Computational Thinking, between Scylla and Charybdis

Michael Lodi, Simone Martini

Abstract After recalling the evolution of the concept of "computational thinking", we present its meaning in the work of Seymour Papert, who used for the first time the expression in his *Mindstorms*. For him, the technical aspect of "thinking like a computer scientist" (which is the main content of Wing's use of the term) cannot be separated from the social, and affective dimension of building computational objects in an environment rich of computational principles and meaningful for the community. We will argue that only keeping together the technical meaning and the social aspect, CT will be the formidable actor of change that its proposers envisaged for it.

KEYWORDS: Computational thinking; Constructionism; Computer science; Computer science education; Seymour Papert.

January 2019

Michael Lodi, Simone Martini
Dipartimento di Informatica–Scienza e Ingegneria,
Università di Bologna and INRIA, Bologna, Italy.
Mura Anteo Zamboni, 7
40126 Bologna BO
Italy

Michael Lodi: ORCID iD 0000-0002-3330-3089
email: michael.lodi@unibo.it

Simone Martini: ORCID iD 0000-0002-9834-1940
email: simone.martini@unibo.it
tel: +39 051 2094979
Research partially conducted while on sabbatical leave at the Collegium – Lyon Institute for Advanced Studies. Partial support from French ANR project PROGRAMme.

Conflict of Interest: The authors declare that they have no conflict of interest.

Computational Thinking, between Scylla and Charybdis

Abstract

After recalling the evolution of the concept of "computational thinking", we present its meaning in the work of Seymour Papert, who used for the first time the expression in his *Mindstorms*. For him, the technical aspect of "thinking like a computer scientist" (which is the main content of Wing's use of the term) cannot be separated from the social, and affective dimension of building computational objects in an environment rich of computational principles and meaningful for the community. We will argue that only keeping together the technical meaning and the social aspect, CT will be the formidable actor of change that its proposers envisaged for it.

KEYWORDS: Computational thinking; Constructionism; Computer science; Computer science education; Seymour Papert.

1 Introduction

"Computational thinking" (CT) is one of the buzzwords of the moment, in some educational contexts. The modern (and long) wave of this expression started, as it is well known, with a seminal essay by Jeannette Wing (Wing, 2006), who argues that learning to think like a computer scientist would be a benefit for everyone, in whatever profession involved. Despite the vagueness of the proposal, Wing's position was taken as a trampoline for several initiatives to bring computer science into all levels of K-12 education; see, e.g., (Guzdial, 2015; International Society for Technology in Education (ISTE) & Computer Science Teachers Association (CSTA), 2011), which have produced educational material, definitions, even assessment methods. After much hype about the subject, recently, a more detached, and objective, critical review of CT has been produced, also framing it in its historical context (Tedre & Denning, 2016). Under this perspective, CT should be understood *inside the discipline of computing*, as the (scientific and cultural) substratum of the technical competences.

Historically, however, the expression CT was used for the first time by Seymour Papert in 1980, with a different *nuance* of meaning, which should not be forgotten. The analysis of this sense of CT will be the subject of the central section of the paper. Our thesis is that only keeping together both meanings, CT will be the formidable actor of change that its proposers envisaged for it.

Before turning to Papert's CT, however, we summarise some of earlier attempts to identify the concepts that are peculiar to computer science, and which are now

covered under the umbrella of CT. We refer to the lucid (Tedre & Denning, 2016)[1] for a comprehensive historical account and assessment.

2 Prehistory

The end of the fifties and the early sixties are the years in which the field of computing gradually builds its self-understanding as an autonomous discipline (Tedre, 2014). This process goes hand in hand with the need to specify the traits and concepts distinguishing the new discipline from other sciences, like applied mathematics, or physics, or engineering. A first, important, process is the *linguistic shift* of the programming task (Nofre, Priestley, & Alberts, 2014). In the early days, programming was mainly a technological affair (strictly coupled to the technology of the different computers). The emergence (and the need) of computer-independent ("universal", in the terminology of the time) programming languages allowed the expression of *algorithms* in a machine neutral way, thus making algorithms and their properties amenable to a formal study. Programming languages themselves were treated as object of study — from the formal definition of their syntax (Backus, 1959; Backus et al., 1960), to the gradual emergence of a mathematical theory of computation (McCarthy, 1960, 1961), and, later, of a mathematical semantics (Naur, 1966; Floyd, 1967). Bruno Latour, with genial insight, explains in this way the relationship between a new science and its language:

> No scientific discipline exists without first inventing a visual and written language which allows it to break with its confusing past. (Latour, 1986)

The availability of universal programming languages is felt as the opportunity for computing to evolves from its "obscure" past (made of mathematics, cybernetics, logic, physics, engineering, linguistics) and consciously presents itself as the science of algorithmic problem solving, for which the new languages are developed. Of course, this "founding language" should not be identified with a specific programming language. It is an early recognition that the contemporaneous presence of *different* specific languages (at various levels, with various purposes, with various targets) is an asset of the discipline, and that no language will work for all uses[2].

