.

This carries the risk that a class might never proceed to working with computers, and in particular, it might *prevent* students from having an experience of programming a computer. This was never the intention of the unplugged approach, as it was created as a gateway to the richness of computer science, which includes programming. One reaction to this is to eschew the unplugged approach altogether, and focus *only* on students using devices, which highlights the importance of showing teachers ways of connecting unplugged activities to programming. However, a more compelling reason to combine unplugged and programming has emerged: unplugged has been found to work well to *support* students learning programming (Hermans & Aivaloglou, 2017), where students have gained more from learning programming *after* having an unplugged experience.

Unplugged also remains a valuable tool for introducing teachers to Computational Thinking, particularly if they are sceptical about their ability to understand this new topic (Curzon et al., 2014). It also provides a way to reduce "screen time" for those who are concerned that Computational Thinking might be mainly about being on devices. There are other approaches to introducing students to Computational Thinking, but they need to be considered in the light of who they are available to. For example, robotics can be highly motivating, but may only be available to schools that have ready funding; conversely, web-based systems are accessible to anyone who has a web-browser, including web-only devices such as a Chromebook.

In this paper we address the question of the relative value of using an unplugged approach compared with having students working on computers. Not surprisingly, there is a variety of research that supports the idea of not choosing between the two, but rather, working out how to combine them for the best outcomes. We explore ways of connecting unplugged activities with programming, including some new activities designed specifically to introduce programming concepts such as sequence, variables, iteration and selection; these activities can then be linked directly to programming challenges, including Parsons problems (Du, Luxton-Reilly and Denny, 2020) and the PRIMM approach (Sentance et al. 2019) to develop students' programming skills. We also explore more general "plugging it in" activities that provide programming links for more traditional unplugged activities, such as data representation and algorithms. We analyse *why* the unplugged approach is effective at engaging students by reviewing how it works through the lenses of *Notional Machines*, *Semantic Profiles*, and the *Zone of Proximal Development*. This approach to Computational Thinking is then connected to mathematics teaching.

2. Unplugged Activities and Computational Thinking

The CS Unplugged approach connects well to teaching Computational Thinking (Bell 2018; Bell & Lodi 2019, Caeli & Yadav 2019), and it integrates well with other subjects, particularly mathematics (Moursund 2006; Dorling & White 2015).

There are several definitions of Computational Thinking that share common properties, but focus on different views of computation or education (Curzon et al. 2019). Given our focus on programming here, we will use a definition given by Wing that specifically draws attention to the computation:

> Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent (Wing, 2011).

In a conventional context, the solutions would be represented as computer programs, and the "information-processing agent" is the computer. However, unplugged activities (such as the ones found on the website csunplugged.org) map well to a broader view, where the "information-processing agent" is typically a student who is governed by strict rules of a particular activity, such as only being able to follow the instructions "left", "right" or "forward" in a sequence; or only being able to compare two values and take one of two paths (i.e. an "if" statement); or being restricted to looking under one cup at a time (indexed access to an array).

Although originally used as an outreach tool for experts visiting schools, where the time and resources weren't available to engage students with programming on a computer, it has since become popular for use in the classroom (Bell, Rosamond & Casey, 2012; Bell & Vahrenhold, 2018). Teachers around the world have been expected increasingly to introduce their students to elements of Computational Thinking (Yadav, Hong & Stephenson, 2016), and the unplugged approach has been an easy starting point because it uses familiar materials, and avoids the challenges of working with digital devices (Faber et al., 2017; Falkner et al. 2019). However, this can lead to an "unplugged only" approach, which can lead students to miss the connection to digital devices (Feaster et al., 2011; Taub, Armoni & Ben-Ari, 2012; Wohl, Porter & Clinch, 2015), and miss out on experiencing the power of programming.

Of course, Computational Thinking isn't just about developing programming skills in young students, since they are not directly preparing as programmers, and many may not pursue that as a career pathway. However, it is valuable to develop their self-efficacy and understanding of the concepts and range of knowledge needed to live in a digital world, particularly for younger students for whom education is about developing general skills and knowledge. Pérez (2018) explores the "dispositions" of computational thinking (tolerance for ambiguity, persistence, and collaboration), stressing their relevance across a range of educational domains.

3. Unplugged and Programming

The unplugged computing approach would appear to be antithetical to teaching programming, and might be seen as a compromise that should only be used if devices aren't available (which was indeed the situation when it was developed in the 1990s, and remains the case in some contemporary schools that don't have access to power, let alone computers). Once devices are available, it would be tempting to focus on getting students to write programs (sometimes referred to as "coding"), and this is consistent with a constructionist approach.

However, despite the increasing availability of devices to teach computer programming, the unplugged approach has remained popular. While this could be due to the attraction of working with familiar physical resources rather than having to wrangle devices and software, there is also a strong sense that students are learning something valuable from the unplugged activities (Thies & Vahrenhold, 2016). Another factor may be the widely expressed concern that too much "screen time" isn't good for students' health or learning outcomes; see for example Sigman (2012), Neophytou, Manwell & Eikelboom (2021), Pardhan et al. (2022).

A turning point in the understanding of the value of using unplugged activities compared with teaching programming was the work of Hermans and Aivaloglou (2017), who found that unplugged activities seem to act as a catalyst when combined with programming; their results showed that spending about half of the students' time on unplugged activities and half on programming led to better outcomes than spending the same total time on programming only. In particular, they found that students' level of programming skill was similar with both approaches, but the students' *self-efficacy* and *programming vocabulary* was higher when the programming was prefaced by unplugged activities.

A related observation was made by Wohl, Porter & Clinch (2015), who experimented with delivery of three approaches to teaching computer science to young students (Scratch, Cubelets and unplugged) in different orders and found that students had the highest level of understanding after engaging with the unplugged session. They observed that if unplugged is to be used, students found it most engaging when it was the first of the approaches used, and they observed that working with devices first didn't work as well: "it seems when unplugged is put in context of the more 'technological' sessions it is viewed as being less exciting" (p. 59). They also noted that "after the unplugged session the pupils were most engaged in the concepts of computer science. However, after the Scratch session the pupils had the most relevant ideas for things to try next" (p. 59). This again points to the value of combining the two, to gain both benefits - engagement with concepts, *and* inspiration to implement their own ideas.

