Informatical Thinking

Michael Lodi

Informatical Thinking[1]

Michael LODI
Department of Computer Science and Engineering, INRIA Focus, Lab. CINI Informatica e Scuola
Alma Mater Studiorum – Università di Bologna
Bologna, Italy
michael.lodi@unibo.it

Abstract: In this paper, we reviewed many definitions of computational thinking, finding they share a lot of common elements, of very different nature. We classified them in mental processes, methods, practices, and transversal skills. Many of these elements seem to be shared with other disciplines and resonate with the current narrative on the importance of 21st-century skills. Our classification helps on shedding light on the misconceptions related to each of the four categories, showing that, not to dilute the concept, elements of computational thinking should be intended inside the discipline of Informatics, being its "disciplinary way of thinking".

Keywords: computational thinking, informatics, misconceptions, definition, disciplinary way of thinking, informatical thinking.

1. Introduction[2]

The expression *computational thinking* (CT, from now on) seems to have been firstly used in print by Seymour Papert (1980) and then was brought to the attention of the informatics community by Jeannette Wing (2006).

From 2006, a considerable body of literature has been produced to search for a better definition of this concept, to provide tools and frameworks, to introduce and assess CT in K-12 education.

Even if there is no agreement between authors, a lot of proposed definitions stress the fact that CT is not only about technical methods and practices, but also about mental processes and transversal competences[3] like *creativity, collaboration, tolerance for ambiguity, resilience,* and more. However, educational and psychological research warns about optimistic claims on the transfer of competences from a discipline to other far domains and to general skills.

In this paper, we review some of the most important definitions emerged in the last years and propose a classification of the common elements that can be useful to better frame the misinterpretations of the concepts.

We will argue that CT must maintain its bond with informatics, representing its "disciplinary way of thinking".

2. Definitions of computational thinking[4]

The expression "computational thinking" was brought back to the informatics community by Wing (2006), gaining massive attention[5]. In that seminal article, Wing did not give a definition, but related the concept to informatics, stating "*Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science*", arguing that "*thinking like computer scientists*" would be a benefit for everyone, not just for professionals or scientists.

---

[1]This is an authors' pre-print version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in Olympiads in Informatics, Volume 14 (2020), 2020, pp. 113–132, DOI: 10.15388/ioi.2020.09, https://ioinformatics.org/journal/v14_2020_113_132.pdf

[2] This paper is based on material from author's PhD thesis (Lodi, 2020).

[3] Often referred also as *transversal skills*, *soft skills* or *key competences*, in the context of EU documents, in particular in the "*Personal, social and learning competence*," see for example http://data.consilium.europa.eu/doc/document/ST-5464-2018-ADD-2/EN/pdf

[4] This section is an expansion of the work presented in Corradini et al. (2017b).

[5] Currently (July 2020), the paper has more than 6300 citations, according to Google Scholar.

In these years, many definitions have been proposed. In Corradini et al. (2017b) authors started from five of the most important definitions to find out constituting elements of CT. Juškevičienė & Dagienė (2018) schematized many of the definitions proposed from Papert's views up to 2017. We review all of them in the following table. In the third column, we provide pointers to the classification that we will propose in Section 3.

