Science & Education (2021) 30:883–908
https://doi.org/10.1007/s11191-021-00202-5

ARTICLE

Computational Thinking, Between Papert and Wing

Michael Lodi[1,2] • Simone Martini[1,2]

Accepted: 10 February 2021/ Published online: 28 April 2021 Published online: 28 April 2021

Abstract
The pervasiveness of Computer Science (CS) in today's digital society and the extensive use of computational methods in other sciences call for its introduction in the school curriculum. Hence, Computer Science Education is becoming more and more relevant. In CS K-12 education, computational thinking (CT) is one of the abused buzzwords: different stakeholders (media, educators, politicians) give it different meanings, some more oriented to CS, others more linked to its interdisciplinary value. The expression was introduced by two leading researchers, Jeannette Wing (in 2006) and Seymour Papert (much early, in 1980), each of them stressing different aspects of a common theme. This paper will use a historical approach to review, discuss, and put in context these first two educational and epistemological approaches to CT. We will relate them to today's context and evaluate what aspects are still relevant for CS K-12 education. Of the two, particular interest is devoted to "Papert's CT," which is the lesser-known and the lesser-studied. We will conclude that "Wing's CT" and "Papert's CT," when correctly understood, are both relevant to today's computer science education. From Wing, we should retain computer science's centrality, CT being the (scientific and cultural) substratum of the technical competencies. Under this interpretation, CT is a lens and a set of categories for understanding the algorithmic fabric of today's world. From Papert, we should retain the constructionist idea that only a social and affective involvement of students into the technical content will make programming an interdisciplinary tool for learning (also) other disciplines. We will also discuss the often quoted (and often unverified) claim that CT automatically "transfers" to other broad 21st century skills. Our analysis will be relevant for educators and scholars to recognize and avoid misconceptions and build on the two core roots of CT.

For Martini: Research partially conducted while on sabbatical leave at the Collegium – Lyon Institute for Advanced Studies

✉ Simone Martini
simone.martini@unibo.it

[1] Dipartimento di Informatica-Scienza e Ingegneria, Alma Mater Studiorum – Università di Bologna, Mura Anteo Zamboni, 7, 40126 Bologna, BO, Italy

[2] Inria Sophia Antipolis-Bologna, Valbonne, France

1 Introduction

Today's society is permeated by digital technology and culture. Yet, almost in all countries, the reform of school curricula to include better Computer Science (CS) Education is far from complete (CECE, 2017; Code.org et al., 2020). Most other sciences use digital technology to present their concepts and elaborate information (simulation of physical phenomena, DNA sequencing and analysis, or manipulation of abstract geometrical objects, and so on).

Computer Science[1] is an independent and recognized scientific discipline (Denning 2013). While we believe that CS is the discipline to be taught at school (and we will elaborate on this throughout the paper), the buzzwords of the moment for CS K-12 education are mainly "coding"[2] and "computational thinking" (CT). In this paper we will focus on CT, reviewing, with a historical approach, two different contexts in which the expression emerged.

The modern (and long) wave of the expression "computational thinking" started with a seminal essay by Wing (2006)[3], 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); ISTE and CSTA (2011b)), which have proposed educational material, definitions, and assessment methods (Grover & Pea 2013). Historically, however, the expression CT was used for the first time by Seymour Papert[4] in 1980, with a different *nuance* of meaning, which should not be forgotten. For Papert, CT is the result of his *constructionist* approach to education, where social and affective dimensions are as important as the technical content.

Both Wing's and Papert's CT come (or seem to come) with the idea that the competencies acquired as CT will easily (or even automatically) *transfer* to other disciplines (see Section 5.2 for Wing and Sections 3 and 6 for Papert, respectively). This largely unverified (when not disputed by the relevant educational literature) claim has also been a reason for their appeal, especially among the nonspecialists.

The thesis that we will argue in this paper is that the two different dimensions proposed by Wing and Papert should both be maintained when introducing CT in K-12 education. We will present a historical reconstruction of their two contributions, discussing how both have been misunderstood, and rectifying their original meaning. From Wing, we should retain the centrality of computer *science*—CT being *the (scientific and cultural) substratum of the technical competencies.* In this way, CT becomes a lens and a set of categories for understanding the algorithmic fabric of today's world. From Papert, we should retain the constructionist idea that only a social and affective involvement of the student in constructing a (computational) artifact will make programming an interdisciplinary tool for learning (also) other disciplines.

The paper will focus on CT in K-12 education. We will make the point that CT is not a discipline, *per se*. CT must be understood in its proper context, which is that of CS. As it is the

[1] See Section 1.1 for a terminological focus on different ways in which the discipline is called.
[2] See Section 1.1 for a brief discussion on the possible different meanings of the term.
[3] Jeannette Wing is a researcher in the field of formal languages. She is a Professor of CS at Columbia University. During her career, she served as Head of Department at Carnegie Mellon University, as Corporate Vice President of Microsoft Research, and as assistant director for Computer and Information Science and Engineering at the NSF.
[4] Seymour Papert was a mathematician, computer scientist, and educator. He has been Professor of Math and of Education at MIT. He worked on pioneering studies on artificial intelligence, theorized the constructionist learning theory, and co-invented the LOGO educational programming language.

case for all sciences, CT is *also* transversal to other disciplines, but cannot be reduced to this role. There is mathematics for and inside physics, like there is physics inside and for chemistry. But this does not mean that mathematics, or physics, are only instrumental to those other disciplines. They could be transversal because they have a proper epistemological identity. Computer Science, and CT inside it, is no different.

There is a CT transversal to computational sciences: Papert's CT stresses the importance of the computer as a powerful meta-tool for "making the abstract concrete." But Wing's CT reminds us that without rooting these tools into CS, they would be just scattered techniques lacking method and unity. There is a CT as a result of the teaching of Computer Science: this is the core of Wing's CT. But Papert's CT reminds us that the mere disciplinary stance would limit the potentials of this protean tool that the computer could be.

The analysis of "Papert's CT" will be the subject of the central Section 3 of the paper. Before turning to this, however, we summarize 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 work of Tedre and Denning (2016)[5] for a detached critical review, which also frames CT in its historical context. A specific contribution of the present paper is a detailed analysis of the (supposed) Papert's claim that the competencies acquired as CT will transfer to other disciplines. We will show that Papert had a far more complex position, and that he never claimed that transfer would happen *per se* (see Section 3 and, especially, Section 6.) Section 4 will discuss the discontinuity in the school curricula produced by the impact of digital technologies in everyday lives, and hence why Wing's proposal made such a momentum. Sections 5 and 6 will summarize the main contributions of Wing's and Papert's CT, respectively—those that, we argue, should *both* be retained if CT has to be that formidable actor of change that its proposers envisaged for it.

Given the heterogeneous landscape around CT, our review and analysis can help other educational researchers better focus on the core ideas behind the expression (and the different nuances in which scholars use it) when designing, researching, and evaluating new methods or competence acquisition. Moreover, it can be useful for educators to better judge what to introduce in their classes, why, and through which materials.