This also matches approaches such as "Use-Modify-Create" (Lee et al., 2011) and "PRIMM" (Predict-Run-Investigate-Modify-Make) (Sentance et al., 2019), where students are encouraged to engage with digital artefacts before programming, developing playfulness and curiosity before creating their own programs (Schulte, 2012).

Li et al. (2022) performed a meta-analysis and found that the unplugged approach "should be used more for primary school students", while a more programming centred approach "should be used more for secondary students" (p. 8008). The work reported below is focussed on primary school teachers because that is where introductory programming and computational thinking is taught in the local curriculum, but the discussion should be read with the understanding that it may not apply to advanced programming classes.

We should emphasise that the discussion below assumes that an unplugged activity "before" programming applies to individual concepts on a small scale; for example, students might experience an activity that introduces the concept of a variable, and then they implement that concept in a programming language, so in a series of lessons students are alternating unplugged experiences with programming activities.

As well as creating a context for programming where students have engaged with the subject before having to learn details of a programming language environment, unplugged activities also provide very precise scenarios that students can implement as a program after practising the

algorithms physically, so they are already very familiar with how they work. For example, the "parity magic trick" involves counting how many black and white cards there are, which students may have practised a number of times in order to do the trick successfully, and therefore will be very familiar with what the output is for various inputs, including extreme cases (such as all cards being the same colour). This approach has been made explicit in the "Plugging-it-in" section of csunplugged.org.

4. Unplugged Pedagogy

Here we analyse how the unplugged approach works through three theoretical lenses that have been applied elsewhere to computing education: the Notional Machine (Du Boulay, O'Shea, & Monk, 1981), Semantic Profiles (Maton, 2013) and the Zone of Proximal Development (Vygotsky & Cole, 1978).

4.1. Notional Machines

One way of explaining why unplugged activities seem to have a positive effect on programming skills is through the *Notional Machine* (NM). The Notional Machine is a conceptual computer created by teachers to facilitate learners' understanding of hidden aspects about computers and programs at run-time. It represents something learners can (mentally) interact with so the teacher can draw their attention to hidden aspects of computing. A Notional Machine is implicit in all programming teaching methods (Du Boulay, O'Shea & Monk, 1981; Robins, Rountree & Rountree, 2003; Sorva, 2013).

The Notional Machine is not necessarily taught in programming classrooms or introduced to learners explicitly, but often stays as an abstract model of a computer created by the teacher in the context of teaching (Fincher et al., 2020). Accurate Notional Machines underpin successful performance in Computational Thinking and understanding Notional Machines is a prerequisite for effective teaching of computing (Bower and Falkner, 2015). Attention to Notional Machines becomes important when introducing students to computing via programming, irrespective of it being implicit or explicit in the teachers' or learners' thinking.

Wing's definition of Computational Thinking introduced above uses the concept of an *information processing agent* (which is referred to elsewhere as a computational agent) to perform computation, much like a Notional Machine. The typical goal of Computational Thinking is to produce an algorithm or a computer program. Computational Thinking encourages solutions to be defined rather clearly and implemented easily using a computer, as if they are expressed in the form of a set of instructions to a "computational agent" for processing. Nevertheless, the Computational Thinking definition has provided flexibility for the computational agent to be either human or machine as long as it follows a particular instruction set precisely and blindly, executing the steps without consideration for what the purpose of the program is.

Unplugged pedagogy facilitates conceptualisation and mental model development in learners by providing a physical experience of relevant computing concepts and making them more relatable and reachable for students. Unplugged makes use of fellow students or physical objects to act as computational agents, especially in learning to program. For example, the "Kidbots" activity involves giving instructions to a fellow student (the "bot") to move around a grid drawn on the ground to reach a target square (Figure 1), where all the instructions are written in advance of being executed. Another activity uses a flip book (Figure 2) to represent a variable; the nature of the flipbook enforces the concept that a variable can store only one value, and it can be accessed or updated. In the early stages of learning to program, working with physical computational agents like these allows learners to comprehend aspects of a Notional Machine without having to be exposed to any programming language.

In many cases the Notional Machine is implicit, and not explicitly understood by teachers who are new to computing or programming. In cases like Kidbots, however, the Notional Machine of a physical computational agent of this nature is explicitly visible, and the rules it can follow are

easily articulated and understood. This is very useful with younger learners, where the teaching goal might primarily be developing a reasonably solid understanding of programming concepts, rather than knowing a particular programming language or actual implementation. Teachers can also develop a shared understanding with their students, which can be scaffolded to a much more refined Notional Machine later, both as the learners' as well as teachers' programming knowledge grow. Having tangible computational agents like this enables learners to exercise their understanding (i.e. their mental model), instead of it being hidden in the machine (computer) and being primarily conceptual, giving them deeper insights into the elements of computing. A computational agent facilitates a teacher's consciousness of their own mental model against the Notional Machine they are expected to teach, especially if the teachers are computing novices. Computational agents thus become good stepping stones to scaffolding to a robust Notional Machine.

Figure 1
The Kidbot activity from CS Unplugged

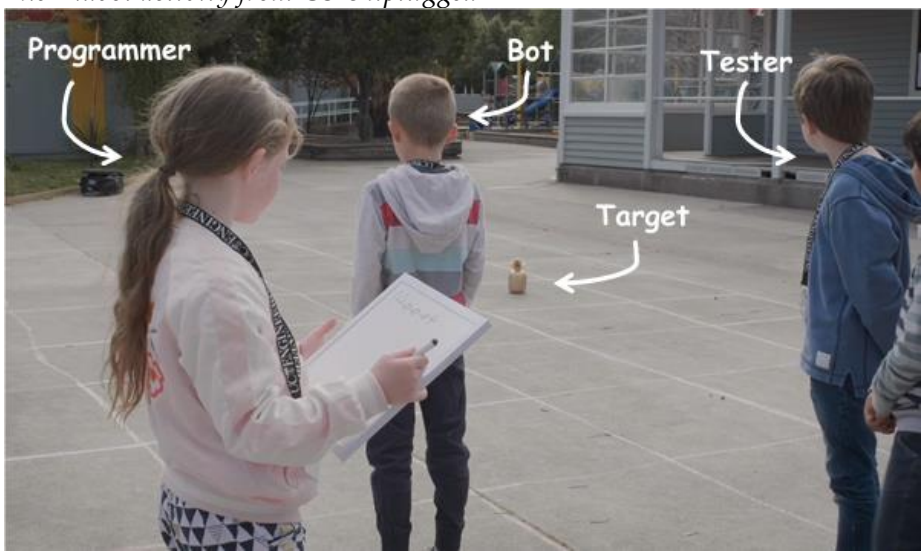
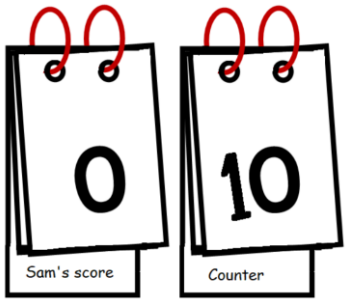