It is in this context that, not later than 1960, Alan Perlis uses the term *algorithmizing* ("quantitative analysis of the way one does things"), classifying it as "part of the basic thought processes" that "everyone should learn [...] sooner or later" (Katz, 1960). For him "students will have a chance to use computers better [...] by virtue of understanding them as general tools to be used in reasoning [...] rather than as devices to solve particular problems". It is one of the earliest recognition of a specific, disciplinary approach to problem solving that would be the result of being exposed to, and having acquired, the competences of that new field, which at that

[1] Peter Denning has been writing several critical papers on CT and its hype, see, e.g., (Denning, 2009, 2017; Denning, Tedre, & Yongpradit, 2017).

[2] See, for instance, (Gorn, 1963) and its insistence on the role of mechanical languages.

same time struggled to be recognised as an autonomous scientific discipline[3]. In one of the many attempts to describe this new science and its boundaries, George Forsythe (first Head of Computer Science at Stanford) comments on the educational value of computer science: "The most valuable acquisitions in a scientific or technical education are the general-purpose mental tools which remain serviceable for a lifetime. I rate natural language and mathematics as the most important of these tools, and computer science as a third" (Forsythe, 1968).

It is especially in the seventies that this line of thought comes to maturity—Computer Science provides "general thinking tools", useful for everyone (recall Perlis' position). Marvin Minsky in his Turing award lecture (Minsky, 1970) has a long section ("developed with Seymour Papert[4]") on mathematics education. The thesis is that the computer scientist has the role, the responsibility, and the competences to "work out and communicate models of the process of education itself." The very last statement of the paper is that the computer scientist "is the proprietor of the concept of procedure, the secret educators have so long been seeking."

Edsger W. Dijkstra, again in a paper discussing the epistemological status of programming in comparison to mathematics (Dijkstra, 1974), observes that programming gives a unique opportunity to master the complexity of a system, which is handled through a "hierarchical composition," where "a single technology" (that of programming languages of different levels of abstraction) encompasses all the levels of the hierarchy. It is this dealing with "mastered complexity" which "gives programming as an intellectual activity some of its unique flavors." The "programmer's agility with which he switches back and forth between various semantic levels" is a sort of "a mental zoom lens".

Among the many other possible citations and quotes, let us conclude with Donald Knuth (one of the stars at Stanford, recipient of the Turing award in 1974, at the age of 36), who is convinced "of the pedagogic value of an algorithmic approach; it aids in the understanding of concepts of all kinds;" "a student who is properly trained in computer science is learning something which will implicitly help him cope with many other subjects" (Knuth, 1974).

The *anonymous* computational thinking that emerges is *the natural sediment of disciplinary learning of computer science*—that which remains behind when all the

[3] A struggle that was going to be long. The first *Computer Science* department of the US was established in 1962 at Purdue University, where Perlis had served in the computation center from 1951 to 1956. Samuel D. Conte, first Head of that department, will recall in a 1999 Computerworld magazine interview: "Most scientists thought that using a computer was simply programming — that it didn't involve any deep scientific thought and that anyone could learn to program. So why have a degree? They thought computers were vocational vs. scientific in nature" (quoted in Conte's obituary at Purdue University, 2002). Next computer science departments to be established would be those at the University of North Carolina at Chapel Hill, in 1964, and at Stanford in 1965. Still in 1967, Perlis, Newell and Simon (three Turing award recipients; Simon will also be a Nobel laureate in Economics) feel the need of a letter to Science (Newell, Perlis, & Simon, 1967) to argue "why there is such a thing like computer science". See also Knuth's reconstruction of the contribution of George Forsythe to this process (Knuth, 1972).

[4] And again, in the introduction: "Papert's views pervade this essay."

technicalities and the definitions of the discipline are long forgotten[5]. Common belief among many computer scientists, as we have seen, it comes often together with the (largely undocumented and unproven) claim that the metacognitive skills gained through programming (or, more generally, through computer science techniques) *transfer*[6] to other disciplines—from the already quoted (Minsky, 1970) and the related (Feurzeig, Papert, Bloom, Grant, & Solomon, 1970)[7] for mathematics, to the far reaching (Mayer, Dyck, & Vilberg, 1986) and (Pea & Kurland, 1984), for which see also the commentary criticism (Salomon, 1984).

However, the impact of this process on actual reform of education or, more generally, on the cultural debate was modest. Computers and computer science were still mainly confined in scientific and engineering milieux, and in large corporations. A situation which did not change much when this anonymous thinking received for the first time its current name.

3 Papert's Computational Thinking, in context

Seymour Papert seems to be the first to use in print the expression "computational thinking" (Papert, 1980, p. 182). Contrary to Wing's use in 2006, however, this single occurrence of CT in *Mindstorms* is by no means an attempt to a definition: "Their [of people using computers for providing mathematically rich activities] visions of how to integrate computational thinking into everyday life was insufficiently developed." It is used *en passant*, after many other "computational" *something*[8]. What is central to *Mindstorms* is not "thinking"—it is rather "constructing", by computational means, concrete versions of abstract mathematical concepts; or, it is building personal mental models to understand the world—computational "environments" ("metaphors", "ideas", etc.) are one of the most effective and economic ways to obtain such models in an autonomous manner. The appeal of the computer is that it provides a concrete reference for the abstract concepts to be understood.