As we will argue throughout the paper, the major way to foster CT is to teach Computer Science fundamental ideas, concepts, and practices: given the historical and philosophical nature of this paper, we refer to the book edited by Grover (2020) for a comprehensive collection of examples of activities to teach CS in K-12 education, rooted in research findings. For scholars, we refer to the handbook edited by Fincher and Robins (2019) for a comprehensive picture of state of the art in Computer Science Education Research.

1.1 Terminology and Background

Since much literature has been produced around CT, from very different contexts (CS Education, Science Education, Pedagogy), let us define and discuss important concepts we will use in the following.

[5] Peter Denning has been writing several critical papers on CT and its hype, see, e.g., Denning (2009, 2017); Denning et al. (2017).

1.1.1 CS, coding

In Europe, CS is often referred to as Informatics[6] or Computing (especially in the UK). While many definitions have been proposed, in this paper we will stick to the following.

Definition of Denning et al. (1989) *The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, "What can be (efficiently) automated?"*

Note that in the above definition, "computing" was used as a shorthand for "computer science and engineering,"[7] while today "computing" tends to be used as an umbrella term for a family of disciplines (Shackelford et al., 2006) (Computer Science, Computer Engineering, Information Systems, Information Technology, Software Engineering) or fields that deal with computation (such as computer science, computational science, information science, computer engineering, and software engineering.) (Denning 2013)

For computer scientists and computing professionals, the word "coding" can have different meanings. It can refer to the representation of information (letters, pixels, sounds) with a (numeric) "code," for example, UNICODE for characters (in this case we should talk more formally about *encoding*). Or, it can refer to the encryption process of a plaintext in a ciphertext for privacy or authentication reasons. In the context of software development, it is recognized as one phase (the one where you "physically" write the program) of the programming process, which also includes design, algorithm development, test, debug, documentation, and maintenance. Today, coding and programming are used as synonyms in the tech jargon (Armoni 2016), but we observe that "coding" is nowadays often used to denote a "more playful and non-intimidating description of programming for beginners" (Prottsman 2015). We believe that coding, or programming, is (only) an important tool to learn the fundamental aspects of CS, as we will discuss in Section 5.

1.1.2 Transfer and CS

In the context of this paper, we assume the following.

Definition of Ambrose et al. (2010) *The application of skills (or knowledge, strategies, approaches, or habits) learned in one context to a novel context.*

Transfer can be guided or spontaneous. Although the latter is preferable, it is also more difficult for it to occur. Moreover, it is easier for transfer to happen if it is near (i.e., in similar contexts) rather than far (between very different disciplines or contexts) (Robins et al., 2019).

In the context of CS education, mainly near transfer has been studied. Negative transfer (i.e., where previous knowledge is a disadvantage) has been found: learning a second language

[6] Note that this term tends to assume a different meaning in America, more focused on societal aspects of computers. See for example https://faculty.washington.edu/ajko/advice#csis. Accessed10 February 2021.

[7] "We immediately extended our task to encompass both computer science and computer engineering, because we concluded that no fundamental difference exists between the two fields in the core material." (Denning et al., 1989, p. 10)

is sometimes harder than the first, and it is hard to translate natural language communication into programming (Guzdial 2015). Positive transfer has been found in the "computational thinking patterns" from games to scientific simulations in a rule-based environment (Basawapatna et al., 2011; Ioannidou et al., 2011), between visual and textual interfaces (Hundhausen et al., 2009) and between block-based and text-based languages (Weintrop and Wilensky 2019). Transfer from unplugged to plugged activities is still debated (Bell & Lodi 2019).

Studies on far transfer between computer science and other disciplines have a long-dating story, starting from the debate on LOGO between Pea and Papert (see Sections 2 and 6). Successful examples of *using* CS (and in particular programming) mainly as a tool to foster the learning of other subjects are as follows. The "Bootstrap:Algebra" module succeeded (Schanzer et al., 2015) in leveraging middle and high school students' performances on algebra problems, through teaching them algebra concepts by programming video games with the functional programming language Racket. In the context of Physics, diSessa (2000; 2018) used his programming language Boxer to successfully teach sixth-grade students the physics of motion (e.g., falling objects, vector calculus) by making them program and experiment with simulations. As noted by Guzdial (2015), however, the amount of computer science learned by students in these and other examples is confined to what they strictly need to use in the main subject they are learning.

Transfer between studying programming and far domains like the so-called higher-order thinking skills (like problem solving, critical thinking, creative thinking, and decision making) or even further psychological constructs (like resilience, grit, and growth mindset) seems much harder—as predicted by general education research (for example, Gick and Holyoak (1980) found that problem decomposition, one of the most highlighted aspects of CT, is not easily transferable between different domains). The studies in this direction are scattered and inconclusive. A recent meta-review (Scherer et al., 2019) found evidence of strong near transfer, and of moderate-to-low far transfer: it is "more likely to occur in situations that require cognitive skills close to programming," but recognizes flaws in the current body of research and advocates for more studies. As we will extensively discuss throughout the paper, both Papert's and Wing's CT may have been attractive because of some transfer promises, but their importance must be searched for in some of their deeper characteristics.

1.1.3 Computational Sciences, and CT for Math and Science

Computational thinking and computer science—although clearly related—must not be confused with *computational sciences.* As explained by Denning and Tedre (2019, ch. 7), the use of CS methods in other sciences brought to the birth of *computational sciences*: "the branches of every scientific field that specializes in using computation, such as computational physics, bioinformatics, and digital humanities." In particular, computing helped other sciences in simulation, information interpretation of nature, and numerical methods. These fields are separated from CS, because computer scientists would lack the disciplinary knowledge to work in them. Moreover, the use of computation is done for different reasons: scientists exploit computational tools, "using modeling and simulation to explore phenomena, test hypotheses, and make predictions"; computer scientists study those very tools.

Many math and science educators are getting in touch with CT since it has been introduced in the US "New Generation Science Standards" (NGSS 2013), where "mathematical and computational thinking" is defined as a "practice of science and engineering." In that context, mathematical thinking and computational thinking are seen as useful tools for exploring scientific or engineering problems. There is a long tradition of using CS tools for better learning math and science with languages like LOGO, Emile, Boxer, NetLogo, StarLogo, and Racket (for a review, see Guzdial (2015, pp. 48-49)). In that context, the focus is (rightfully) to study methods for fostering science learning, and programming is mainly seen as a tool.

In this paper, we will follow a radically different focus, looking at the problem from a computer science point of view: as said, computer science is a scientific discipline worth being taught for its own sake. Our exploration will bring us back also to its transversal potential.

Before digging deeper into the analysis of the disciplinary and transversal potential of "thinking like a computer scientist," let's go back to the birth of the discipline, where the *idea* of CT started to emerge as well.