Figure 2
Representing variables using a flipbook



Models of computation are very simple and highly visible within the unplugged style learning activities; they provide simple rules (such as "you can only take one of the two paths based on comparing two numbers", which can later map to the idea of an "if" statement), and are designed to scaffold students to understand genuine computational challenges (such as sorting algorithms, data representation and intractable problems). The tangible nature of unplugged activities enables establishing successful mental models in learners, and moving from unplugged to plugged-in programming activities enables teachers to establish increasingly better mental models that are closer to a robust Notional Machine. It also allows learners to mature their mental models by avoiding (or at least, recognising) possible misconceptions when they build on their prior understanding from the physical experience to understanding the expected Notional Machine. The

pathway through unplugged activities gives early success and a positive experience, before potential complications in a "plugged-in" setting, such as setting up an account, learning the interface of the development environment, and understanding the syntax of a programming language.

This view gives insight into why a combination of unplugged and programming experience can be effective, and how unplugged pedagogy becomes a good scaffolding strategy to learning programming.

4.2. Semantic Profiles

Here we look into reasons for the effectiveness of the unplugged approach through the lens of Semantic Profiles, a concept based on Legitimation Code theory (Maton, 2013). LCT encompasses several dimensions, including the dimension of Semantics, which explores the context-dependence of knowledge (semantic gravity - SG) and the degree of complexity of practices, dispositions and contexts (semantic density - SD). For example, both SG and SD can be relatively weaker (-) or stronger (+) in a continuum of strengths, and vary during a teaching session. Semantic profiling provides a graphical representation of how, during a lesson, meaning shifts between simpler knowledge and more context-dependent knowledge. Semantic waves are formed by shifts in meaning in both directions between the SG+/SD- and SD+/SG- range in a continuum (identified as the *semantic range*), and offer a potential means of traversing this range in classroom practice. These waves identify learning moments, as the teacher and students explore ideas, moving back and forth (in a process referred to as unpacking and re-packing) between students' existing knowledge and new concepts that the teacher wishes to introduce.

Curzon et al. (2020) have already analysed unplugged activities using this approach, and have shown how an unplugged computing activity forms a semantic wave covering a wide semantic range. Here we extend this to professional development classes for teachers where unplugged is used to support teaching programming: using an unplugged activity to introduce a programming concept first and then moving to programming using a computer. This combined approach is also referred to as 'alternating unplugged'. We studied the Semantic Profiles of two different approaches of teaching students to program (i.e. a traditional plugged-in only approach, and an alternating unplugged approach) in two teachers' professional development (PD) courses in introductory programming, mostly novice to programming.

In the alternating unplugged approach, a set of carefully designed unplugged activities were used to introduce distinct programming concepts (sequence, variables and selection). Each activity was followed by a plugged-in exercise in the Scratch programming language, given as a Parsons problem (Parsons & Haden, 2006), where the teachers were given the blocks needed to implement a program, and had to put them together in the correct structure. In the plugged-in only course, the traditional, straight-to-program approach was used to introduce the same three concepts using the same set of Parsons problems, but without using any unplugged activity.

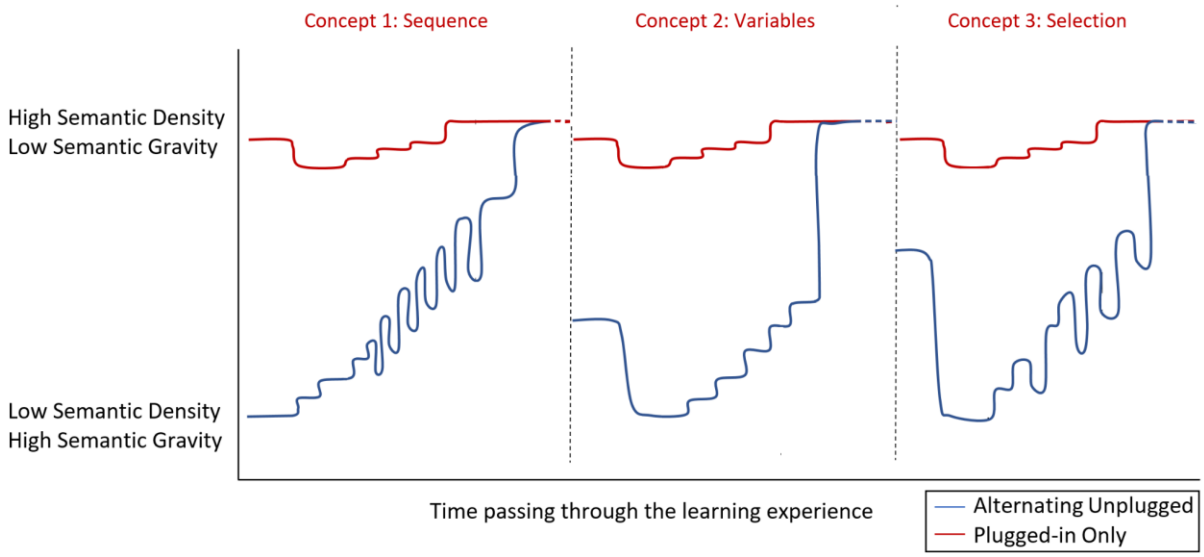
The participants in our study were 152 pre-service teachers and 31 in-service teachers. The participants' self-efficacy towards both teaching Computational Thinking topics and towards computer programming were measured using the same questionnaire instruments, before and after the PD course. The teaching self-efficacy was measured based on the Teaching Self-efficacy Scale developed by Schwarzer et al. (1999) and computer programming self-efficacy was measured based on scales developed by Ramalingam and Wiedenbeck (1998) and Kukul et al. (2017). Based on the teachers' responses to the self-efficacy questionnaire, both approaches could cover a similar amount of content within the same amount of time without hindering the teachers' self-efficacy. However, we observed that with the unplugged combined approach teachers spent comparatively less time programming, yet achieved the same learning objectives.