A naive reading of *Mindstorms* may give the impression that it backs the idea of the transfer of (meta-)skills from CT to other disciplines. This seems explicitly stated already in (Minsky, 1970), where Minsky emphasizes his shared view with Papert: "our conjecture [is] that the ideas of procedures and debugging will turn out to be unique in their transferability."

---

[5] We refrain from a historical reconstruction of this folklore expression, often said of "culture". It appears in print at least in 1908, in Gaetano Salvemini's "Cos'è la cultura".

[6] For a discussion about "transfer of skills", see section 6.

[7] However, we will discuss (Feurzeig et al., 1970) in details in the next section.

[8] The adjective "computational" is used 39 times in the book (CT is used only once). Among the expressions used more than once throughout the book we find: c. ideas (p. 17, 121, 145, 155); c. culture (p. 5, 100, 170, 174); c. metaphor (p. 105, 154, 169, 171, 187); c. model (p. 106, 164, 169); c. environment (p. 182 twice, 212);

On this count, however, it is useful to read (Feurzeig et al., 1970)[9], written in the same year of Minsky's lecture, where Papert and colleagues made claims on how "appropriate teaching with a suitable programming language can contribute to mathematics education". Let us review some of them.

The first claim seems indeed to support the idea of CT transferring to general skills, as already noted by (Tedre & Denning, 2016): "programming facilitates the acquisition of rigorous thinking and expression." However, this concept is explained further in the article—the peculiarities of computer programming make it a privileged tool for learning problem solving with an experimental approach[10]. In fact, children have to impose on themselves rigor and precision in instructing the computer—being explicit and precise is not imposed (incomprehensibly) by an enforcement of the teacher, but naturally emerges from the need of being understood by an automated executor with a limited instruction set, which is unable to perform any "human" inference. Briefly: the computer creates an intrinsic motivation to learn by trial, error and debug.

A next claim is that, again, "programming provides highly motivated models" for the so called *heuristic concepts* (e.g. "formulate plan", "separate the difficulties", "find a related problem", "contrast between global planning and formal details of a solution", "sub-goals and sub-problems", "debugging as a definite, constructive, plannable activity", and so on). Note again the emphasis on the fact that programming provides high motivations for learning these concepts. Moreover, note also that many of these ideas are included in modern CT definitions, often as CT "practices" or "approaches" (e.g. those from CSTA, Google, CAS, ScratchEd (International Society for Technology in Education (ISTE) & Computer Science Teachers Association (CSTA), 2011; Google, 2017; CAS, 2014; Brennan & Resnick, 2012)). In (Bender, 2017), the author even states that "heuristics" is the name given by Papert to what today we call CT.

Finally, (Feurzeig et al., 1970) confirm that the purpose of their experiment is to use programming as a foundation for teaching mathematics, rather than teaching programming as a topic on its own (however recognising the importance of this second aim).

In summary, it is not the programming skills which count, or the "algorithmizing" concepts acquired through programming. A reading of *Mindstorms* which takes into account the entire book, and not single, isolated quotations makes clear that the focus is on the use of computers as formidable tools for "addressing what Piaget and many others see as the obstacle which is overcome in the passage from child to adult thinking." "Knowledge that was accessible only through formal processes can now be approached concretely. And the real magic comes from the fact that this knowledge includes those elements one needs to become a formal thinker." Any

[9] (Feurzeig et al., 1970) was written ten years before (Papert, 1980). It reports on the first fifteen months of using the LOGO programming language in teaching mathematics to three classes: second, third and seventh grade.

[10] Authors recognize that is theoretically possible to teach programming as an abstract mathematical concepts, without using computers, but this will cause the loss of the *essential* aspect of hands-on learning.

rendering of Papert's position as a mere transfer of meta-skills would thus be a gross misunderstanding. The outcome that Papert and his group envisage is not the result of a generic exposure to computational concepts and education. In (Papert, 2000), he explains: "In Mindstorms I made the claim that [...] the ability to program would allow a student to learn and use powerful forms of [...] ideas. It did not occur to me that anyone could possibly take my statement to mean that learning to program would in itself have consequences for how children learn and think. [...] Papers were written on 'the effects of programming (or of Logo or of the computer)' as if we were talking about the effects of a medical treatment." The modality of interaction with the computational media is as (and probably *more*) important than its contents.

It is now high time to come back to the quotation about CT, and to read it in its context:

> I have no doubt that in the next few years we shall see the formation of some computational environments that deserve to be called "samba schools for computation." There have already been attempts in this direction [..., but] they have failed to make it because they were too primitive. [...] Their visions of how to integrate computational thinking into everyday life was insufficiently developed. But there will be more tries, and more and more. And eventually, somewhere, all the pieces will come together and it will "catch." [...] They will be manifestations of a social movement of people interested in personal computation, interested in their own children, and interested in education. (Papert, 1980, p. 182).