2 Prehistory

The end of the 1950s and the early 1960s are the years in which the field of computing gradually builds its self-understanding as an autonomous discipline (Tedre 2014): computer science. 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 et al., 2014). In the early days, programming was mainly a technological affair (strictly coupled to the technology of 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 an 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 (Floyd 1967; Naur 1966). 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 feels like the opportunity for computing to evolve from its "obscure" past (made of mathematics, cybernetics, logic, physics, engineering, linguistics) and consciously present 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.[8]

It is in this context that an *idea* that, we will argue, will become *computational thinking*, starts to emerge. For example, no 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

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

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 competencies of that new field, which at that same time struggled to be recognized as an autonomous scientific discipline.[9] 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 1970s 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"[10]) on mathematics education. The thesis is that the computer scientist has the role, the responsibility, and the competencies 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."

Donald Knuth (one of the stars at Stanford, recipient of the Turing award in 1974, at the age of 36), 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).

As anticipated, we believe that what these quoted authors evoke is a concept, an *idea* of what is today called computational thinking. We propose to characterize this concept as *the natural sediment of disciplinary learning of computer science*—that which remains behind when all the technicalities and the definitions of the discipline are long forgotten.[11] Moreover, it is easy to spot, already in these early proposals, the (largely questioned and unproven—see, for example, Guzdial (2015) and Lewis (2017)) claim that the cognitive skills gained through

[9] A struggle that was going to be long. The first *Computer Science* department of the USA 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) felt the need of a letter to Science (Newell et al., 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).

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

[11] We refrain from a historical reconstruction of this folklore expression, often said of "culture."

programming (or, more generally, through computer science techniques) easily *transfer*[12] to other disciplines—from the already quoted Minsky (1970) and the related Feurzeig et al. (1970)[13] for mathematics, to the far-reaching Mayer et al. (1986) and Pea and Kurland (1984), for which see also the commentary criticism from 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 that did not change much when this *idea* of computational 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*.[14] 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," and so on) are one of the most effective and economical 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.

Before zooming on Papert's view on CT, let us first introduce the issue of the possible transfer of skills. 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 even explicitly confirmed by Minsky (1970), where he 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."

On this count, however, it is useful to read Feurzeig et al. (1970),[15] 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, showing in fact that the initial, naive idea of "automatic transfer from learning programming to learning math" is specified in a more precise and realistic idea of "using programming as a tool for experimenting and learning with math."

Their first claim seems indeed to support the idea of CT transferring to general skills, as already noted by Tedre and 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

12 For a discussion about "transfer of skills", see also Section 6.

13 We will discuss Feurzeig et al. (1970) in detail in the next section.

14 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).

15 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.

problem solving with an experimental approach.[16] In fact, children have to impose on themselves rigor and precision in instructing the computer—being explicit and precise is not imposed (incomprehensibly) by enforcement of the teacher,[17] 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 a 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"). Note again the emphasis on the fact that programming provides high motivations for learning these concepts. Moreover, also note that many of these ideas are included in modern CT definitions, often as CT "practices" or "approaches" (see Section 5.1). Bender (2017) 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 recognizing the importance of this second aim).

In summary, it is not the programming skills that count, or the "algorithmizing" concepts acquired through programming. A careful reading of Feurzeig et al. (1970) and of *Mindstorms,* which takes into account the entire texts—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" (Papert 1980). Any 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. Papert (2000) 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.

To clarify this modality of interaction, 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

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

[17] For example, the need to learn and use a new programming construct needed for a computer (or, say, a robot) to perform a specific task can be accepted much easily if it is felt as a *necessity* for "making it work" rather than "a thing the teacher wants me to learn".

> 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." (Papert 1980, p. 182) [...] They will be manifestations of a social movement of people interested in personal computation, interested in their own children, and interested in education.

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*[18] 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). Papert (1980, p. vi), 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. Piaget distinguishes between "concrete" and "formal" thinking, the first already present at the age of first grade and consolidated afterward, 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 into 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, 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 (Papert 1980, p. viii).

Every student will be obsessed by—will be in love with—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 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. While the initial focus was on Math, Papert's project was much more ambitious: creating *microworlds* (LOGO Turtle being the first one) speaking different languages for learning the most diverse concepts.

Mindstorms is particularly critical of traditional school systems. According to Papert, computers and programming will make old schools obsolete and useless, because learning will happen in constructionist 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

[18] 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.

learns, the expert teaches, but also practices for harder moves. There is a great social cohesion, a great sense of belonging, a firm idea of having a "common purpose." Although learning is spontaneous and natural, it is also *deliberate*—the 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, from 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 they are immersed in an environment whose activities are both rich with computational principles and meaningful for the community. The social value of learning CT is discussed in Section 6.

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 the 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[19], and bring them powerful ideas about mathematics, physics, and probability (which are the fields touched upon in Mindstorms).

Finally, Papert (2000) is optimistic about the fact 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 4 and discuss popular misunderstandings in current education in Section 5.

Papert used the expression again, in some of his important writings (Papert 1993, 1996b, 2006; Papert & Harel 1991), but again only *en passant*. We will review some of these uses: while they appear in very different contexts, they reinforce Papert's constructionist ideas of personalized, affective, social learning through computers.

The expression "computational thinking and practice" is used in Papert and Harel (1991) as opposed to "computer literacy" and "computer-aided instruction (CAI)" in relationship with the feminist battle. Despite the innovation, CAI seems to support "the abstract and impersonal detached kinds of knowing" of traditional schools. By contrast, "many women prefer working with more personal, less-detached knowledge and do so very successfully," and it is argued that Papert's CT moves towards this direction.

The expression is used again by Papert (1993, p. 184), talking about the origin of computers for the need of calculating missile trajectories: "many top mathematicians were mobilized to the task, among them John von Neumann and Norbert Wiener, who

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

became [...] leading pioneers in the emergence of computers and of computational thinking." However, what is more interesting is that, in the same chapter, Papert envisages a new subject that he proposes to call "Cybernetics for children," and which shares many characteristics with what we are calling here "Papert's CT." In fact, Papert (1993, p. 181-182) proposes a subject that:

- Is the "kernel of knowledge needed for a child to invent" and to build entities that resemble or evoke real-life technologies;
- Uses that kernel as a starting point for connections with other areas;
- Uses "technology as a medium for representing behaviors that one can observe in oneself and other people";
- Fosters a more intimate and affective relationship between the student and his work;
- Has a more pluralistic underlying epistemology.

Papert (1996b) returns again to the expression when comparing two solutions to a geometrical problem: one used a powerful geometry tool to solve the problem as it would have been solved in a traditional pen and paper setting using Euclidean geometry, the other one used a Monte Carlo simulation. Both of these methods, Papert says, use a computational tool to find a solution effectively, but none of them makes the underlying mathematics clearer. So, "[t]he goal is to use computational thinking to forge ideas that are at least as 'explicative' as the Euclid-like constructions (and hopefully more so) but more accessible and more powerful." Papert then proposes a method to use Turtle geometry to understand the mathematical concepts underlying the solution deeply.