Figure 3 shows heuristically drafted Semantic Profiles of the two lesson approaches (each over three concepts), based on independent observations from two observers. The observers took notes and discussed with each other how the students had engaged at the various stages of the lessons,

but the curves shown are necessarily only a heuristic representation of what the observers saw in how the lesson was delivered, and how they perceived students were progressing. The Semantic Profile of the traditional, plugged-in only approach, indicated by the red line in Figure 3, is 'high-flattening' (stays in the higher region of the continuum), with very limited connections made to learners' existing knowledge. Comparatively, the Semantic Profile of the alternating unplugged approach covers a much larger semantic range as indicated by the blue lines.

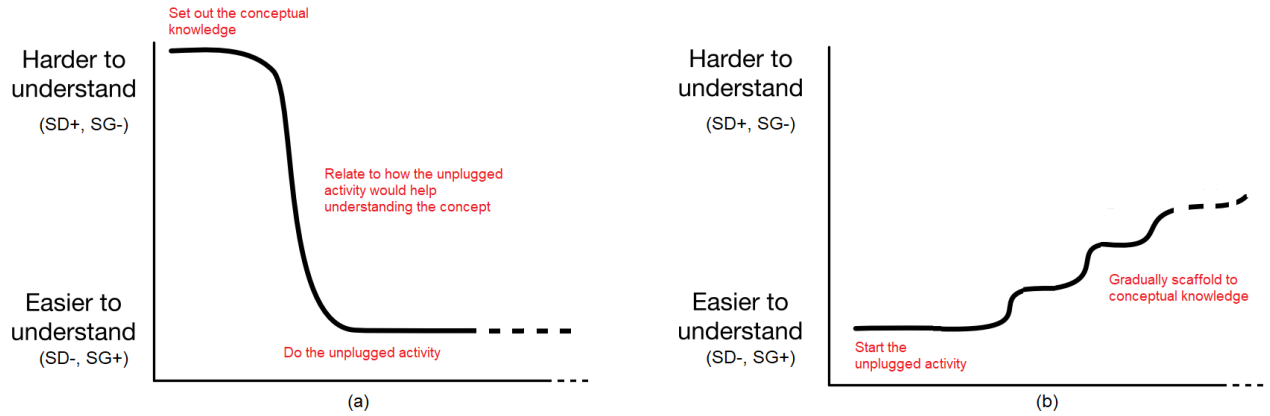
Although an unplugged approach can form a clear wave-like profile that covers a large semantic range (Curzon et al. 2020), had the programming lesson been unplugged *only*, the Semantic Profile would be 'low-flattening' (stays in the lower region of the continuum), without having to complete the learning objective by actually writing a program. This shows how an approach that uses unplugged activities to introduce the concept and then extend it to a plugged-in exercise successfully manages to avoid the two semantic extremes in a programming classroom. Moreover, a similar amount of content and learning objectives can be achieved with less screen time.

Figure 3
Semantic Profiles of introductory programming with and without unplugged examples



We observed that a teacher's approach to delivering an unplugged activity could determine the overall shape of the profile. Figure 4 shows two possible starting points of a Semantic Profile in two different approaches to unplugged in a programming classroom. Figure 4(a) is when a teacher explains concept upfront, then relates to the context. Figure 4(b) is when teacher moves to the unplugged exercise with limited or no reference to computing concepts and draws out ideas to scaffold context to concept as the lesson progresses. In a programming class, the teacher might start with high SD (e.g. introduce the learning objective up front), unpack the ideas before doing the unplugged activity and repack the ideas afterwards, shifting along the SD+/SG- to SG+/SD- continuum, forming a more defined semantic wave. For outreach or other situations, a teacher might start with low SD and high SG (e.g. simply launch into a magic trick without explaining the purpose) and then gradually pack the ideas drawn out from learners with concepts, to bring the learners to a higher SD region, forming a more step-like profile than a wave. However, in either scenario, the unplugged activity causes it to cover a large semantic range.

Figure 4
Starting point of a semantic profile: (a) teacher explains the concept upfront, then relate to context (b) teacher moves to doing the activity and draw out ideas to scaffold context to concept



4.3. Zone of Proximal Development

Unplugged computing uses students' playful engagement in physical activities appropriate to their abilities and understanding to explain computing concepts in a simple manner. Therefore, the learning experience during an unplugged computing activity can also be seen as very similar to the learners' development described by the concept of Zone of Proximal Development (ZPD). The ZPD is described as the distance between individual performance and assisted performance, thus a knowledge development zone in which a student operates with assistance of a more knowledgeable other (MKO), bringing them along a learning process that utilises their level of competence and/or skill to learn challenging knowledge and/or difficult content (Vygotsky & Cole, 1978; Vygotsky, 1986).

