Authors' Response: Keeping the "Computation" in "Computational Thinking" Through Unplugged Activities

Tim Bell, Michael Lodi

Authors' Response: Keeping the "Computation" in "Computational Thinking" Through Unplugged Activities[1]

Tim Bell
University of Canterbury, New Zealand • tim.bell/at/canterbury.ac.nz

Michael Lodi
University of Bologna & INRIA Focus, Italy • michael.lodi/at/unibo.it

Abstract: The commentaries provide useful questions and responses that help us understand better how unplugged activities serve as scaffolding to engage students in computer science. They help us to consider how activities relate to computational thinking, particularly by connecting the scaffolding in the activities to the limits of computation. This in turn helps us to navigate the somewhat disputed boundary between activities that clearly use computation as it occurs on physical devices, and metaphors that could potentially be misleading.

1. The responses to our target article connecting the unplugged approach with computational thinking have provided valuable avenues to explore. Below we explore the key points raised: the relationship with programming, how Lev Vygotsky's ideas can help us understand and extend unplugged activities, and questions around the transfer of skills.

Relationship between unplugged and programming

2. **Michael Weigend** highlights the opportunity that unplugged activities provide for students to get outside and move around. This is a strong contrast to the stereotypical view of computing classes requiring students to have more "screen time," which can be associated with physical problems as well as being framed as a passive activity (although, in practice, programming can be both very social and far from passive, as students create software out of nothing). He raises the possibility of addressing different learning styles (§7). A variety of styles can be helpful when teaching, and different content can benefit from different modalities, although too much focus on learning styles is not necessarily positive (Willingham, Hughes & Dobolyi 2015: 269). Nevertheless, teachers will readily recognize that a physical break from using a computer helps students to refocus, and switching between different types of activities can maintain their attention.

[1] This is an authors' pre-print version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in Constructivist Foundations 14, 3 (2019), 357–359. https://constructivist.info/14/3/357.bell

3. **Weigend** (§1) and **Juraj Hromkovič & Jacqueline Staub** (§1) both mention CS Unplugged as an *alternative* to programming; this might incorrectly give the impression that the unplugged approach is intended to replace programming in a curriculum, but of course it is only intended as an alternative to consider at each point of the teaching (for example, as the introduction to a lesson), and not as a wholesale replacement. Funding agencies might be attracted to the idea that computing can be taught without computers, but it would be a perverse outcome if the idea that aspects of computational thinking *can* be taught without computers were to get translated into expecting it to be taught *without computers at all*. This does not seem to be the intention of "alternative to programming" in **Weigend**'s response (§1), since he goes on in §6 to remind us that "CS Unplugged is not intended to replace programming but to complement it." This complementary nature is important to convey, but can be hard to communicate if educators and officials want to focus on one approach or another, and not see them as "both… and."

4. Unplugged activities can teach complex CS ideas without the need to master programming in order to implement those ideas, making those ideas accessible to younger students. But ultimately the skill of programming is needed if students are to have full power over digital devices, and this skill requires considerable time on task to master. Mixing both the engineering approach (programming) with a philosophical/scientific approach (unplugged) provides a variety of pathways into the subject, which will appeal to the variety of students who encounter this. It is particularly valuable while the subject is new in schools, and not well understood by students, as the different modes provide ways to address stereotypes and misunderstandings. It is also useful that teachers feel comfortable teaching the subject because of this approach, but it is also important that they do not stop with simple activities under the misapprehension that this is all there is to computational thinking.

Relationship between unplugged, constructionism, scaffolding and guidance

5. Unplugged activities are not inherently constructionist because most of them do not ask for the construction of a personal artifact that is meaningful (both in the cognitive and the affective sense) for the students, as **Hromkovič & Staub** (§10) point out. In the target article, we have already pointed out (§19) that this is the case (and this is also mentioned by **Jane Waite** §4). However, unplugged activities share some intents with Seymour Papert's constructionism (teach abstract concepts with concrete, constructive and kinesthetic activities) and take a constructivist approach, since students (re)construct knowledge for themselves (**Weigend** §4).