Finally, "computational thinking" will return many times in Papert's last[20] talk (Papert 2006), a few months after the publication of Wing's paper. The starting point is that the school system is dominated by graphocentrism, because it uses obsolete technologies—pencil and paper.[21] This reduces knowledge "to the kind of knowledge that can be written down: propositions"—a "propositional thinking" which is suitable for testing and grading students. LOGO was the first step beyond this paradigm, introducing "procedural thinking": knowledge as instructions, expressed through a programming language. Nevertheless, Papert acknowledges that this is only a first step towards *computational thinking*. One of the main aspects 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[22] may, of course, be used to teach programming or standard geometry, but their primary intended role is that 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* an affective sense. The significant 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 big mathematical 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.

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

[21] 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.

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

4 The Digital Discontinuity

Despite rapid society evolution, school curricula, in any country, have significant inertia—it changes very slowly, and radical reforms are rare (Jónasson 2016; Tyack & Cuban 1995). They happen only when a fracture between the curriculum itself and the society it is supposed to represent occurs (Reimers & Chung 2016). This is what started to happen at the beginning of the twenty-first century, provoking what we may call the digital discontinuity: the digitalization 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 the school to cope with innovation became inevitable (see, for one of the earliest proposals, the Fluency with Information Technology (FIT) framework; Committee on Information Technology Literacy 1999).

The US "National science education standards" (NRC 1996) stress the role of computers in simulation (of complex phenomena of other sciences) and their use "for the collection, analysis, and display of data," but still fail to see computing as an autonomous, and yet pervasive, science. Fadel et al. (2015) see the situation of early 2000 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.")

It is in this context that Wing's peroration about CT (Wing 2006) made its triumphant march, twenty-six years after *Mindstorms*, and after the dramatic rise of the digital society and computational sciences. 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 scientists—most scientific disciplines become "computational," stably adding *simulation* as the 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 broader 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."[23]

The need to respond to the societal changes was matched by an educational offer, which was broad (and undefined) enough to appeal both to those wanting a formal presence of computer science in the curriculum, and to those who would instead go for mere "digital

[23] This widely quoted definition comes from an unpublished work by Cuny, Snyder and Wing, and referred in Wing (2011). Moreover, Wing says it was originated by a discussion with Alfred Aho, who provided a very similar definition, more focused on "algorithmic thinking": "We consider computational thinking to be the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms" (Aho 2011). It is worth noticing that Aho stresses, in particular, the role played in this definition by the information processing agent, and that computational thinking should be based on clearly defined models of computation.

literacy," or simply "coding" (in the US "New Generation Science Standards" (NGSS 2013), for instance, "computational thinking" is applied both to the "use of digital tools" and to the creation of "sequences of steps called algorithms.") The fact that CT was not operationally (or epistemologically) clearly defined may have even helped in its diffusion. As Tedre and Denning (2016) say, "Wing's formulation struck a resonant chord," and around that manifesto several interests coalesced into a movement to bring CT into all levels of K-12 education. It is now well ingrained in the school system, up to be present in several comprehensive reform proposals (Berry 2013; K–12 Computer Science Framework 2016; or Ananiadou & Claro 2009), which led the way to the inclusion of CT in the PISA 2021 study[24], as part of mathematics.

5 The Importance of Wing's CT

5.1 Elements of CT

After Wing's seminal paper, many researchers tried to define CT and identify its common elements. Lodi (2020) analyzed the definitions and frameworks proposed in the last decade by a number of educators, in particular, by analyzing definitions from Wing (2006, 2008, 2011), Aho (2011), ISTE and CSTA (2011a, b), Google (n.d.), Brennan and Resnick (2012), Csizmadia et al. (2015), Grover and Pea (2013), Selby and Woollard (2013), Weintrop et al. (2016), Kalelioğlu et al. (2016), Krauss and Prottsman (2016), Shute et al. (2017), College Board (2017), Juškevičienė and Dagienė (2018), and Denning and Tedre (2019).

Lodi found out that most of the definitions suggested CT is a way of thinking, or thought process, for a particular type of problem solving: the one that involves some external agent to automate for carrying out the task. Moreover, he proposed a broad classification of the elements of CT found in many of the abovementioned frameworks.

- Mental processes: mental strategies useful to solve problems (Algorithmic thinking; Logical thinking; Problem Decomposition and Modularization; Abstraction; Pattern recognition; Generalization).
- Methods: operational approaches widely used by computer scientists (Automation; Data Collection, Analysis and Representation; Parallelization; Modeling and Simulation; Analysis and Evaluation; Programming).
- Practices: typical practices used in the implementation of computing machinery based solutions (Experimenting, iterating, tinkering; Test and debug; Reuse and Remix).
- Transversal skills: general ways of seeing and operating in the world fostered by thinking like computer scientists; useful life skills that can enhance thinking like a computer scientist (Design and Create; Communicate and collaborate; Reflect, learn, meta-reflect, understand the world computationally; Be tolerant for ambiguity; Be persistent when dealing with complex problems).

[24] 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 primary domain assessed in the edition 2021, as it was in 2003 and 2012.

We believe these elements, while of course shared with other disciplines or very general, must be understood inside the discipline of CS; otherwise, as stated by Voogt et al. (2015), this "runs the risk of diluting the idea of CT, blurring and making it indistinct from other 21st century skills". For example, Corradini et al. (2017a) found that teachers saw the value of a nation-wide "coding" project more in fostering transversal competencies or domain-general skills (like promotion of awareness and comprehension of problem solving, logical thinking, creativity, attention, planning ability, motivation for learning, student interest, cooperation) than in teaching CS core concepts. The same sample (Corradini et al., 2017b) struggled to give a sound and complete definition of CT, focusing on some crucial aspects like problem solving, mental processes, and logic, but often forgetting to refer in any way to an information-processing agent.

As recognized by Lodi (2020), in Math it is usual to talk about *mathematical thinking*, or *mathematical reasoning*, or *mathematical problem solving*. Moreover, it is clear to researchers that "some aspects of mathematical problem solving are largely discipline-specific (e.g., the knowledge base), some heavily discipline-oriented (e.g., strategies and beliefs), some much like discipline domain-general (e.g., metacognition)" (Li et al., 2019, p. 8). Even if many authors recognized some of CT characteristics in other disciplines, like math and physics, we must pay attention to not lose their specific CS instantiation. Grover and Pea (2013) argue that computing features extend and differentiate these elements from other domains.

This does not mean "thinking like a computer scientist" is the Swiss Army of scientific or even human thinking, but that—to talk about computational thinking—the specific CS context in which to understand CT elements must be maintained. For example, as diSessa (2018, p. 21) points out, the concept of abstraction (one of the core elements of Wing-like definitions) seems to be very different in Math (inferential abstraction), Physics (empirical abstractions), and CS (practical abstraction). Another example is the idea of "computational model" (Denning & Tedre 2019, pp. 160-161): for (computational) scientists, they are sets of (differential) equations used to describe a physical process, to compute numerical data about the process, to simulate the process; for computer scientists are (abstract) machines that run programs written in a particular (programming) language.