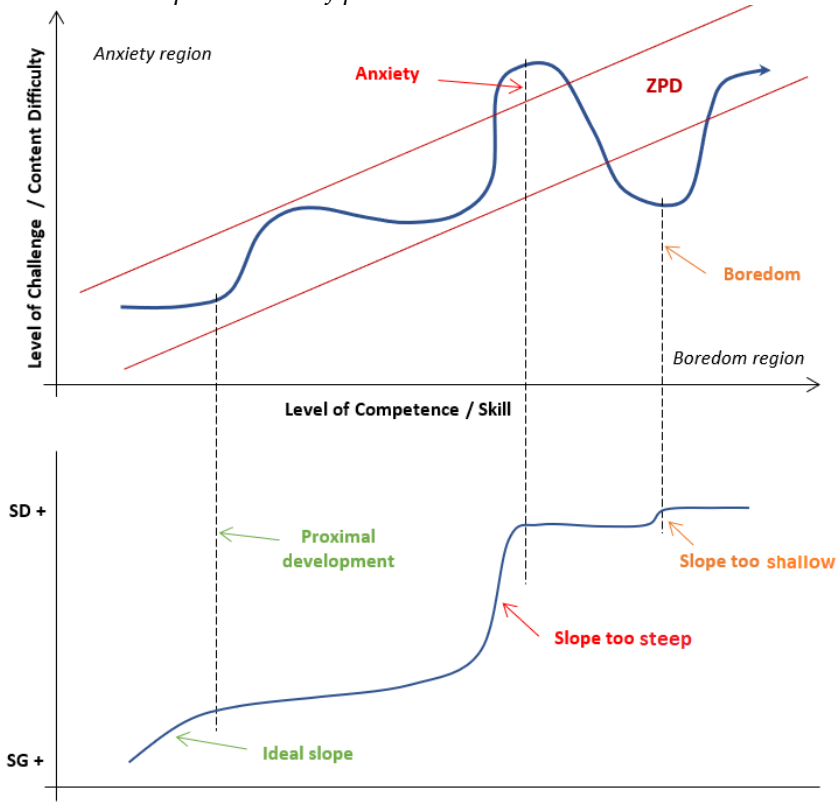
During an unplugged computing activity, students, most often novice or beginner, use their existing knowledge to learn or understand a computational concept by linking their everyday knowledge into the technical context introduced, with the guidance of a teacher (MKO). The teacher guides the learners by simple hints or posing simple questions. Peer learning moments that involve a knowledgeable peer during a learning process are also very common in an unplugged computing activity. Both of these influences are central to the ZPD (see further discussion in Section 5 below). Thus, engaging in an unplugged computing activity ideally situates the learner within the ZPD, providing an environment both for successful learning and for fostering effective dispositions.

When studying the behaviour of the Semantic Profiles of unplugged lessons, particularly in learning programming, heuristically the ZPD can be seen as related to the rate of change of a Semantic Profile of the activity, essentially shifting students back and forth between existing and new knowledge while learning a programming concept. The unplugged activity is designed based on knowledge already familiar to the student (therefore high in semantic gravity) and computing concepts introduced are mostly technical and challenging (high in semantic density) for them. Therefore during unplugged computing lessons, traversing along the semantic range over time inevitably forms semantic waves, as well as situating the learning process well within the ZPD.

The relationship between different slopes in a Semantic Profile to ZPD is shown in Figure 5. The smoother and more gradual upshifts along the semantic range during a learning exercise, the more firmly it situates a learner within the ZPD. Mixing unplugged learning in lessons helps to achieve this overall effect in their Semantic Profiles, while forming many clear local semantic waves during the lesson. Nevertheless, it should be noted that covering a larger semantic range too fast may push the learner into anxiety, and conversely if too slow, into boredom (these terms as used by the original ZPD authors).

Figure 5

Mapping from the concepts of Semantic Profiles (lower graph) to ZPD (upper graph). The vertical dotted lines mark the phenomena of particular interest between the two



5. Unplugged and Mathematics Education

Mathematics education is a much older and larger discipline than computing education, but there are many areas of overlap. The recent *Encyclopedia of Mathematics Education* (Lerman, 2020), for example, has entries on core computing topics such as Algorithmics, Algorithms, Computational/Algorithmic Thinking. There are also entries that strongly relate to CS Unplugged, such as those on Embodied Cognition, Inquiry-Based Mathematics Education, Manipulatives in Mathematics Education, Mathematical Games in Learning and Teaching, and Zone of Proximal Development in Mathematics Education. As already noted, the origin of CS Unplugged was interwoven with concepts from mathematics education. The "Family Maths" programme of the 1970s, and *MEGA-Math* in the 1980s, were seminal influences, and "The Unplugged project began out of an interest in providing engaging and accessible ways of introducing children to big ideas from mathematics and computer science" (Bell, Rosamond & Casey, 2012, p. 400).

Given that the significance and popularity of information technology and computing have driven the emergence of unplugged computing education as a distinct and active field in its own right, with its own growing body of literature, it is probably timely to emphasise that in terms of both its origins and broader context, unplugged approaches have strong overlaps with the general field of mathematics education. The role of computational thinking in mathematics education, for example, is explored in detail and compared with "mathematical thinking" by Pérez (2018), see also extensive reviews in Barcelos et al. (2018) and Nordby, Bjerke & Mifsud (2022). In some cases it isn't even possible to draw a clear line between the disciplines of Computer Science and Mathematics. This is seen particularly in the area of theoretical computer science, where a formal and mathematical approach is taken to computational problems; programs can be seen as a mathematical entity subject to proofs of correctness, data structures are relationships with mathematical properties, automata are imaginary machines that help us reason about the limits of

computation, and so on. In fact, one of the authors of CS Unplugged, Mike Fellows, has said somewhat facetiously that "Computer Science is the rock-and-roll of mathematics" (Bell, Rosamond & Casey 2012, p. 438) - it can be seen as a form of mathematics that might not be completely pure, but is in widespread popular use and has practical applications. Since unplugged activities are done without a computer, many of the activities are inherently mathematical (although not all of them; for example, activities based on human-computer interaction could be seen as essentially psychology or design experiments). In a grade-school context, many of the activities are easily recognisable as exercising arithmetic skills - for example, adding up the check digit in a product code to see if the formula works motivates students to exercise basic arithmetic skills - but then asking them to find a digit that could be changed yet still produces the same check digit moves into the realm of algebra, and even more so when considering how two adjacent digits could be swapped to still produce the same checksum. Many other activities exercise both arithmetic skills and algebraic skills as students work out what is possible, and what "has to be" (i.e. an informal proof), for example, exploring the patterns in binary representations or two-dimensional parity checks.

In the context of the topics explored in this paper, it is no surprise that the pedagogical theories considered in Section 4 have correlates in the mathematics education literature. The Notional Machine in computing is similar to the use of Function Machines in early Algebra, and other forms of explicit calculation strategies and algorithms. In general terms these are examples of models that are deployed for pedagogical purposes (Sorva, 2013; Robins, 2019). Modelling theory distinguishes between conceptual models (objective and publicly shared), and mental models (individual knowledge structures), where mental models may accurately reflect conceptual models to varying degrees (Hestenes, 2010; Sorva, 2013). Both kinds of models have been the focus of considerable attention within mathematics education.