To understand this surprising reference to Brazilian "samba schools" we need to elaborate on Papert's learning theory. For Jean Piaget (who inspires Papert's educational view) the way we acquire knowledge determines how much it is valid for us. He encourages the "use of active methods which give broad scope to the spontaneous research of the child or adolescent and require that every new truth to be learned be rediscovered or at least reconstructed by the student, and not simply imparted to him" (Piaget, 1973, p. 15). Piaget's *constructivism*[11] will be transformed by Papert into his own learning theory, *constructionism*. It shares with constructivism the idea of active building of knowledge through experience; it adds that learning is especially effective when the learner is involved in the active construction of objects meaningful to her. To construct these objects, she needs *building materials* (concrete or abstract)[12]. Piaget distinguishes between "concrete" and "formal" thinking, the first already present at the age of first grade and consolidated afterwords, the second which does not appear until, say, age 12. Papert argues that "the computer can concretize (and personalize) the formal", thus allowing "to shift the boundary separating concrete and formal." For him, anything can be easily understood, if it can be assimilated to the collection of mental models already present in the learner's mind. This is why one needs "objects to think with", the building bricks of the personal (construction of) knowledge. In the choice of these materials,

[11] A set of psychological and learning theories sharing the idea that knowledge is actively constructed or reconstructed by learners rather than being transmitted to them.

[12] Papert, for example, states that as a child he was obsessed with gears. He always used mental models about how gears work as a tool to understand the world, and even complex mathematical concepts like differential equations.

however, there is not only a cognitive aspect—for the constructionist, at play there is always a fundamental affective component. Papert himself says he was *in love* with gears.

Every student will be obsessed by something different, and here comes the power of computers, their protean ability to simulate and execute every other model, so that they may bring to everyone the building materials she loves most. Finally, the environment where the learning happens is also fundamental. Computers can create a world where, for instance, you "speak mathematical language" (Papert called it Mathland)—like you learn a foreign language by living in a foreign country, you learn deep mathematical concepts by experimenting, and having concrete, practical experiences in Mathland.

Mindstorms is particularly critical towards traditional school systems. Computers and programming will make old schools obsolete and useless, because learning will happen in constructivist environments, which would resemble traditional Brazilian samba schools, so fundamental for the preparation of the Rio Carnival. They are not schools in the traditional western meaning; they are rather clubs ranging from hundreds to thousands of people, from children to their grandparents, from novices to professionals. Members of each school gather every weekend to dance and to meet with friends. All of them dance: the novice learns, the expert teaches, but also practices for harder moves. There is a great social cohesion, a great sense of belonging, a strong idea of having a "common purpose." Although learning is spontaneous and natural, it is also *deliberate*—results of a year of work are spectacular, professional level representations, with references to traditions and with strong political undertones. All of this is present in Papert's reference to "samba schools of computations": environments where children and grown-ups may learn (by doing) the principles of computation, and use them to learn other disciplines, in a computational perspective. Their learning method will be radically different from what is common in traditional schools. No knowledge is transmitted, and pupils will learn because are immersed in an environment whose activities are both rich of computational principles and meaningful for the community.

That Papert's prediction about the revolution in the school system did not materialize, it is a plain truism, and his ideas in Mindstorms have been misunderstood and oversimplified. In retrospect, (Papert, 2000) reminds about the subtitle of the book ("Children, Computers, and Powerful Ideas"), acknowledging that both enthusiasts and detractors focused on the first two elements, forgetting the third, the most important one. Children are able to learn powerful ideas about the world (e.g., mathematical concepts like the idea of "zero", or "probabilistic thinking"), but these ideas have been disempowered by schools, which teach mathematics only through application of formulas, or by proposing problems situated in "fake" contexts that fail to be meaningful for pupils. The real thesis of Papert's CT, is not that "learning to program will *in itself* have consequences on how children learn and think", but that ability to program a computer can help re-empowering pupils[13], and bring to them powerful ideas about mathematics, physics, probability (which are the fields

[13] This is also made clear in the "Introduction to the second edition" of Mindstorms (1993), which critiques the critiques of the first edition based on a supposed claim that "'doing Logo' or 'working

touched upon in Mindstorms). In (Papert, 2000), finally, it is recognized that the change that was hoped for schools in the 1980s will eventually happen because of the increasing dissonance between school and society, and the increasing availability of technologies and ideas needed for the change to happen. We will return on this in Section 5.

The expression "computational thinking" will return many times in Papert's last[14] talk (Papert, 2006), a few months after the publications of Wing's paper. The starting point is that the school system is dominated by graphocentrism, because it uses obsolete technologies — pencil and paper[15]. This reduces knowledge "to the kind of knowledge that can be written down: propositions" — a "propositional thinking" which is good for testing and grading students. LOGO was the first step beyond this paradigm, introducing "procedural thinking": knowledge as instructions, expressed through a programming language. But Papert acknowledges that this is only a first step towards *computational thinking*. One of the main aspect of CT is what he calls "object oriented thinking", not referring to the programming paradigm, but defining it as the "making and understanding of computational objects". These computational objects[16] may of course be used to teach programming, or standard geometry, but their intended role is another: they are objects you can "get to know [...] more like the way you get to know a person". Again: these are objects to think with, in a cognitive *and* in an affective sense. The great contribution of CT should be making "key, big ideas" of mathematics accessible to children, thus allowing to "turn learning upside down": in history, people started using and developing math for concrete aims, and doing so they "developed something called *mathematical thinking*", and only gradually this became the field of formal, pure mathematics. But nowadays in school this process is reversed: we wait to teach mathematical big ideas when pupils are ready to learn the abstractions needed to manage the formal part. With computers, they can start from the applications and gradually go up to abstractions.