5.2 CT (and CS) is Not "Coding"

Wing's CT lays in the path well-marked by the early computer scientists (see Section 2), extracting from computer science a list of thinking patterns ("mental tools that reflect the breadth of the field of computer science"). In particular, Wing (2006, p. 35) is clear in stating that "computer science is not computer programming. Thinking like a computer scientist means more than being able to program a computer."[25]

To computer scientists, it appears clear the inseparable bond between CS and CT, expressed by Jeannette Wing (2006) in her seminal article. However, to the general public, not explicitly trained in CS, these aspects may appear obscure or not so related to CS itself. That is why many educators, for example, tend to stick with the most understandable examples (but for which there is no scientific evidence (Guzdial 2015)): those stating that CT can transfer to

[25] While programming is the most relevant expression tool of computer scientists, the focus of CS is on the automatic elaboration of information (just to make examples: how to encode, transmit, store, search, sort different kinds of data? What elaborations are unfeasible or theoretically impossible?).

everyday life situations (for example, using CS ideas of pre-fetching and caching when preparing a backpack, or the idea of pipeline when organizing a buffet).

Indeed, we see many well-published initiatives where CT is identified (more or less explicitly) with “coding” (that is, just one phase of the programming process), 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. We 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. The problems stem especially when, instead of understanding CT as a cover for the scientific core of computer science, it is viewed instead as a new discipline,[26] which can be taught and evaluated *as such* (Armoni 2016).

5.3 CT Is Not a (New) Discipline, CS Is

As discussed by Nardelli (2019), important school reforms (like the English national computing curriculum,[27] the USA *Every Student Succeeds Act*” (ESSA),[28] and “Computer Science for All”[29] and the French curriculum[30]) seem to acknowledge that CS is the subject to be taught. In all of them, CT is not considered to be a new subject, but as *the conceptual sediment of disciplinary learning of CS.*

Having anchored CT *inside* CS (and not *apart from* CS), however, we must not let some of the criticisms of computational thinking (e.g., the fact that some of its characteristics are shared with other disciplines) diminish the value of CS as an autonomous discipline. As we already recalled in Section 4 with respect to the Cuny-Snyder-Wing’s definition of CT, one of the novelties of CS is the central, constituent role of *effective processes* and the possibility to actually perform (execute) those processes on a computing agent. This brings a radical change with respect to, say, mathematics: from “solving problems” to “effectively solving problems and make something do the computation for us.” Indeed, unlike other disciplines, *CS abstractions can be executed mechanically* (Wing 2008). This means that CS abstractions can be “animated” without having to build a new physical representation of the abstraction itself. In fact, central to CT, as contrasted, e.g., to mathematical thinking, is the creation of executable models. These models are discrete and effective (in the technical sense of computability theory), and they scale at different abstraction levels. These three characteristics make them radically different from other models used in science, which are usually continuous, are

[26] Just to give an example, in Italy “computational thinking” has been inserted as an addendum to the “National Indications for K-8 education”, and “coding/programming” is set to become a school discipline from 2022, while CS is barely mentioned.
[27] https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study. Accessed 10 February 2021.
[28] https://www.ed.gov/essa. Accessed 10 February 2021.
[29] https://www.csforall.org/. Accessed 10 February 2021.
[30] https://www.education.gouv.fr/pid285/bulletin_officiel.html?pid_bo=33400. Accessed 10 February 2021.

effective only in some (special) cases, and are set at a specific abstraction level, with no way to move to a different level, either in the same language or by a uniform (compositional) translation into other languages.[31]

Moreover, CS, through its "languages," provides linguistic support for abstraction: appropriate linguistic constructs allow us to define abstractions in a systematic way, and to move uniformly between their various levels arriving "down" to the physical execution of the identified abstractions. In this, computer science is distinguished from other disciplines, which must build each time "ad hoc" material objects that express the phenomena modeled. This ability to build "executable abstractions" makes CS an important aid for the study of other disciplines as well, because it can make concrete (and so tangible, usable, "tinkerable") the abstract concepts of those disciplines.

5.4 Why Teaching CS

Some detractors of CT (see, for instance, Mannila et al. (2014) for a balanced review) accuse computer scientists of being arrogant in wanting to "teach everyone how to think" or to want to transform every child into a software developer. However, this is a blatant caricature of what Wing's CT affirms. The aim is not to teach software development, but to teach CS in order to give tools to understand and act in our digital world.

In the same vein, already in Wings' original paper, and then in subsequent works, computational thinking has been referred to as the "fourth basics" to be taught in addition to the classic "reading, writing, and arithmetic."[32] This idea also emerges in older works, as in the words of Forsythe mentioned in Section 2. Once again, this has to be understood avoiding its most immediate, simplistic reading. The goal of the first cycle of education is not simply acquiring the "three (or four) R"—it is, instead, the promotion of basic cultural and social literacy, in order to acquire the languages and codes of our culture(s). This literacy *includes* instrumental literacy ("reading, writing, and arithmetic"), and enhances it through the languages and knowledge of the various specific disciplines. This, therefore, frames the "three basic skills"—whose learning was the goal of literacy over the centuries—as *tools* for a higher task: learning to understand the social context in which the student finds himself living: tools that, in today's complex world, need specific languages and knowledge from different disciplines in order to provide adequate basic training. The digital discontinuity we discussed in Section 4 makes CS one of these disciplines (its CT core being especially in focus here). Of course, since transfer is not automatic, to make CT relevant for social and cultural literacy, it is crucial to explicitly link the CS concepts that will be taught to the "computational world" that the students find outside school.

6 The Importance of Papert's CT

Papert's ideas are often framed like "programming will teach students to think [...] or be logical thinkers" (Lewis 2017), similarly to what has been said for decades for the teaching of Math, Latin, or ancient Greek (see, e.g., National Research Council (2000, p. 51)). As seen, this may be true, but not because an *automatic competence transfer* happens (research shows it is very difficult), but

[31] As already introduced at the end of Section 5.1, a canonical example here is mathematical physics and the differential equations it uses as a pervasive model: only in particular cases such a model is effective in the technical sense, and sometimes simple problems are not even analytically solvable (e.g., the three-body problem).
[32] The "fourth R" together with "Reading, wRiting, aRithmetic".

because "engaging in intellectually demanding tasks is important for student's cognitive development" (Lewis 2017). As already described in Section 3, during the 1980s, after Papert's proposals, some research about the 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, but this happens only if they are used as constructionist tools: to create meaningful artifacts. Transfer is something that does not happen automatically, and needs an active and deliberate effort (Papert 1996a):

> [...] the problem of *transfer*: If you learn something in one context, can you use it in another? [...] The answer is simple: depends on what you actually learned. Of course the skill won't transfer if what you learned was a meaningless ritual [...]. But if you understood the principles, and if you have an attitude of self-confidence in trying and modifying, your old experience will let you find out quickly what to do in the new situation. *Transfer is not something that happens to you. It's something you do.*
> (Papert 1996a, p. 125-126, emphasis as in original)

In this framing, we also have to be cautious regarding the claim "programming helps students learn Science and Math" (Lewis 2017). As already discussed in Section 1.1, it is unlikely that learning to program will directly transfer to better learning of other STEM disciplines (Lewis 2017). What is likely is that students can learn better some scientific concepts with the help of *specifically designed programming environments* (LOGO being the originating one), where the amount of CS and programming concepts is limited to what strictly needed for the specific disciplinary learning (Guzdial 2015, pp. 48-49).