6. Open-ended programming projects could provide an alternative opportunity for learning concepts that appear in the unplugged repertoire in a constructionist way, but it would be a matter of chance, if students encounter the ideas without guidance. For example, a student might rediscover binary search or a sorting algorithm as they encounter a need for searching or sorting in a program they are writing, or they may end up using a built-in function to perform this, and seek an explanation for its good or bad

performance. Algorithms such as sequential search are often built into programming languages, and students may find that it appears to work on small samples, but then observe that programs running on larger data sets seem to become unresponsive. A constructive learning approach could lead to better knowledge retention, if it is carefully planned and implemented. However, it is not likely to be suitable for complete novices (Fincher & Robins 2019: 254). **Weigend** (§6) points out the value of understanding some of the issues before implementing a program, and taking advantage of theoretical knowledge to reach a good practical outcome sooner. The suggestion by **Hromkovič & Staub** (§10) to add a constructionist component (the creation of meaningful objects) to CS Unplugged activities is worth exploring.

7. All the questions posed by **Waite** (§§8, 14, 15) are very interesting, but it will require considerable philosophical and pedagogical work to answer them. In Q1, **Waite** asks whether Lev Vygotsky's (1978) "mediators can be used to analyze scaffolding of teaching computational thinking in unplugged activities?" Using Vygotsky's mediators to analyze the scaffolding in unplugged activities is very much in the spirit of how unplugged is intended to be delivered, with the teacher carefully navigating through the zone of proximal development to keep students engaged (**Waite** §§8f).

8. Picking up **Waite**'s Q2, "Can a measure of the 'degree of freedom' afforded by each mediator for each concept introduced be defined?," we note that one way of looking at unplugged activities is to see them as removing a number of degrees of freedom so that the student's experience is in line with what conventional computation is. For example, the searching and sorting activities allow just two values to be compared at a time by hiding values, or only allowing balance scales to compare two values; in the binary representation the two-state nature of bits is enforced with the rule that a card is either fully visible or not. In terms of the learning process, students are initially guided with specific closed questions ("Which of the two values is heavier?"; "Will 16 dots be too many?"), but later this is expanded to more general questions such as "is there another way to represent this value?"

9. Although unplugged activities tend to begin by reducing the degrees of freedom considerably (as noted by **Waite** §5, there are many ways that the experience can be extended where the restriction on the degrees of freedom is removed and students are free to extend their learning in a self-directed manner. Continuing with the binary representation as an example, students could explore related *patterns*; consider how other *types* of data might be represented; investigate *arithmetic* with binary numbers; investigate *cultural* implications of alphabet representation; or consider other *number systems*. **Hromkovič & Staub** (§10) provide further ideas for how students might create and implement new ideas that follow on from the limited degrees of freedom in an unplugged activity.

10. We see a similar tension in learning to program, where some approaches offer open activities (like "create a project that talks about you in Scratch"), and closed ones (such as solving a set of programming puzzles in guided turtle-based languages). The teacher

has the important role of determining how much guidance will benefit the students at any point.

11. **Waite**'s Q3 ("In what ways is an analysis of Vygotsky's mediators useful?") provides us with a valuable lens for thinking about which elements of the activities are essential to enable students to stay within approaches that relate directly to computational systems, and enables us to broaden that thinking to other encounters with computational thinking. It gives us a tool to consider how well the "information processing agent" (Wing 2011: 20) matches the limitations that computational systems have, and this in turn helps us to navigate the somewhat disputed boundary of what CT is and is not (Denning 2017: 37, Fincher & Robins 2019: 516). If the scaffolding keeps learners within the constraints of what computers can do, then we are satisfying the stricter views of what CT is, and if an activity is outside those boundaries, we are able to consider to what extent the metaphor or analogy inherent in the activity is guiding students to ideas that will work when the "information processing agent" is a computer, with its attendant unforgiving constraints. For example, using recipes as an analogy for algorithms (target article §8) means that the information processing agent is a cook, and the lack of constraints on what language can be used in a recipe means that it is further from being an algorithm than, say, the "kidbots" activity (from csunplugged.org/en/topics/kidbots/), where only three commands are available.

Unplugged and transfer of CT skills

12. **Weigend** (§10) poses the question of whether or not students genuinely see the connection between the activities and computational thinking skills. We think it is a good point because we are aware that CS Unplugged activities alone are not guaranteed to transfer computer science concepts and attitudes (target article §36).