The International Community of Teachers of Mathematical Modelling and Applications (ICTMA) is an "organisation that exists to promote Applications and Modelling (A&M) in all areas of mathematics education - primary and secondary schools, colleges and universities"[2]. It has published several books in the series *International Perspectives on the Teaching and Learning of Mathematical Modelling*[3] including *Modeling students' mathematical modeling competencies* (Lesh, Galbraith, Haines & Hurford, 2010) and *Mathematical Modelling in Education Research and Practice* (Stillman, Blum & Biembengut, 2015). There are many other examples in the broader mathematical literature, see for example the SimCalc project (Roschelle, Kaput & Stroup 2012), a summary would be well beyond the scope of this brief overview! Hestenes (2010) suggests that "Students should become familiar with *a small set of basic models as the content core* for each branch of science, along with selected extensions to more *complex models*" (p. 33). The notional machine in computing education is one such basic model, and as discussed above a progression of Notional Machines can serve as a focus for the development of learners' mental models at various stages of learning.

Although less widespread than the use of models, Semantic Profiles and other concepts from Legitimation Code Theory (LCT) have also been explored in mathematics education. Focusing on the transition from first to second year mathematics courses, Conana, Solomons and Marshall (2022) used an analysis of the Semantic Profiles of typical classes to motivate changes. These included for example "a deliberate focus on widening the semantic range in classroom sessions through referring to specific examples wherever feasible", and "exploring and unpacking a range of representations" (p. 217); in conjunction with other interventions the effect was "significant changes in student's attitudes towards Mathematics learning, as well as in their learning outcomes" (p. 220). Doran (2021) explores the interaction of mathematics, images and language in physics, with a significant focus on "knowledge through mathematics" (p. 172). Using the construct of "semantic plane" (defined by dimensions of density and gravity), and examples based on derivation and quantification, Doran shows how technical meaning can be built in ways that

[2] https://www.ictma.net/index.html

[3] https://www.springer.com/series/10093/books

move from relatively common-sense and empirical meanings to technical and theoretical. Further examples of the use of LCT in mathematics education can be found on the Legitimation Code Theory Website[4]. Dankenbring (2021) reviews applications of semantic gravity in STEM education in the domains of lecture design, syllabus / curriculum design, assessment design, teaching practices and professional development. For a discussion of LCT in STEM education generally see Winberg, McKenna and Wilmot (2020).

Vygotsky's Zone of Proximal Development (ZPD) has such a broad and enduring impact on educational theory and practice that it has almost certainly (whether recognised or not) influenced every teaching discipline. In Russia, disciples of Vygotsky (particularly Leonid Vladimirovich Zankov) developed his ideas into a system of mathematics education. This "Zankov system", which was "verified by large scale pedagogical experiments conducted in 1970s at public schools" is still (as of 7 years ago) in use in Russian schools (Guseva and Solomonovich, 2016, p. 775). In the US in the 1980s and 1990s the National Council of Teachers of Mathematics (NCTM) Standards documents emphasised the importance of social interaction and communication in mathematics education, ideas which "can be traced back to the ideas of Lev Vygotsky" and the ZPD (Steele, 1999, p. 38). In one highly influential study (Goos, Galbraith & Renshaw, 2002) the authors present a three year study of social interaction and metacognitive activity in senior secondary school mathematics classrooms, with a focus on the factors which fostered a "collaborative zone of proximal development" (p. 207); they reported that the impact of "challenge", which is central to the ZPD, emerged as a major "stimulus for mathematical thinking" (p. 218). Roth (2020) briefly summarises literature on the ZPD in mathematics education, including recent developments and the different ways in which the theory has been "reworked" or "reformulated".

In summary, there are many overlaps between unplugged approaches in computing education and related topics in mathematics education. The three theoretical lenses applied to unplugged teaching in Section 4 above, the Notional Machine (conceptual and mental models), Semantic Profile, and the Zone of Proximal Development, are all (albeit to varying degrees) active areas of research in mathematics education.

6. Conclusion

Using unplugged activities before programming can lead to better outcomes for students in the same amount of time. In this paper we have explored why this could be the case, by reviewing literature that reports on these experiences, and also using different theoretical lenses to analyse how the combination of experiences can engage students.

Unplugged activities are a way of exposing learners to an easily accessible and explicit Notional Machine for a simple computational activity, such as instructing a "Kidbot". Through a careful progression of activities, and the introduction of actual programming tasks, the underlying Notional Machines continue to provide scaffolding and support for users as they grow more sophisticated, eventually (ideally) transitioning to the machines underlying full programming languages.

Within a given teaching activity the use of unplugged elements increases the potential semantic range for both learners and teachers, facilitating the use of learners' existing knowledge and concrete experience as a foundation for learning more abstract or technical concepts. Analysing activities in terms of their Semantic Profile raises useful questions. At what semantic level does an activity begin? Does its progression draw on learners' existing knowledge and experience in order to support its goals at the abstract / technical level? Are there sharp transitions that need to be carefully managed?

Aspects of a Semantic Profile analysis map very usefully into the well established context of the Zone of Proximal Development. Is a learning activity as a whole situated within the target learners' ZPD? Does the activity progress too quickly or too slowly, potentially leading to frustration or boredom?

4 https://legitimationcodetheory.com/publications/database/tag/mathematics/

Combining these various theoretical frameworks provides plausible explanations for the efficacy of unplugged activities. It also allows educators to explicitly explore and answer questions that may facilitate the design and delivery of effective learning experiences. Finally, there is of course a significant overlap between unplugged computing and various topics in mathematics education, and the two areas are able to support each other. All of the theoretical lenses used in this paper are also in use, to varying degrees, in mathematics education (and other fields). As such, one of the goals of this paper is to highlight the significance of each field and community of practice to the other, so as to share knowledge on topics of possible interest and support collaboration.

Acknowledgements: This work is partially based on the PhD submission of Bhagya Munasinghe. We are grateful to the anonymous reviewers for helpful suggestions.