Much more interesting, and in line with Papert's CT, is the idea 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 (2014) show examples of students using programming and "making" to create, and then to connect with others while sharing their creations.[33] 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 kinds of social connections and sharing of computational artifacts.

6.1 Other "Avatars" of (Papert's) CT

As briefly mentioned in Section 1, from the appearance of Wing (2006), CT has been the subject of significant investigation, aimed to classify its definitions, discuss the different emerging epistemologies, and give policymakers a guide for its effective inclusion in school curricula. Let us cite Lodi (2020) for a synthetic review and classification of possible definitions; see also Kafai et al. (2020), which we will discuss later.

However, both before and after Wing's paper, as also recognized by Grover and Pea (2013), other "avatars" of CT appeared, under different names. Among some of them, we see a clear path

[33] For example, in programming environments like Scratch, kids can create sharable applications that will be used by millions of users in online communities. Students can interact as in social media, by commenting and appreciating others' work. Moreover, they can "see inside" projects, reading the "source code" made of blocks - in an open source community fashion, and most importantly "remix" projects, buy building new works based on others' creations (Kafai 2016).

which, sometimes starting with Wing's CT, goes back to Papert's CT, and which shows how understanding CT as (only) programming and programming-related concepts would be a gross limitation. We review here the most relevant ones for this paper.

diSessa (2018) proposes *computational literacy*: using computation to explore ideas, solve problems, and express yourself as you do with language literacy (Guzdial 2015). His work was directly influenced by Papert and shares important aspects of this form of CT. diSessa (2018, p. 24) clearly contrasts "coding academies" where programming is taught for its own sake[34]: as he sees computation as a new literacy, he emphasizes the need to "have something to say." With computation, likewise with literacy, it is not sufficient to learn grammar: you have to say something meaningful with your writings. His students confirmed that programming was engaging because they could do something interesting and important with computers, which clearly resonates with Papert's CT.

Kafai (2016) refers to *computational participation*, which involves "solving problems, designing systems, and understanding human behavior in the context of computing." It explicitly relates programming "to *make* and *be* in the digital world," and advocates "greater focus on underlying social and cultural dimensions of programming," thus moving "from tools to communities." The *make* reference is elaborated by Rode et al. (2015), in favor of computational *making*, where physical materials and the use of tangible interfaces are used for learning. They argue for the efficacy of using e-textiles to teach computing, but warn that, for *computational making*, "additional skills are needed beyond computational thinking to be successful," including "aesthetics, creativity, constructing, visualizing multiple representations, and understanding materials."

Tissenbaum et al. (2019) move even further, advocating *computational action*, where "while learning about computing, young people should also have opportunities to create with computing that have direct impact on their lives and their communities," in such a way to make "computing education more inclusive, motivating, and empowering for young learners." They outline two key dimensions of computational action—*computational identity* and *digital empowerment*. They relate computational identity to the "importance of young people's development of scientific identity for future STEM growth," quoting in this context Maltese and Tai (2010).

In this gradual expansion of perspectives, Guzdial et al. (2019) insist that CT is a moving target that cannot (and should not be) taken as the real goal. We must instead redesign computing to match the challenges of contemporary society, "along the lines of extending mathematics and systems organizations to model complex situations that go beyond our commonsense reasoning." This "will lead to much better programming language designs and environments as part of a larger curriculum made from the most powerful ideas about systems, processes, science, math, engineering, and computing itself." The reference to "programming language designs" is a much revealing one,[35] because it implicitly links this vision to what could be seen as one of the first incarnations of "Papert's CT," the idea of "procedural literacy," proposed by Sheil (1980) at Xerox PARC, in the context of the birth of Smalltalk—one of the first, and most influential, object-oriented programming languages, designed by Alan Kay, one of the authors of Guzdial et al. (2019). According to Kay and Goldberg (1977, p. 31), the language—implemented in a system called Dynabook—was intended to be used "as a programming and problem-solving tool; as an interactive memory for the storage

[34] As should be clear after reading this work, we believe by contrast that CS is *also* important for its own sake.
[35] Another one: "Intellectually honest computing-based models for understanding systems can lead to much better notion of programming."

and manipulation of data; as a text editor; and as a medium for expression through drawing, painting, animating pictures, and composing and generating music." We are here at the core of Papert's inspiration[36] and contribution—computers and computer programming are that protean tool that could impersonate most other tools. They are meta-instruments that, used in the right way and in the right environment, could provide "representations to help thinking language, mathematics, computing" (Guzdial et al., 2019), and empower people with higher insights and deeper thinking.

Finally, as a further witness of the relevance of Papert's view in today's debate, Kafai et al. (2020) classify CS education approaches to CT along three different theoretical framings, each of them articulating specific educational goals. The first framing is *cognitive CT*, which "seeks to provide students with an understanding of key computational concepts, practices, and perspectives." The second is *situated CT*, which "stresses personal creative expression and social engagement as a pathway in becoming computationally fluent building on youth interest in digital media and production." The last one is *critical CT*, which reminds that "computing is not an unequivocal social good, and proposes an analytical approach to the values, practices, and infrastructure underlying computation." We believe it is not too far a stretch to see in cognitive CT the evolution of Wing's CT, and in situated CT the heir of Papert's one.

We see in Kafai et al. (2020) a position, articulated from a theoretical viewpoint, close to the one that we put forward in this paper from a historical point of view. Rather than "pitting the different framings against each other," the authors argue that all of them should be embraced in education research, because "the diversity of perspectives on computational thinking is a resource, and not a stumbling block."

6.2 Why Papert's CT

From Papert's CT, we must preserve the innovation potential of the s*amba schools of computation*, without letting them be marginalized into after-school 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 the school. Indeed, solving problems using computers and computer science principles forces one to think logically/systematically/procedurally, not because of 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 and share them, and build and explore with them. Furthermore, 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 imperative for social inclusion.

[36] Kay learned Papert's ideas from Marvin Minsky in summer 1967; he then met Papert and his group in winter 1968: "This encounter finally hit me with what the destiny of personal computing really was going to be: [...] a personal dynamic medium [which] had to extend into the world of childhood." The paper from which this paragraph is taken (Kay 1993), in Section *Smalltalk and Children* quotes Cesare Pavese: "To know the world we must construct it" (*This business of living*, 1952.) And Kay adds: "In other words, we make not just to have, but to know. But the having can happen without most of the knowing taking place." Which is Papert's critique to Piaget: to construct for knowing and not just for having, we must construct *meaningful* objects and relations.