4 Wing and CS for all

The twenty-six years between *Mindstorms* and (Wing, 2006) are those of the dramatic rise of the digital society, and of computational science. On one side we see the availability of digital tools to everybody, through the World Wide Web as the single infrastructure through which all different technologies are deployed. On the scientific side, computational tools are no longer the product of only computer

---

with computers' would *cause* change in how children think. [...] Logo does not itself produce good learning any more than paint produces good art."

[14] The very next day, Papert was struck by a motorbike and received a serious brain damage.

[15] The evocative image Papert proposes is that of an alien anthropologist visiting Earth and understanding that all knowledge workers adopted computers as their main work tool — all except students.

[16] The implicit reference is, of course, to LOGO's turtles.

scientists—most scientific disciplines become "computational", stably adding *simulation* as a third component of science, after theory and experiment (Winsberg, 2010). Wing's paper was then published at the right moment, for selling CT to a larger audience than computer enthusiasts. From simple "algorithmizing", in Wing's hands CT becomes a large umbrella for thought processes and techniques covering also natural information-processing, as synthesized in the Aho-Cuny-Snyder-Wing definition: "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."[17] Wing's CT lays in the path well-marked by the early computer scientists (cfr. Section 2), extracting from computer science a list of thinking patterns ("mental tools that reflect the breadth of the field of computer science"), which include efficiency, approximation, recursion, using abstraction and decomposition in problem solving, or exploiting "reduction, embedding, transformation, or simulation" to reformulate a difficult problem. In particular, Wing is careful in stating that "computer science is not computer programming. Thinking like a computer scientist means more than being able to program a computer" (pag. 35). As (Tedre & Denning, 2016) says, "Wing's formulation struck a resonant cord," and around that manifesto several interests coalesced into a movement to bring CT into all levels of K-12 education; see, e.g., (Guzdial, 2015) and (International Society for Technology in Education (ISTE) & Computer Science Teachers Association (CSTA), 2011). At the same time, several well-grounded criticism has been raised (see, for instance, (Mannila et al., 2014) for a balanced review) for ambiguity of definition, non substantiated claims regarding the transfer of meta-cognitive skills from CT to other disciplines, even arrogance. The problems stem when, instead of understanding CT as a cover for the scientific core of computer science, it is viewed instead as a new discipline, which should be taught and evaluated *as such* (Armoni, 2016), (Lodi, Martini, & Nardelli, 2017).

Despite the many misconceptions (Denning et al., 2017), the CT movement has been instrumental for the inclusion of (some flavor of) computer science in general education (*CS for all*: "For everyone, everywhere" (Wing, 2006), page 34). In Great Britain[18], in USA[19], in France[20], the governing bodies of public education have issued instructions for the teaching of computer science in all schools. Most of them have an explicit reference to CT, but never as a discipline independent from computer science as a whole. On the same vein, see also the recent joint proposal of ACM Europe and Informatics Europe for an "Informatics for all" initiative (Caspersen, Gal-Ezer, McGettrick, & Nardelli, 2018), which adopts a two-tier strategy. A first tier takes the form of informatics as a specialisation, i.e. a funda-

[17] This widely quoted definition is referred to only in unpublished work, for instance Wing's http://www.cs.cmu.edu/ CompThink/resources/TheLinkWing.pdf. See also (Aho, 2011), which stresses in particular the role played in this definition by the information processing agent.

[18] https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

[19] http://www.nsf.gov/csforall

[20] http://www.academie-sciences.fr/pdf/rapport/rads_0513.pdf

mental and independent school subject. The second tier would be the integration of informatics with other school subjects.

Still, we also see many, well-published initiatives where CT is identified (more or less explicitly) with "coding" (that is, the last phase of the programming process)[21], an equation that "keeps spreading into the jargon of CS educational research, of CS curricular development (at all levels), of stakeholders such as politicians who determine or affect educational policies, school principals, and school advisors, among others" (Armoni, 2016). That computer science (and hence CT) is much broader than "coding" (indeed much broader than "programming") is something that computer scientists and educators have known for decades (Tedre, 2014)—accepting now the identification would be a significant step back.