Author contributions: All authors are agreed with the results and conclusions.

Declaration of interest: None of the authors have a conflict of interest in this project.

Ethics declaration:

Funding: This work was partially funded by the Accelerating Higher Education Expansion and Development (AHEAD) project, Ministry of Education, Sri Lanka.

References

Barcelos, T. S., Munoz, R., Villarroel, R., Merino, E., & Silveira, I. F. (2018). Mathematics Learning through Computational Thinking Activities: A Systematic Literature Review. *Journal of Universal Computer Science*, 24(7), 815-845. https://doi.org/10.3217/jucs-024-07-0815

Bell, T. (2018). CS Unplugged and Computational thinking. In V. Dagienė & E. Jasutė (Eds.), *Constructionism 2018* (pp. 21–28). Vilnius, Lithuania.

Bell, T., & Lodi, M. (2019). Constructing computational thinking without using computers. *Constructivist Foundations*, *14*(3), 342-351.

Bell, T., & Vahrenhold, J. (2018). CS Unplugged—How Is It Used, and Does It Work? In Böckenhauer, H. J., Komm, D., & Unger, W. (Eds.), *(2018). Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday (Vol. 11011).* (pp. 497–521). Springer.

Bell, T., Rosamond, F., & Casey, N. (2012). Computer Science Unplugged and related projects in math and computer science popularization. In H. L. Bodlaender, R. Downey, F. V Fomin, & D. Marx (Eds.), *The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the occasion of his 60th birthday* (Vol. LNCS 7370, pp. 398–456). Springer. https://www.doi.org/10.1007/978-3-642-30891-8_18

Bower, M., & Falkner, K. (2015). Computational thinking, the notional machine, pre-service teachers, and research opportunities. *Proceedings of the 17th Australasian Computing Education Conference (ACE 2015), CRPIT 160,* 37-46.

Caeli, E. N., & Yadav, A. (2019). Unplugged approaches to computational thinking: A historical perspective. TechTrends, 64(1), 29-36. https://doi.org/10.1007/s11528-019-00410-5

Conana, H., Solomons, D., & Marshall, D. (2022). Supporting the transition from first to second-year Mathematics using Legitimation Code Theory. In *Enhancing Science Education* (pp. 206–223). Routledge. https://doi.org/10.4324/9781003055549

Curzon, P., Bell, T., Waite, J., & Dorling, M. (2019). Computational thinking. In S. Fincher & A. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 513–546). Cambridge University Press.

Curzon, P., McOwan, P. W., Plant, N., & Meagher, L. R. (2014). Introducing teachers to computational thinking using unplugged storytelling. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education --- WiPSCE '14* (pp. 89–92). https://doi.org/10.1145/2670757.2670767

Curzon, P., Waite, J., Maton, K., & Donohue, J. (2020). Using semantic waves to analyse the effectiveness of unplugged computing activities. *ACM International Conference Proceeding Series*, 1-10. https://doi.org/10.1145/3421590.3421606

Dankenbring, C. A. (2021). *Legitimation code theory as an analytical tool for examining discourse within integrated STEM education* (Doctoral dissertation). Purdue University Graduate School.

Doran, Y. J. (2021). Multimodal knowledge: Using language, mathematics and images in physics. In Maton, K., Martin J. R. & Doran, Y. J. (eds) *Teaching Science* (pp. 162–184). Routledge. https://doi.org/10.4324/9781351129282

Dorling, M., & White, D. (2015). Scratch: A Way to Logo and Python. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 191–196. https://doi.org/10.1145/2676723.2677256

Du, Y., Luxton-Reilly, A., & Denny, P. (2020). A review of research on parsons problems. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*, 195–202. https://doi.org/10.1145/3373165.3373187

Du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-machine Studies*, *14*(3), 237–249. https://doi.org/10.1016/S0020-7373(81)80056-9

Faber, H. H., Wierdsma, M. D., Doornbos, R. P., van der Ven, J. S., & de Vette, K. (2017). Teaching computational thinking to primary school students via unplugged programming lessons. *Journal of the European Teacher Education Network, 12*(1), 13-24.

Falkner, K., Sentance, S., Vivian, R., Barksdale, S., Busuttil, L., Cole, E., Liebe, C., Maiorana, F., McGill, M., & Quille, K. (2019). An international comparison of K-12 computer science education intended and enacted curricula. *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, 1-10. https://doi.org/10.1145/3364510.3364517

Feaster, Y., Segars, L., Wahba, S. K., & Hallstrom, J. O. (2011). Teaching CS unplugged in the high school (with limited success). In *Proceedings of the 16th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011*, 248-252. https://doi.org/10.1145/1999747.1999817

Fincher, S., Jeuring, J., Miller, C. S., Donaldson, P., Du Boulay, B., Hauswirth, M., Hellas, A., Hermans, F., Lewis, C., Mühling, A., Pearce, J. & Petersen, A. (2020). "Notional Machines in Computing Education: The Education of Attention". In: *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education.* ITiCSE-WGR '20. Trondheim, Norway: Association for Computing Machinery, 2020, pp. 21–50. https://doi.org/10.1145/3437800.3439202

Goos, M., Galbraith, P., & Renshaw, P. (2002). Socially mediated metacognition: Creating collaborative zones of proximal development in small group problem solving. *Educational studies in Mathematics*, *49*(2), 193-223. https://doi.org/10.1023/A:1016209010120

Guseva, L. G., & Solomonovich, M. (2017). Implementing the zone of proximal development: From the pedagogical experiment to the developmental education system of Leonid Zankov. *International Electronic Journal of Elementary Education*, *9*(4), 775-786.