Paper	**Description**	**Elements (Sec. 3)**
Wing (2006)	In her seminal article, Wing informally defines CT as *"thinking like computer scientists"*.	1)
Wing (2008, 2011)	Wing defines more formally CT as "*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*". This definition is attributed to Jan Cuny, Larry Snyder, and Jeannette M. Wing, in an unpublished work from 2010: "Demystifying Computational Thinking for Non-Computer Scientists," referenced by Wing (2011) herself. Moreover, Wing says it was originated by a discussion with Alfred Aho. Wing also identifies characteristic elements of CT. In particular, she states the most important elements are *abstraction* (the "mental" tool of computing) and *automation* (by using a computer, the "metal" tool of computer scientists): "*computing is the automation of our abstractions*" (Wing, 2008). Wing (2011) recognises significant overlapping or inclusions between CT and other types of thinking: logical thinking, algorithmic thinking, parallel thinking, compositional reasoning, pattern matching, procedural thinking, and recursive thinking.	1) d) 2) a) 1) a) b) 2) c) e) f)
Aho (2011)	Alfred Aho provides a definition very similar to the Cuny-Snyder-Wing one, 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.*" 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.	1) a)
ISTE&CSTA (2011a,b)	ISTE & CSTA proposed an operational definition, targeting specifically K-12 educators. They define CT as a problem-solving process that includes (but is not limited to) the following characteristics: • *Formulating problems in a way that enables us to use a computer and other tools to help solve them* • *Logically organizing and analyzing data* • *Representing data through abstractions such as models and simulations* • *Automating solutions through algorithmic thinking (a series of ordered steps)* • *Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources* • *Generalizing and transferring this problem-solving process to a wide variety of problems* Moreover, they state that CT is "*supported and enhanced by a number of dispositions or attitudes*" that include • *Confidence in dealing with complexity* • *Persistence in working with difficult problems* • *Tolerance for ambiguity* • *The ability to deal with open ended problems* • *The ability to communicate and work with others to achieve a common goal or solution* Finally they propose a CT vocabulary (ISTE & CSTA, 2011a), listing a set of CT terms with a brief definition or explanation: • *Data Collection* • *Data Analysis* • *Data Representation* • *Problem Decomposition* • *Abstraction*	1) a) c) d) f) 2) a) b) c) d) e) 4) b) d) e)

	• *Algorithms and Procedures* • *Automation* • *Simulation* • *Parallelization*	
Google (n.d.)	Google assumes the same ISTE/CSTA definition but - instead of a vocabulary - lists and (re)defines a series of CT concepts, pointing out that they are *mental processes* or *tangible outcomes*: • *Abstraction* • *Algorithm Design* • *Automation* • *Data Analysis* • *Data Collection* • *Data Representation* • *Decomposition* • *Parallelization* • *Pattern Generalization* • *Pattern Recognition* • *Simulation*	1) a) c) d) e) f) 2) a) b) c) d)

Brennan & Resnick (2012)	Brennan and Resnick present a *computational thinking framework* to describe learning and development that take place when designing and programming interactive media with Scratch platform. They state CT involves three dimensions:
	• *computational concepts* designers employ as they program:
	  ○ *sequences*
	  ○ *loops*
	  ○ *parallelism*
	  ○ *events*
	  ○ *conditionals*
	  ○ *operators*
	  ○ *data*
	• *computational practices* designers develop as they program:
	  ○ being incremental and iterative,
	  ○ testing and debugging,
	  ○ reusing and remixing
	  ○ abstracting and modularizing;
	• *computational perspectives* designers form about the world around them and about themselves:
	  ○ expressing
	  ○ connecting
	  ○ questioning
		1) a) c) d)
		
		2) b) c) f)
		
		3) a) b) c)
		
		4) a) b) c)

Csizmadia et al. (2015)	Computing at school assume a Wing-like definition: CT is "*learning to think in ways which allow us, as humans, to solve problems more effectively and, when appropriate, use computers to help us do so*" and then state it involves
	• six concepts
	  ○ *Logic*
	  ○ *Algorithms*
	  ○ *Decomposition*
	  ○ *Patterns*
	  ○ *Abstraction*
	  ○ *Evaluation*
	• five approaches
	  ○ *Tinkering*
	  ○ *Creating*
	  ○ *Debugging*
	  ○ *Persevering*
	  ○ *Collaborating*
		1) a) b) c) d) e) f)
		
		2) e)
		
		3) a) b)
		
		4) a) b) e)

Grover and	After a literature review, assumed the Aho-Cuny-Snyder-Wing definition and agreed	1) a) b)