5 The digital discontinuity

The school curriculum, in any country, have a big inertia — it changes very slowly, and radical reforms are rare. They happen only when a significant fracture is experimented between the curriculum itself and the society it is supposed to represent. This is what started to happen at the beginning of the twenty-first century: the digitalisation of everyone's life, the substitution of information for the capital as the driving force of industrial innovation, the contraction of the perceived distances due to the availability of direct sources of information, started to become so prominent—and evident to everybody—that a request for school to cope with innovation became inevitable. In (Fadel, Bialik, & Trilling, 2015) this is identified as "the perfect learning storm", caused by four converging forces: knowledge work ("steady supply of well-trained workers, using brainpower and digital tools to apply well-honed knowledge skills to their daily work"); thinking tools (the necessity to use–and not be overwhelmed by–the digital tools for thinking, learning, communicating, collaborating, and working); digital lifestyles (the naturalness of use of digital, mobile, ubiquitous tools requires that also learning become interactive, personalized, collaborative, creative, and innovative); learning research (developments in learning technology allow "to personalize learning to meet each student's learning abilities and disabilities, learning styles and preferences, and unique profile of talents and competence.") Moreover, Pierre Bourdieu's notion of "cultural capital" (Bourdier, 1977) applies easily here: if school wants to maintain its role as a driver of social mobility, it has to change its approach, so that the digital resources that every student nowadays has, could become a capital for everybody, and not only for those children and young people who come from homes where that culture is present and that capital is already exploited (see also (Merchant, 2007)).

It is in this context that Wing's peroration about CT made its triumphant march. The need to respond to the societal changes was matched by an educational offer,

[21] Mentioning just a few of them: Code.org and its *Hour of Code*, the EU Code Week, the ECDL Foundation and its *Computing and Digital Literacy* document, CodeAvengers, CodeMonkey, CodeCombat, etc.

which was broad (and undefined) enough to appeal both at those wanting a formal presence of computer science in the curriculum, and at those who would instead go for mere "digital literacy", or simply "coding." The fact that CT was not operationally (or epistemologically) defined, only helped in its diffusion. It is now well ingrained in the school system, up to be present in several comprehensive reform proposals. Of these, let us only cite (Ananiadou & Claro, 2009), an influent working paper of the OECD, which lead the way to the planned inclusion of CT in the PISA 2021 study[22], as part of mathematics.

6 Teaching computer science

Despite the ambiguities surrounding CT, the momentum it gave to teaching CS to all students is something that cannot be undervalued. Following (Lewis, 2017), we discuss in this section, in the light of our previous arguments, some of the immediate, and long term benefits that are claimed for this "CS for all."

First, an old, and much used, claim is that *Programming can teach students to think logically*. The same has been said for decades for the teaching of Latin, or ancient Greek. It may be true, but not because an *automatic competence transfer* happens (on the contrary, research in education shows that transfer is difficult, especially between distant domains (Ambrose, Bridges, DiPietro, Lovett, & Norman, 2010)), but because "engaging in intellectually demanding tasks is important for student's cognitive development" (Lewis, 2017). After Papert's proposals, during the 80s some research about cognitive effects of programming has been conducted, leading to mixed results (Scherer, 2016). Papert himself acknowledges that what he calls the "Latinesque" justification for teaching something is not sufficient: one should always test if there are other ways to achieve the same goals (Papert, 2006). It should be clear from the previous sections, however, that he thought computers and programming have some peculiar characteristics that make them particularly useful for the training in logical thinking.

Second, a specific claim regards the use of CS techniques in STEM education: *Programming helps student learn Science and Math*. While, again, it is unlikely that learning to program will directly transfer in better learning of other STEM disciplines (Lewis, 2017), studies from the 80s (for a review, see (Guzdial, 2015, pp. 41-49)) show that students can learn better some scientific concepts with the help of *specifically designed programming environments* (LOGO being the originating one). By contrast, some (but not all) of these studies found that students learned

[22] The Programme for International Student Assessment (PISA) is a triennial international worldwide survey by the Organisation for Economic Co-operation and Development (OECD) for the evaluation of educational systems through the measure of the performance on mathematics, science, and reading of students in their 15th year. Mathematics is the major domain assessed in the edition 2021, as it was in 2003 and 2012. At the moment the present paper is written, a preliminary document with reference to CT is available: "PISA 2021 Mathematics Framework", Directorate for Education and Skills, 6 April 2018, document EDU/PISA/GB(2018)4.

only those basic concepts of CS/Programming that were needed to interact with the learning environments[23].

Much more interesting is that *Programming provides emotional value, agency and motivation*. Indeed, in the constructionist spirit, programming can be "a medium for creation, communication and creative expression" (Lewis, 2017). Kafai and Burke (Kafai & Burke, 2014) show examples of students using programming and "making" to create, and then to connect with others while sharing their creations. Of course, programming is not the only medium useful for this; however, the fact that through programming we can simulate (and make concrete) many physical or abstract processes, gives it a particular educational role. Moreover, the availability of mobile devices and the ease of connection and sharing through the Internet, make programming a privileged, central tool for such kind of "samba schools."

Now that we live in an "ubiquitous computing" world, we cannot forget that *Learning CS helps student understand the world, act and grow as digital citizens*. If schools do not provide intellectual instruments to be active digital citizens (e.g. to take informed decisions about privacy and security), to access digital jobs, to be informed towards higher study of computer science or engineering, this will perpetuate discriminations, letting student decide their future only on the basis of stereotypes[24]. Remembering that no transfer is automatic, it is important to explicitly link the concepts from CS to the "computational world" that the students will found outside school.