Hermans, F., & Aivaloglou, E. (2017). To scratch or not to scratch?: A controlled experiment comparing plugged first and unplugged first programming lessons. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (pp. 49–56). ACM. https://doi.org/10.1145/3137065.3137072

Hestenes, D. (2010). Modeling theory for math and science education. In Lesh, R., Galbraith, P. L., Haines, C. R. and Hurford, A. *Modeling students' mathematical modeling competencies* (pp. 13–41). Springer. https://doi.org/10.1007/978-1-4419-0561-1

Kukul, V., Gökçearslan, Ş., & Günbatar, M. S. (2017). Computer programming self-efficacy scale (CPSES) for secondary school students: Development, validation and reliability. *Eğitim Teknolojisi Kuram ve Uygulama*, *7*(1), 158-179.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*(1), 32–37. https://doi.org/10.1145/1929887.1929902

Lerman, S. (2020). *Encyclopedia of mathematics education*. Springer. https://doi.org/10.1007/978-3-030-15789-0

Lesh, R., Galbraith, P. L., Haines, C. R., & Hurford, A. (2010). *Modeling students' mathematical modeling competencies*. Springer. https://doi.org/10.1007/978-1-4419-0561-1

Li, F., Wang, X., He, X., Cheng, L., & Wang, Y. (2022). The effectiveness of unplugged activities and programming exercises in computational thinking education: A Meta-analysis. *Education and Information Technologies*, *27*(6), 7993-8013. https://doi.org/10.1007/s10639-022-10915-x

Maton, K. (2013). Making semantic waves: A key to cumulative knowledge-building. *Linguistics and Education 24*(1), 8–22. https://doi.org/10.1016/j.linged.2012.11.005

Moursund, D. (2006). *Computational thinking and math maturity: improving math education in K-8 schools*. University of Oregon Press.

Neophytou, E., Manwell, L. A., & Eikelboom, R. (2021). Effects of excessive screen time on neurodevelopment, learning, memory, mental health, and neurodegeneration: A scoping review.

International Journal of Mental Health and Addiction, *19*, 724–744. https://doi.org/10.1007/s11469-019-00182-2

Nordby, S. K., Bjerke, A. H., & Mifsud, L. (2022). Computational thinking in the primary mathematics classroom: A systematic review. *Digital Experiences in Mathematics Education*, 8(1), 27–49. https://doi.org/10.1007/s40751-022-00102-5

Pardhan, S., Parkin, J., Trott, M., & Driscoll, R. (2022). Risks of digital screen time and recommendations for mitigating adverse outcomes in children and adolescents. *Journal of School Health*, 92(8), 765–773. https://doi.org/10.1111/josh.13170

Parsons, D. & Haden, P. (2006). Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In D. Tolhurst & S. Mann (Eds.), *Proceedings of the 8th Australasian Conference on Computing Education* (Vol. 52, pp. 157-163). Australian Computer Society.

Pérez, A. (2018). A framework for computational thinking dispositions in mathematics education. *Journal for Research in Mathematics Education*, *49*(4), 424-461. https://doi.org/10.5951/jresematheduc.49.4.0424

Ramalingam, V. & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research, 19*(4), 367-381. https://doi.org/10.2190/C670-Y3C8-LTJ1-CT3P

Robins, A. V. (2019). Novice programmers and introductory programming. In Fincher, S. A. & Robins, A. V. (Eds.), *The Cambridge handbook of computing education research*, (pp. 327– 376). Cambridge University Press. https://doi.org/10.1017/9781108654555.013

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137-172. https://doi.org/10.1076/csed.13.2.137.14200

Roschelle, J., Kaput, J. J., & Stroup, W. (2012). SimCalc: Accelerating students' engagement with the mathematics of change. In M. J. Jacobsen & R. B. Kozma (Eds.), *Innovations in science and mathematics education*, (pp. 60-88). Routledge. https://doi.org/10.4324/9781410602671

Roth, W. M. (2020). Zone of proximal development in mathematics education. In Lerman, S. (Ed.), *Encyclopedia of mathematics education*, (pp. 913-916). Springer. https://doi.org/10.1007/978-3-030-15789-0

Sentance, S., Waite, J., & Kallia, M. (2019). Teaching computer programming with PRIMM : a sociocultural perspective. *Computer Science Education*, *29*(2–3), 136–176. https://doi.org/10.1080/08993408.2019.1608781

Schulte, C. (2012, November). Uncovering structure behind function: the experiment as teaching method in computer science education. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education* (WiPSCE '12) (pp. 40-47). https://doi.org/10.1145/2481449.2481460

Schwarzer, R., Schmitz, G. S., & Daytner, G. T. (1999). *Teacher self-efficacy*. https://userpage.fu-berlin.de/~health/teacher_se.htm

Sigman, A. (2012). Time for a view on screen time. *Archives of disease in childhood*, *97*(11), 935–942. http://dx.doi.org/10.1136/archdischild-2012-302196

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education*, *13*(2), 1-31. https://doi.org/10.1145/2483710.2483713

Steele, D. F. (1999). Research, reflection, practice: Learning mathematical language in the zone of proximal development. *Teaching Children Mathematics*, *6*(1), 38-42. https://doi.org/10.5951/TCM.6.1.0038

Stillman, G. A., Blum, W., & Biembengut, M. S. (2015). *Mathematical modelling in education research and practice*. Springer. https://doi.org/10.1007/978-3-319-18272-8

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS Unplugged and middle school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education, 12*(2), 1-29. https://doi.org/10.1145/2160547.2160551

Thies, R., & Vahrenhold, J. (2016, July). Back to school: Computer science unplugged in the wild. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education. ITiCSE '16* (pp. 118–123). https://doi.org/10.1145/2899415.2899442

Vygotsky, L. S. (1986). *Thought and language* (Revised edition). Massachusetts Institute of Technology.

Vygotsky, L. S., & Cole, M. (1978). *Mind in society: Development of higher psychological processes*. Harvard University Press.

Winberg, C., McKenna, S., & Wilmot, K. (Eds.). (2020). *Building knowledge in higher education: Enhancing teaching and learning with legitimation code theory*. Routledge. https://doi.org/10.4324/9781003028215

Wing, J. (2011). Research notebook: Computational thinking—What and why. *The link magazine*, *6*, 20-23. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wohl, B., Porter, B. & Clinch, S. (2015, November). Teaching computer science to 5-7 year-olds: An initial study with Scratch, Cubelets and Unplugged Computing. In *WiPSCE '15. London, United Kingdom: Association for Computing Machinery*, (pp. 55–60). https://doi.org/10.1145/2818314.2818340

Yadav, A., Hong, H. & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. *TechTrends, 60,* 565–568. https://doi.org/10.1007/s11528-016-0087-7