Pea (2013)	that the following elements are accepted: • *Abstractions and pattern generalizations (including models and simulations)* • *Systematic processing of information* • *Symbol systems and representations* • *Algorithmic notions of flow of control* • *Structured problem decomposition (modularizing)* • *Iterative, recursive, and parallel thinking* • *Conditional logic* • *Efficiency and performance constraints* • *Debugging and systematic error detection*	c) d) f) 2) b) c) d) e) f) 3) b)
Selby and Woollard (2013)	In a widely referenced Technical Report, Selby and Woollard examined a number of CT definitions, and argued that the most relevant and useful elements are: • *thought process* • *abstraction* • *decomposition* • *algorithmic thinking* • *evaluation* • *generalization*	1) a) c) d) f) 2) e)
Weintrop et al. (2016)	They propose a definition of CT for mathematics and science. From a literature review, they start with an initial set of activities: • *Ability to deal with open-ended problems* • *Persistence in working through challenging problems* • *Confidence in dealing with complexity* • *Representing ideas in computationally meaningful ways* • *Breaking down large problems into smaller problems* • *Creating abstractions for aspects of problem at hand* • *Reframing problem into a recognizable problem* • *Assessing strengths/weaknesses of a representation of data/representational system* • *Generating algorithmic solutions* • *Recognizing and addressing ambiguity in algorithms* After that, by analyzing CT activities for math and science, propose the "Computational thinking in mathematics and science taxonomy". • *Data practices:* o *Collecting Data* o *Creating Data* o *Manipulating Data* o *Analyzing Data* o *Visualiging Data* • *Modeling and simulation practices:* o *Using Computational Models to Understand a Concept* o *Using Computational Models to Find and Test solutions* o *Assessing Computational Models* o *Designing Computational Models* o *Constructing Computational Models* • *Computational problem solving practices:* o *Preparing Problems for Computational Solutions* o *Programming* o *Choosing Effective Computational Tools* o *Assessing Different Approaches/Solutions to a Problem* o *Developing Modular Computational Solutions* o *Creating Computational Abstractions* o *Troubleshooting and Debugging* • *Systems thinking practices:* o *Investigating a Complex System as a Whole* o *Understanding the Relationships within a System*	1) a) d) e) f) 2) b) d) e) f) 3) b) 4) a) d) e)

	○ *Thinking in levels* ○ *Communicating Informations about a System* ○ *Defining Systems and Managing Complexity*	
Kalelioğlu et al. (2016)	They view CT as a "*complex higher-order thinking, skills may require to use the power of human cognitive ability and embrace the support of machines to think and solve problems.*" They propose a "Framework for Computational Thinking as a Problem Solving Process" in five steps. • *Identify the problem* ○ *Abstraction* ○ *Decomposition* • *Gathering, representing and analysing data* ○ *Data collection* ○ *Data analysis* ○ *Pattern recognition* ○ *Conceptualising* ○ *Data representation* • *Generate, select and plan solutions* ○ *Mathematical reasoning* ○ *Building algorithms and procedures* ○ *Parallelisation* • *Implement solutions* ○ *Automation* ○ *Modelling and simulations* • *Assessing solutions and continue for improvement* ○ *Testing* ○ *Debugging* ○ *Generalisation*	1) c) d) e) f) 2) a) b) c) d) e) 3) b) 4) a)
Krauss and Prottsman (2016)	Krauss and Prottsman define[6] CT as *using thinking patterns and processes to pose and solve problems or prepare programs for computation*. Lessons plans are given for the following categories: • *Decomposition (data analysis)* • *Pattern matching (data visualization)* • *Abstraction (data modelling, pattern generalization)* • *Automation (algorithm design, parallelization, simulation)*	1) a) c) d) e) 2) a) b) c) d) 4) a)
Shute et al. (2017)	After an extensive literature review, they provide a very general definition of CT: "*the conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts.*". They then recognize the following categories and subcategories, giving however explanations that are quite general and far from Informatics. • *Decomposition* • *Abstraction (data collection and analysis, pattern recognition, modelling)* • *Algorithms (algorithm design, parallelism, efficiency, automation)* • *Debugging* • *Iteration* • *Generalization*	1) a) c) d) e) f) 2) a) b) c) d) e) 3) a) b) 4) a)
College Board (2017)	Proposes a CT framework for the *AP CS Principles* course, made of six practices. • *Connecting Computing* ○ *Identify* impacts *of computing.* ○ *Describe connections between people and computing.* ○ *Explain connections between computing concepts.* • *Creating Computational Artifacts* ○ *Create a computational artifact with a practical, personal, or societal intent.*	1) a) d) 2) b) d) e) 3) b) 4) a) b) c)