13. We also believe that without explicit guidance from the teacher, it is too much to expect students to see the big picture. Exploring binary search can appear to focus on learning one particular algorithm and how to analyze it; however, the purpose is rather to sensitize students to the idea that the algorithm behind a program can have a significant effect on its *scalability*. If a company grows or starts processing more data, the resources required to do the computing required needs to grow only moderately as the size of the business grows. If their software innovation includes a key algorithm that is $O(n^2)$ in the number of customers, then doubling the income base will quadruple the running cost, and the process will fail due to its success in attracting interest! Issues like scalability are important, but to engage with those issues, students need to investigate particular algorithms (perhaps for searching and sorting), and in the classroom there is a risk that the learning could appear to be an exercise in memorizing a catalog of many algorithms to do the same task.

14. The risk is not only that students might lose track of the big picture, but teachers might also become so focused on the details of a curriculum or helping students achieve well in assessment that they put to one side thinking about why the components are there. Therefore, it is important to bear in mind the connection of computational

thinking to computer science (Nardelli 2019: 33). Computer science is an independent scientific discipline with its own fundamental concepts and ideas (Bell, Tymann & Yehudai 2018) and so concepts like abstraction, decomposition, logical thinking, and so on, can and should be encountered in the context of the discipline. This is in line with the historical evolution of the concept, where computational thinking grew from the world of computer science (**Hromkovič & Staub** §§5-7). This relatively young history shows how new views of what should be taught have evolved, and new ideas for tools to engage students have developed in parallel with our understanding of what should be taught.

15. Furthermore, it is strongly debated whether or not it is possible to teach general "higher order thinking skills" (diSessa 2018: 20), because it is not clear whether they transfer from one discipline to another (or even between different areas of a discipline). Education research tells us transfer is difficult, especially between less related domains (Fincher & Robins 2019: 249). We therefore agree that it is not sufficient to practice an idea (e.g., an algorithm or the idea of binary representation) in an unplugged activity to deeply learn and understand it. Similarly, in computer science education, successful transfer from solving programming problems to domain-general problem-solving skills is achieved only when activities are specifically designed to do so (Fincher & Robins 2019: 20).

16. This is why unplugged activities include a reflection phase where students connect the activities with the computer science concepts – the recently updated website guides teachers considerably on this, and also explicitly explains the connections with computational thinking. Moreover, the new version offers "plugging it in activities" (target article §16) where students can use the context of programming to practice the concepts learned.

Conclusion

17. Teaching computational thinking is a new idea in many school curricula, and both teachers and education officials are still gaining experience with it, which means that discussions like this one are valuable for exploring possibilities and avoiding pitfalls. The responses have been helpful in exploring the nuances of how computational thinking can be brought to life for school students. Given the nascent nature of this field, the community needs to remain open to a variety of approaches, and while such discussions help us to identify promising approaches, it is the results from day-to-day classroom practice that speak the loudest. It will be many years until the long-term effects of different educational approaches in computational thinking become clear, and in the meantime, we have an obligation to be thoughtful about what is implemented. We thank the respondents for their contribution to this important discussion.

References

Bell T., Tymann P. & Yehudai A. (2018) The big ideas in computer science for K-12 curricula. Bulletin of EATCS 124. http://bulletin.eatcs.org/index.php/beatcs/article/view/521

Denning P. J. (2017) Remaining trouble spots with computational thinking. Communications of the ACM 60(6): 33–39.

diSessa, A. A. (2018) Computational literacy and “the big picture” concerning computers in mathematics education. Mathematical thinking and learning 20(1): 3–31.

Fincher S. A. & Robins A. V. (eds.) (2019) The Cambridge handbook of computing education research. Cambridge University Press.

Nardelli E. (2019) Do we really need computational thinking? Communications of the ACM 62(2): 32–35.

Vygotsky L. S. (1978) Mind in society. Edited by Michael Cole, Vera John-Steiner, Sylvia Scribner & Ellen Souberman. Harvard University Press, Cambridge MA. Russian original contributing sources written in the 1930s and onwards and published 1960 onwards.

Willingham D. T., Hughes E. M. & Dobolyi D. G. (2015) The scientific status of learning styles theories. Teaching of Psychology 42(3): 266–271.

Wing J. (2011) Research notebook: Computational thinking – What and why? The Link: The Magazine of the Carnegie Mellon University School of Computer Science 6: 20–23. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why