If every subject (CS included) is taught in a more constructive, meaningful, and creative way, like in a "samba school," this will contribute to the empowerment of all students, in particular the young and the disadvantaged.

7 Conclusions

The inclusion of CT in the PISA 2021 study may be seen as the ultimate 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 against "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.

We believe that both "Wing's CT" and "Papert's CT," two related but different views, have been misinterpreted. Paradoxically, this misinterpretation made them initially popular, but risks being their condemnation if their most profound—and most important—features are not considered and understood.

(1) Wing's CT has been a way to shine a light on the urgent need to teach "core CS concepts," as a tool to understand and being an active participant in a digital society where computation is ubiquitous. Non-computer scientists focused on the (unverified) claims that this computational problem-solving strategy— often simplified with "coding"—could easily transfer to every human activity. Teaching computer science should instead focus on its "big ideas,[37]" fundamental to understand and act in a digital world, rather than on the technical details. Since general ideas are hard to teach and transfer (see, e.g., Lewis (2017)), details are of course needed, but only if instrumental for conveying the deep concepts, and always in the context of meaningful and deliberate learning.

(2) Papert's CT has been a way to shine a light on the fact that technology was (and often, still, is) used to replicate traditional transmissive instruction (*instructionist Computer-Aided Instruction*). People focused on the unverified claims (only apparently supported by Papert) that learning programming could automatically result in better thinking skills. What should be kept in mind is, instead, that programming a computer can be a formidable tool (or, more evocatively, a meta-tool) for thinking and for more meaningful learning (both cognitively and affectively—like what happens in samba schools) by creating and experimenting.

The many different stakeholders involved in the introduction of CS/CT/"coding" in K-12 education (scholars, policymakers, educators) should benefit from our analysis. Instead of focusing on unverified claims about transfer, they should consider the more profound aspects of Wing's CT and Papert's CT we brought up. The former should clarify that CS is the scientific discipline to be introduced and taught in K-12 (with CT being its scientific and cultural substratum, as a lens to understand the computational nature of today's digital world).

[37] For example: "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," and "digital systems communicate with each other using protocols" (Bell et al., 2018).

The latter should be the driving force for a strong emphasis not only on the cognitive aspects of the discipline, but on the immense educational value of social and affective involvement of the student into the construction of a (computational) artifact, and on the value of computation as an interdisciplinary tool for learning.

Code Availability n/a

Author Contribution The authors contributed equally to the research and the writing of the paper.

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement. The study has been funded by Università di Bologna, under institutional research grants (unlabelled). Martini has also received funds from the Collegium – Lyon Institute for Advanced Studies, for his sabbatical leave.

Data Availability n/a

Declarations

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

References

Aho, A. V. (2011). Ubiquity symposium: Computation and computational thinking. *Ubiquity, 2011*(January). https://doi.org/10.1145/1922681.1922682.

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*. Hoboken: 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.

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* (pp. 125–132). Paris: UNESCO.

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

Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). Recognizing computational thinking patterns. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 245–250). Dallas, TX, USA: Association for Computing Machinery. https://doi.org/10.1145/1953163.1953241.

Bell, T., & Lodi, M. (2019). Constructing Computational Thinking Without Using Computers. *Constructivist Foundations, 14*(3), 342–351.

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, 17*(54).

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

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).

CECE, Committee on European Computing Education. (2017). *Informatics education in Europe: Are we all in the same boat?* New York, NY: ACM. https://doi.org/10.1145/3106077.

Code.org, CSTA, & ECEP Alliance. (2020). 2020 *State of Computer Science Education: Illuminating disparities.* https://advocacy.code.org/stateofcs. Accessed 10 Feb 2021.

College Board. (2017). *AP computer science principles*. College Board. Retrieved from https://apcentral.collegeboard.org/pdf/ap-computer-science-principlescourse-and-exam-description.pdf. Accessed 10 Feb 2021.

Committee on Information Technology Literacy: Lawrence Snyder, A. V. A., Marcia Linn, Arnold Packer, Allen Tucker, Jeffrey Ullman, Andries Van Dam. (1999). *Being fluent with information technology*. Washington, D.C.: National Academy Press.

Corradini, I., Lodi, M., & Nardelli, E. (2017a). Computational thinking in Italian schools: Quantitative data and teachers' sentiment analysis after two years of "Programma il Futuro". In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 224–229). Bologna, Italy: Association for Computing Machinery. https://doi.org/10.1145/3059009.3059040.

Corradini, I., Lodi, M., & Nardelli, E. (2017b). Conceptions and misconceptions about computational thinking among Italian primary school teachers. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 136–144). Tacoma, Washington, USA: Association for Computing Machinery. https://doi.org/10.1145/3105726.3106194.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational thinking - a guide for teachers*. Swindon: Computing at School. Retrieved from https://eprints.soton.ac.uk/424545/. Accessed 10 Feb 2021.

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

Denning, P. J. (2013). The science in computer science. *Communications of the ACM, 56*(5), 35–38. https://doi.org/10.1145/2447976.2447988.

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

Denning, P. J., & Tedre, M. (2019). *Computational thinking*. Cambridge, Massachusetts: The MIT Press.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Communications of the ACM, 32*(1), 9–23. https://doi.org/10.1145/63238.63239.

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

diSessa, A. A. (2000). *Changing minds*. Cambridge, MA: The MIT Press.

Dijkstra, E. W. (1974). Programming as a discipline of mathematical nature. *The American Mathematical Monthly, 81*(6), 608–612.

diSessa, A. A. (2018). Computational literacy and "the big picture" concerning computers in mathematics education. *Mathematical Thinking and Learning, 20*(1), 3–31.

Fadel, C., Bialik, M., & Trilling, B. (2015). *Four-dimensional education*. Scotts Valley, CA: 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.

Fincher, S. A., & Robins, A. V. (Eds.). (2019). *The Cambridge handbook of computing education research* (1st ed.). Cambridge: Cambridge University Press. https://doi.org/10.1017/9781108654555.

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.

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology, 12*(3), 306–355. https://doi.org/10.1016/0010-0285(80)90013-4.

Google. (n.d.). Exploring computational thinking. http://g.co/exploringct - The page has now been removed, but can be found in the "CT overview" tab here. https://web.archive.org/web/20181001115843/https://edu.google.com/resources/programs/exploring-computational-thinking/. Accessed 10 Feb 2021.

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

Grover, S. (Ed.). (2020). *Computer science in K-12: An A to Z handbook on teaching programming*. Palo Alto, CA: Edfinity.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051.

Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics, 8*(6), 1–165. https://doi.org/10.2200/S00684ED1V01Y201511HCI033.

Guzdial, M., Kay, A., Norris, C., & Soloway, E. (2019). Computational thinking should just be good thinking. *CACM, 62*(11), 28–30.