[6] As cited in Juškevičienė and Dagienė (2018, p. 270)

	○ *Select appropriate techniques to develop a computational artifact.* ○ *Use appropriate algorithmic and information management principles.* • *Abstracting* ○ *Explain how data, information, or knowledge is represented for computational use.* ○ *Explain how abstractions are used in computation or modeling.* ○ *Identify abstractions.* ○ *Describe modeling in a computational context.* • *Analyzing Problems and Artifacts* ○ *Evaluate a proposed solution to a problem.* ○ *Locate and correct errors.* ○ *Explain how an artifact functions.* ○ *Justify appropriateness and correctness of a solution, model, or artifact.* • *Communicating* ○ *Explain the meaning of a result in context.* ○ *Describe computation with accurate and precise language, notations, or visualizations.* ○ *Summarize the purpose of a computational artifact.* • *Collaborating* ○ *Collaborate with another student in solving a computational problem.* ○ *Collaborate with another student in producing an artifact.* ○ *Share the workload by providing individual contributions to an overall collaborative effort.* ○ *Foster a constructive, collaborative climate by resolving conflicts and facilitating the contributions of a partner or team member.* ○ *Exchange knowledge and feedback with a partner or team member.* ○ *Review and revise their work as needed to create a high-quality artifact*.	
Juškevičienė & Dagienė (2018)	After reviewing many definitions, Juškevičienė & Dagienė found eight CT components groups. • *Data analysis & representation* ○ *Data collection* ○ *Data analysis* ○ *Data representation* ○ *Generalisation* ○ *Patterns finding* ○ *Drawing conclusions* • *Computing Artefacts* ○ *Artefact development* ○ *Artefact designing* • *Decomposition* ○ *Breaking into parts* • *Abstraction* ○ *Details suppression* ○ *Modelling* ○ *Information filtering* • *Algorithms* ○ *Sequence of steps* ○ *Procedures* ○ *Set of instructions* ○ *Automation* • *Communication & collaboration* ○ *Communication* ○ *Collaboration* ○ *Computational analysis* • *Computing & Society*	1) a) b) c) d) e) f) 2) a) b) d) e) 3) b) 4) a) b) c)

	○ *Computing influence* ○ *Computing implication* ○ *Computing concepts* • *Evaluation* ○ *Evaluation* ○ *Correction*	
Denning and Tedre (2019)	Denning and Tedre, in their book about CT, proposed the following definition: *Computational thinking is the mental skills and practices for* • *designing computations that get computers to do jobs for us, and* • *explaining and interpreting the world as a complex of information processes.* Moreover, they distinguish between "CT for beginners" (the one that is spreading in K-12 education, teaching basic computational problem solving) with "CT for professionals" (the one used by cutting-edge engineers and scientists in all fields as a powerful professional tool). They recognize CT has six important dimensions, "windows" looking at CT: • *Methods* • *Machines* • *Computing Education* • *Software Engineering* • *Design* • *Computational Science*	4) a) c) 2) a) d)

3. Comparison

We compared the CT elements found in the analysed definitions.

Those who give a precise definition agree on the fact that **CT is a way of thinking** (*thought process*) for **problem solving**. They all somehow specify that it is a ***computational*** (rather than general) *problem solving*: the formulation and the solution of the problem must be expressed in a way that allows an "external" processing agent (a human or a machine) to carry it out.

Apart from the general statement, all definitions list some constitutive elements of CT. These elements are of very different kinds (from thinking habits to specific programming concepts), and many authors group them in categories, but there is no universal agreement on the classification.