7 Between Scylla and Charybdis

The proposed inclusion of CT in the PISA 2021 study may be seen as the crowning success of "Wing's computational thinking"—it made it into one of the longest-running and accepted international tests of the performance of students across disciplines. It may be read, however, also as a normalizing move of the establishment towards "Papert's computational thinking": instead of "turning learning upside down", the computational objects are integrated into the standard practice of traditional education, thus neutralizing their revolutionary potential. If CT wants to be a cornerstone of education for the next decades, it must navigate carefully between Scylla and Charybdis.

On one side, it must resist to the illusion that "coding" (as representative of any simplistic approach to the acquisition of the basic of computer science) could be a shortcut to the acquisition of that "algorithmizing" that early computer scientists viewed as one of the main contributions of their discipline to general culture. During

[23] Similar results were obtained by Sharon Carver (see (Guzdial, 2017)) when trying to teach problem solving in real life situations through programming: it worked for problem solving, but the students did not learn much programming.

[24] Recall also the discussion about the "digital capital" of Section 5.

its evolution, computer science developed its own big ideas[25] that should become part of all school curricula, to close the digital discontinuity we are facing today.

On the other hand, CT must preserve the innovation potential of the samba schools of computation, without being marginalized into special educational initiatives. We are now aware of the huge potential that comes from making learning constructive and meaningful for students—it re-empowers powerful, big ideas (of mathematics, physics and potentially any other discipline) that have been disempowered by school. Indeed, solving problems using computers and computer science principles forces to think logically/systematically/procedurally, not because of an external imposition, but in virtue of the computers' intrinsic nature. Moreover, constructing computational artifacts gives the opportunity to have computational objects to identify with, and to "concretely" manipulate them, to explore and build with them. And this has to happen in all "standard" schools, because it is there that it is necessary, where the transformation of digital resources into a cultural capital is an imperative for social inclusion.

Teaching computer science should then focus on the "big ideas", rather than on the technical details. These details are of course needed, but only if instrumental for conveying the deep concepts, and always in the context of "deliberate" learning, where also the repetitive technical training assumes its meaning. At the same time we should not forget that introducing a new formal discipline in school can *disempower* its powerful ideas, transforming them in mere transmission, application, repetition and assessment. CS should be the furthest discipline to undergo such process of disempowerment, because of its intrinsic characteristics that make it a perfect tool to convey a radical change in learning.

Only if CT will be savvy enough to preserve also its affective dimension as a "samba school," it will contribute to the empowering of all students, in particular the young and the disadvantaged, and to the bottom-up reconstruction of its concepts, also those of citizenship and participation.

References

Aho, A. V. (2011). Ubiquity symposium: Computation and computational thinking. *Ubiquity*, *2011*(January).

Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C., & Norman, M. K. (2010). *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons.

Ananiadou, K., & Claro, M. (2009). *21st century skills and competences for new millennium learners in OECD countries* (OECD Education Working Papers No. 41). Paris: OECD Publishing.

[25] For example, from (Bell, Tymann, & Yehudai, 2018): "information is represented in digital form.", "programs express algorithms and data in a form that can be implemented on a computer", "digital systems create virtual representations of natural and artificial phenomena", "digital systems communicate with each other using protocols," and so on.

Armoni, M. (2016). Computing in schools: Computer science, computational thinking, programming, coding: The anomalies of transitivity in K-12 computer science education. *ACM Inroads*, *7*(4), 24–27.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference. In *Proceedings int. conf. on information processing, UNESCO* (p. 125-132).

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., ... Woodger, M. (1960). Report on the algorithmic language ALGOL 60. *Commun. ACM*, *3*(5), 299–314.

Bell, T., Tymann, P., & Yehudai, A. (2018). The big ideas in computer science for k-12 curricula. *Bulletin of EATCS*, *1*(124).

Bender, W. (2017). La plataforma de aprendizaje sugar: Affordances educativas para el pensamiento computacional. *Revista de Educación a Distancia*(54).

Bourdier, P. (1977). *Outline of a theory of practice*. Cambridge: Cambridge University Press.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the american educational research association, vancouver, canada* (pp. 1–25).

CAS, B. (2014). *Computational thinking.* Retrieved from `http://barefootcas.org.uk/wp-content/uploads/2014/10/Computational-thinking-Barefoot-Computing.pdf`

Caspersen, M. E., Gal-Ezer, J., McGettrick, A., & Nardelli, E. (2018). *Informatics for all the strategy* (Tech. Rep.). New York, NY, USA: ACM Europe & Informatics Europe.

Denning, P. J. (2009). The profession of IT: Beyond computational thinking. *Commun. ACM*, *52*(6), 28–30.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Commun. ACM*, *60*(6), 33–39.

Denning, P. J., Tedre, M., & Yongpradit, P. (2017). Misconceptions about computer science. *Commun. ACM*, *60*(3), 31–33.