Hundhausen, C. D., Farley, S. F., & Brown, J. L. (2009). Can direct manipulation lower the barriers to computer programming and promote transfer of training? An experimental study. *ACM Transactions on Computer-Human Interaction, 16*(3), 13:1-13:40. https://doi.org/10.1145/1592440.1592442.

Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011). *Computational thinking patterns*. Retrieved from https://eric.ed.gov/?id=ED520742. Accessed 10 Feb 2021.

ISTE, & CSTA. (2011a). *Computational thinking teacher resources*. Washington, DC: International Society for Technology in Education. Retrieved from https://id.iste.org/docs/ct-documents/ct-teacher-resources_2ed-pdf.pdf?sfvrsn=2. Accessed 10 Feb 2021.

ISTE, & CSTA. (2011b). Operational definition of computational thinking for K-12 education. Washington, DC: International Society for Technology in Education. Retrieved from https://id.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf?sfvrsn=2. Accessed 10 Feb 2021.

Jónasson, J. T. (2016). Educational change, inertia and potential futures. *European Journal of Futures Research, 4*, 7.

Juškevičienė, A., & Dagienė, V. (2018). Computational thinking relationship with digital competence. *Informatics in Education, 17*(2), 265–284. https://doi.org/10.15388/infedu.2018.14.

K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org. Accessed 10 Feb 2021.

Kafai, Y. B. (2016). From computational thinking to computational participation in K-12 education. *CACM, 59*(8), 26–27.

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

Kafai, Y. B., Proctor, C., & Lui, D. (2020). From theory bias to theory dialogue: Embracing cognitive, situated, and critical framings of computational thinking in K-12 CS education. *ACM Inroads, 11*(1), 44–53. https://doi.org/10.1145/3381887.

Kalelioğlu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing, 4*(3), 583–596.

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

Kay, A. C. (1993). The early history of Smalltalk. *SIGPLAN Not, 28*(3), 69–95. http://doi.acm.org/10.1145/155360.155364.

Kay, A., & Goldberg, A. (1977, March). Personal dynamic media. *Computer, 10*(3), 31–41.

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

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

Krauss, J., & Prottsman, K. (2016). *Computational thinking and coding for every student: The teacher's getting-started guide*. Thousand Oaks, CA: Corwin Press.

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, pp. 1–40). Greenwich, CT: 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.

Li, Y., Schoenfeld, A. H., diSessa, A. A., Graesser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2019). On Thinking and STEM Education. *Journal for STEM Education Research, 2*(1), 1–13. https://doi.org/10.1007/s41979-019-00014-x.

Lodi, M. (2020). Informatical Thinking. *Olympiads in Informatics: An International Journal, 14*, 113. https://doi.org/10.15388/ioi.2020.09.

Maltese, A. V., & Tai, R. H. (2010). Eyeballs in the fridge: Sources of early interest in science. *International Journal of Science Education, 32*(5), 669–685.

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? *Communications of the ACM, 29*(7), 605–610.

McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the 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.

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

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

National Research Council. (2000). *How people learn: Brain, mind, experience, and school: Expanded edition*. Washington, DC: The National Academies Press. Retrieved from https://www.nap.edu/catalog/9853/how-people-learn-brain-mind-experience-and-school-expanded-edition. Accessed 10 Feb 2021.

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.

NGSS (2013). *The next generation science standards*. Appendix F: "Scientific and Engineering Practices". https://www.nextgenscience.org. Accessed 10 Feb 2021.

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.

NRC. (1996). *National science education standards*. Washington DC: National Academy Press.

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

Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. New York, NY, USA: Basic Books, Inc..

Papert, S. (1996a). *The connected family: Bridging the digital generation gap*. Atlanta, GA: Long Street Press.

Papert, S. (1996b). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning, 1*(1), 95–123.

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). *Keynote lecture.* Keynote at ICMI 17 Conference in Hanoi, Viet Nam. Retrieved from http://dailypapert.com/wp-content/uploads/2012/05/Seymour-Vietnam-Talk-2006.pdf. Accessed 10 Feb 2021.

Papert, S., & Harel, I. (1991). Situating constructionism. In S. Papert & I. Harel (Eds.), *Constructionism (chap. 1)*. Norwood, NJ: Ablex Publishing Corporation.

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*. London: Penguin Books.

Prottsman, K. (2015). Coding vs. Programming – Battle of the Terms! *HuffPost*. Retrieved from https://www.huffpost.com/entry/coding-vs-programming-bat_b_7042816. Accessed 10 Feb 2021.

Reimers, F. M., & Chung, C. K. (2016). *Teaching and learning for the twenty-first century*. Cambridge: Harvard Educational Publishing Group.

Robins, A. V., Margulieux, L. E., & Morrison, B. B. (2019). Cognitive sciences for computing education. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (1st ed., pp. 231–275). Cambridge: Cambridge University Press. https://doi.org/10.1017/9781108654555.010.

Rode, J. A., Weibert, A., Marshall, A., Aal, K., von Rekowski, T., El Mimouni, H., & Booker, J. (2015). From computational thinking to computational making. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (pp. 239–250). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2750858.2804261.

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

Schanzer, E., Fisler, K., Krishnamurthi, S., & Felleisen, M. (2015). Transferring skills at solving word problems from computing to algebra through bootstrap. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, *Presented at the SIGCSE '15: The 46th ACM Technical Symposium on Computer Science Education, Kansas City* (pp. 616–621). Missouri USA: ACM. https://doi.org/10.1145/2676723.2677238.

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.

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2019). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology, 111*(5), 764–792. https://doi.org/10.1037/edu0000314.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition [Project Report]. *University of Southampton (E-prints)*. Retrieved from https://eprints.soton.ac.uk/356481/. Accessed 10 Feb 2021.

Shackelford, R., Lunt, B., McGettrick, A., Sloan, R., Topi, H., Davies, G., et al. (2006). Computing curricula 2005: The overview report. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education - SIGCSE '06 (p. 456)*, *Presented at the 37th SIGCSE technical symposium*. Houston, Texas, USA: ACM Press. https://doi.org/10.1145/1121341.1121482.

Sheil, B. A. (1980). Teaching procedural literacy (presentation abstract). In *Proceedings of the ACM 1980 annual conference* (pp. 125–126). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/800176.809944.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003.

Tedre, M. (2014). *The science of computing: Shaping a discipline*. New York: 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.

Tissenbaum, M., Sheldon, J., & Abelson, H. (2019). From computational thinking to computational action. *CACM, 62*(3), 34–36.

Tyack, D., & Cuban, L. (1995). *Tinkering toward utopia: a century of public school reform*. Cambridge: Harvard University Press.

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies, 20*(4), 715–728. https://doi.org/10.1007/s10639-015-9412-6.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5.

Weintrop, D., & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education*, 142, 103646. https://doi.org/10.1016/j.compedu.2019.103646.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118.

Wing, J. M. (2011). Research notebook: Computational thinking—what and why. *The Link Magazine*, Pittsburg, PA: Carn mputer Science. Retrieved from https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why. Accessed 10 Feb 2021.

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