We classified all the elements into four categories. For each category, we list the elements, trying to summarise all aspects stated in the analysed definitions. Instead of an "intersection approach", keeping only the elements that had a wider consensus (like what was done by Selby & Woollard (2013)), we used a "union approach", trying to build a comprehensive list of all the characteristics proposed by different authors.

1) **Mental/thought processes**: mental strategies useful to solve problems.
   a) *Algorithmic thinking*: use algorithmic/procedural thinking (ISTE & CSTA, 2011b; Wing, 2008, 2011) to design a sequence of ordered step (instructions) to solve a problem, achieve a goal or perform a task (Brennan & Resnick, 2012; Csizmadia et al., 2015; Google, n.d.; ISTE & CSTA, 2011a). Also recognised by (College Board, 2017; Grover & Pea, 2013; Juškevičienė & Dagienė, 2018; Krauss & Prottsman, 2016; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016).
   b) *Logical thinking*: use logical thinking (Wing, 2011) and reasoning to make sense of things, establish and check facts (Csizmadia et al., 2015). Also recognised by (Grover & Pea, 2013; Juškevičienė & Dagienė, 2018).
   c) *Problem decomposition and modularisation*: split a complex problem into simpler subproblems to solve it more easily (Csizmadia et al., 2015; Google, n.d.; ISTE & CSTA, 2011a); modularise (Brennan & Resnick, 2012); use compositional reasoning (Wing, 2008). Also recognised by (Grover & Pea, 2013; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Krauss & Prottsman, 2016; Selby & Woollard, 2013; Shute et al., 2017).
   d) *Abstraction*: get rid of useless details to focus on relevant information or ideas (Brennan & Resnick, 2012; Csizmadia et al., 2015; Google, n.d.; ISTE & CSTA, 2011a; Wing, 2011). Also recognised by (College Board, 2017; Grover & Pea, 2013; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Krauss & Prottsman, 2016; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016).

e) *Pattern recognition*: discover and use regularities in data and problems (Csizmadia et al., 2015; Google, n.d.; Wing, 2011). Also recognised by (Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Krauss & Prottsman, 2016; Shute et al., 2017; Weintrop et al., 2016).

f) *Generalisation*: use discovered similarities to make predictions or to solve more general problems (Csizmadia et al., 2015; Google, n.d.; ISTE & CSTA, 2011b). Also recognised by (Grover & Pea, 2013; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016).

2) **Methods**: operational approaches widely used by computer scientists.

a) *Automation*: automate the solutions (ISTE & CSTA, 2011b; Wing, 2008); use a computer or a machine to do repetitive tasks (Google, n.d.; ISTE & CSTA, 2011a). Also recognised by (Denning & Tedre, 2019; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Krauss & Prottsman, 2016; Shute et al., 2017).

b) *Data collection, analysis and representation*: gather information/data, make sense of them by finding patterns, represent them properly (Google, n.d.; ISTE & CSTA, 2011a); store, retrieve and update values (Brennan & Resnick, 2012). Also recognised by (College Board, 2017; Grover & Pea, 2013; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Krauss & Prottsman, 2016; Shute et al., 2017; Weintrop et al., 2016).

c) *Parallelisation*: carry out tasks simultaneously to reach a common goal (Brennan & Resnick, 2012; Google, n.d.; ISTE & CSTA, 2011a), use parallel thinking (Wing, 2011). Also recognised by (Grover & Pea, 2013; Kalelioğlu et al., 2016; Krauss & Prottsman, 2016; Shute et al., 2017).

d) *Modelling and simulation*: represent data and (real world) processes through models (Google, n.d.; ISTE & CSTA, 2011b), run experiments on models (ISTE & CSTA, 2011a). Also recognised by (College Board, 2017; Denning & Tedre, 2019; Grover & Pea, 2013; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Krauss & Prottsman, 2016; Shute et al., 2017; Weintrop et al., 2016).

e) *Analysis and evaluation*: implement and analyse solutions (ISTE & CSTA, 2011a) to judge them (Csizmadia et al., 2015), in particular for what concerns effectiveness, and efficiency in terms of time and resources. Also recognised by (College Board, 2017; Grover & Pea, 2013; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016).

f) *Programming*: use some common concepts in programming (e.g. loops, events, conditionals, mathematical and logical operators (Brennan & Resnick, 2012), recursion (Wing, 2011)). Also recognised by (Grover & Pea, 2013; Weintrop et al., 2016).