Dijkstra, E. W. (1974). Programming as a discipline of mathematical nature. *Am. Math. Monthly*, *81*(6), 608—612.

Fadel, C., Bialik, M., & Trilling, B. (2015). *Four-dimensional education*. CreateSpace Independent Publishing Platform.

Feurzeig, W., Papert, S., Bloom, M., Grant, R., & Solomon, C. (1970). Programming-languages as a conceptual framework for teaching mathematics. *SIGCUE Outlook*, *4*(2), 13–17.

Floyd, R. W. (1967). Assigning meanings to programs. *Mathematical aspects of computer science*, *19*, 19-32.

Forsythe, G. E. (1968). What to do till the computer scientist comes. *The American Mathematical Monthly*, *75*(5), 454-462.

Google. (2017). *Exploring computational thinking.* Retrieved from `https://edu.google.com/resources/programs/exploring-computational-thinking/`

Gorn, S. (1963). The computer and information sciences and the community of disciplines. *Behavioral Science*, *12*(6), 433-452.

Guzdial, M. (2015). *Learner-centered design of computing education: Research on computing for everyone*. Morgan & Claypool Publishers.

Guzdial, M. (2017). *Why should we teach programming (Hint: It's not to learn problem-solving)* . Retrieved from `https://computinged.wordpress.com/2017/10/18/why-should-we-teach-programming-hint-its-not-to-learn-problem-solving/`

International Society for Technology in Education (ISTE), & Computer Science Teachers Association (CSTA). (2011). *Operational definition of computational thinking for K–12 education.*

Kafai, Y. B., & Burke, Q. (2014). *Connected code: Why children need to learn programming*. The MIT Press.

Katz, D. L. (1960). Conference report on the use of computers in engineering classroom instruction. *Commun. ACM*, *3*(10), 522–527.

Knuth, D. E. (1972). George Forsythe and the development of computer science. *Commun. ACM*, *15*(8), 721–726.

Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly*, *81*(4), 323–343.

Latour, B. (1986). Visualisation and cognition: Thinking with eyes and hands. In H. Kuklick (Ed.), *Knowledge and society studies in the sociology of culture past and present* (Vol. 6, p. 1-40). Jai Press.

Lewis, C. M. (2017). Good (and bad) reasons to teach all students computer science. In S. B. Fee, A. M. Holland-Minkley, & T. E. Lombardi (Eds.), *New directions for computing education: Embedding computing across disciplines.* New York: Springer.

Lodi, M., Martini, S., & Nardelli, E. (2017). Do we really need computational thinking? *Mondo Digitale*(72), 1-15.

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational thinking in K-9 education. In *Proc. of the working group reports of the conference on innovation & technology in computer science education (ITiCSE-WGR '14)* (pp. 1–29). New York, NY, USA: ACM.

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: What's the connection? *Commun. ACM*, *29*(7), 605–610.

McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine, part I. *Commun. ACM*, *3*(4), 184–195.

McCarthy, J. (1961). A basis for a mathematical theory of computation, preliminary report. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference* (pp. 225–238). New York, NY, USA: ACM.

Merchant, G. (2007). Mind the gap(s): Discourses and discontinuity in digital literacies. *E-Learning and Digital Media*, *4*(3), 241-255.

Minsky, M. (1970). Form and content in computer science (1970 ACM Turing lecture). *J. ACM*, *17*(2), 197–215.

Naur, P. (1966). Proof of algorithms by general snapshots. *BIT Numerical Mathematics*, *6*(4), 310–316.

Newell, A., Perlis, A. J., & Simon, H. A. (1967). Computer science. *Science*, *157*(3795), 1373–1374.

Nofre, D., Priestley, M., & Alberts, G. (2014). When technology became language: The origins of the linguistic conception of computer programming, 1950–1960. *Technology and Culture*, *55*, 40-75.

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

Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal*, *39*(3&4), 720–729.

Papert, S. (2006, 12 4). *Keynote lecture.* Retrieved from `http://dailypapert.com/wp-content/uploads/2012/05/Seymour-Vietnam-Talk-2006.pdf` (Keynote at ICMI 17 Conference in Hanoi, Viet Nam)

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*(2), 137-168.

Piaget, J. (1973). *To understand is to invent: The future of education*. Penguin Books.

Salomon, G. (1984). On ability development and far transfer: A response to Pea and Kurland. *New Ideas in Psychology*, *2*(2), 169-174.

Scherer, R. (2016). Learning from the Past. The Need for Empirical Evidence on the Transfer Effects of Computer Programming Skills. *Frontiers in Psychology*, *7*, 1390. doi: 10.3389/fpsyg.2016.01390

Tedre, M. (2014). *The science of computing: Shaping a discipline*. Chapman and Hall/CRC.

Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. In *Proceedings of the 16th Koli Calling international conference on computing education research* (pp. 120–129). New York, NY, USA: ACM.

Wing, J. M. (2006). Computational thinking. *Commun. ACM*, *49*(3), 33–35.

Winsberg, E. B. (2010). *Science in the age of computer simulation*. Chicago: University of Chicago Press.