3) **Practices**: typical practices used in the implementation of computing machinery based solutions.

a) *Experimenting, iterating, tinkering*: in iterative and incremental software development, one develops a project with repeated iterations of a design-build-test cycle, incrementally building the final result (Brennan & Resnick, 2012); tinkering means trying things out using a trial and error process, learning by playing, exploring, and experimenting (Csizmadia et al., 2015). Also recognised by (Shute et al., 2017).

b) *Test and debug*: verify that solutions work by trying them out (Brennan & Resnick, 2012); find and solve problems (bugs) in a solution/ program (Csizmadia et al., 2015). Also recognised by (College Board, 2017; Grover & Pea, 2013; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Shute et al., 2017; Weintrop et al., 2016).

c) *Reuse and remix*: build your solution on existing code, projects, ideas (Brennan & Resnick, 2012).

4) **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.

a) *Design and create*: design and build things (Csizmadia et al., 2015) and computational artifacts, use computation to be creative and express yourself (Brennan & Resnick, 2012). Also recognised by (College Board, 2017; Denning & Tedre, 2019; Juškevičienė & Dagienė, 2018; Kalelioğlu et al., 2016; Krauss & Prottsman, 2016; Shute et al., 2017; Weintrop et al., 2016).

b) *Communicate and collaborate*: connect with others and work together to create something with a common goal and to ensure a better solution (Brennan & Resnick, 2012; Csizmadia et al., 2015; ISTE & CSTA, 2011b). Also recognised by (College Board, 2017; Juškevičienė & Dagienė, 2018).

c) *Reflect, learn, meta-reflect, understand the world computationally*: use computation to reflect and understand computational aspects of the world (Brennan & Resnick, 2012), identify impacts of computing on society (College Board, 2017). Also recognised by (Denning & Tedre, 2019; Juškevičienė & Dagienė, 2018).

d) *Be tolerant for ambiguity*: deal with non-well specified and open-ended, real-world problems(ISTE & CSTA, 2011b). Also recognised by (Weintrop et al., 2016).

e) *Be persistent when dealing with complex problems*: be confident in working with difficult or complex problems (ISTE & CSTA, 2011a), persevering, being determined, resilient and tenacious (Csizmadia et al., 2015). Also recognized by (Weintrop et al., 2016).

4. CT as Informatics "disciplinary way of thinking"

Note that many of the cited elements in the previous classification are broad and general. This led to some critiques (for an overview, see Martins-Pacheco et al. (2020)): some of these concepts are not exclusively associated with Informatics, but taught in other disciplines (e.g., Math and Sciences) or are general skills that children have been learning for a long time before the birth of Informatics. Cansu & Cansu (2019), summarising critics from Hemmendinger (2010), stated for example that:

- *Reformulating hard problems is typical of all domains of problem solving,*
- *Philosophers have been thinking about thinking — recursively — for a long time,*
- *Mathematics surely uses abstraction, and so do all disciplines that build models,*
- *Separation of concerns and using heuristics also characterizes problem-solving in general.*

By contrast, other authors argue that computing features extend and differentiate these elements from other domains (Grover & Pea, 2013), and provide some characteristic problem-solving methods (e.g., the possibility to effectively execute a solution, a model, an abstraction by running an implementation of its algorithm (Martini, 2012)).

We agree, arguing that Informatics is what needs to be taught in schools, and CT is, at most, the *conceptual sediment* of that teaching, what remains even when the technical aspects have been forgotten (Lodi et al., 2017).

In other disciplines, like Math, it is recurrent to talk about mathematical thinking (Sternberg & Ben-Zeev, 1996), or mathematical reasoning (English, 1997), or mathematical problem solving (Schoenfeld, 1985).

Like what mathematical thinking is for Math, CT is Informatics "disciplinary way of thinking" (Pace & Middendorf, 2004) (and this explains the provocative title "Informatical thinking").

Chick et al. (2009), talking about *signature pedagogies* (pedagogies to "engage students in the ways of knowing, the habits of mind, and the values shared by experts in [a] field" (Gurung et al., 2009, p. xvii)), affirms that "effective teaching results from core values and principles of our courses and of our disciplines, rather than from generic views of learning. […] higher-level thinking is inhibited by such generic conceptions and lays the groundwork for questions about the central values, habits, and ways of thinking within their disciplines" (Chick et al., 2009, p. 4).

According to Li et al. (2019, p. 8)

> […] domain-specific thinking and domain-general thinking are not dichotomous, as thinking itself is a complex process involving many different components. Domain-general thinking is often derived from human's thinking performance across different knowledge-lean (e.g., solving a puzzle) or -rich task domains (e.g., solving algebraic equations). Domain-specific thinking is often characterized in terms of its disciplinary content but also involves more general cognitive components. In other words, domain-specific thinking should contain both domain-specific and -general aspects of cognitive activities. For example, a mathematician's thinking is scarcely only mathematics (the knowledge component). It can share possible common elements with a biologist's thinking (e.g., certain aspects of metacognition and meta-representation). The same reasoning applies to students' thinking in specific disciplines. […] 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).

While we agree disciplinary thinking contains both specific and general aspects, we keep spotting the tendency of educators and policymakers to focus, for what concerns CT, only on the more general ones.

Voogt et al. (2015) recognise, in some of the abovementioned definitions of CT, a tension between "*the 'core' qualities of CT versus certain more 'peripheral' qualities*". The latter highly overlap with what we called "transversal skills," and we agree with Voogt that this "*runs the risk of diluting the idea of CT, blurring and making it indistinct from other 21st century skills*".

As CT movement has grown in educational contexts, many unverified claims about the effects of learning CT/Informatics has emerged (e.g., that it will *automatically* transfer to thinking logically, better problem solving in every aspect of life, developing perseverance, getting better results in math and science, and so on (Lewis, 2017)). Most of these claims are not supported by research, and "*appear in blog posts, opinion pieces, and other 'grey literature'*" (Duncan, 2019, p. 32). As Denning & Tedre (2019, p.xiii) put it:

> [CT] is sometimes portrayed as a universal approach to problem solving. Take a few programming courses, the story in the popular media goes, and you will be able to solve problems in any field. Would that this were true! Your ability to solve a problem for someone depends on your understanding of their context in which the problem exists. For

> instance, you cannot build simulations of aircraft in flight without understanding fluid dynamics. You cannot program searches through genome databases without understanding the biology of the genome and the methods of collecting the data. Computational thinking is powerful, but not universal.

As we will see, non-specialist teachers that most probably never studied Informatics in their schooling or training may tend to stick to some "general versions" of the listed characteristics (and especially on the peripheral qualities), not necessarily related to Informatics.

We believe, by contrast, CT must be understood *inside* Informatics: while many characteristics are (more or less apparently) shared with other disciplines, we need to focus on their specific "informatical" instantiation.

We believe that the classification we proposed in the previous section is a good tool to frame the misconceptions about CT and Informatics in K-12 education (Denning, 2017; Denning et al., 2017). In the next four subsections, we will discuss in this light the four main categories we recognised.

4.1 Mental Processes

Mental processes are, on the surface, shared with other disciplines, but should be understood and experienced as Informatics specific. "CT draws on a rich legacy of related frameworks as it extends previous thinking skills" (Lee et al., 2011, p. 32).

First of all, analysed definitions are clear in stressing on the *computational* (rather than *general*) nature of *problem solving*. In facts, many authors (recall for example Aho's position) agree that what differentiates algorithmic thinking (which has been used for centuries, firstly by mathematicians) and computational thinking is the *automation* of the algorithm (Stephens & Kadijevich, 2020).

Next, as diSessa points out, *abstraction*, one of the core CT concepts according to Wing and many others, has different nuances in different disciplines:

> Mathematical abstraction (let's call it, inferential abstraction) occurs in order to build conceptual worlds where a small set of attributes fully define entities, resulting in a substantial inferential fabric—a family of basic ideas and secure inferences (proofs) from them to other ideas (theorems). Abstraction in physics (empirical abstraction) is taking a look at the world and finding in it new things that cut away certain details, but build on others that might initially be completely ignored, in order to create core models that apply across a very wide range of circumstances. Mathematicians do not, in general, need or use the skill of "peeling away" from the world as it exists, nor digging through the difficulties of finding out how the world is in the first place, nor do they have the constraint of confirming empirically that the world admits in a certain abstraction, usually within prescribed limits. Abstraction in computer science (practical abstraction) resides substantially in peeling away the irrelevant particulars of an implementation so that one only need think about the features of a piece of it that are essential for a given use—its "contract" with the rest of the program. (diSessa, 2018, p. 21)

Moreover, it is hugely debated if general skills (like *general problem solving, critical thinking, creative thinking, decision making, and so on*) exist, are transferrable or even teachable (for a comprehensive review, see (Lodi, 2020, ch. 4)). For example, Gick & Holyoak (1980) found that *problem decomposition*, one of the most highlighted aspects of CT (Guzdial, 2019), is not easily transferrable. They described to students a situation where an army had to be divided into small groups to successfully attack a fortress; immediately after they asked the students how to attack a tumour with a laser without damaging healthy tissues. The vast majority of the students were not able to use the same approach (divide the laser in multiple weaker beams). They only managed to do so when explicitly prompted to think at the army example.

That is why these skills should be taught in an Informatics-specific context.

4.2 Informatics methods

Many methods are, again, shared with other disciplines, but we believe they must be experienced in the context of automatic elaboration of information.

Emblematic examples are *unplugged activities*, teaching activities not using a computer or tablet or smartphone to teach informatics concepts and methods. After experiencing the activities without computers, it is essential to relate

what students have done to the specific informatics context, to understand what happens on physical devices. Bell & Vahrenhold (2018) suggest that Unplugged activities offer best results when used in combination with "plugged approaches" (i.e. programming tasks). Using unplugged activities before the plugged one seems to foster even *more* effective results than the programming activities alone.

Two early studies discovered that, without this explicit connection, "*the program* [based on CS Unplugged] *had no statistically significant impact on student attitudes toward computer science or perceived content understanding*" (Feaster et al., 2011) and that "*the students' attitudes and intentions regarding CS did not change in the desired direction*" (Taub et al., 2012).

4.3 Informatics practices

Listed practices are, of course, shared with other disciplines and activities. As we will also discuss for transversal competences, we should not justify the introduction of Informatics in K-12 education mainly for teaching this kind of general approaches (which have been used and taught for centuries before the advent of Informatics).

However, we agree that computers are powerful tools to "concretely experiment with". This is one aspect of the constructionist learning theory from Seymour Papert (see Lodi (2020, chapters 3, 6)).

Already in 1970, Papert and colleagues, while designing and experimenting with the LOGO programming language, noted (Feurzeig et al., 1970) that the peculiarities of computer programming make it a privileged tool for learning problem solving with an experimental approach. In fact, children have to impose on themselves rigour and precision in instructing the computer - being explicit and precise is not imposed (incomprehensibly) by enforcement of the teacher, but naturally emerges from the need of being understood by an automated executor with a limited instruction set, which is unable to perform any "human" inference. Briefly: the computer creates an intrinsic motivation to learn by trial, error and debug.

4.4 Transversal competences

Transversal competences like perseverance and tolerance for ambiguity are useful for learning a difficult topic like Informatics, but including it in the definition may cause people to think CT is mainly about these competences.

For example, Corradini et al. (2017a) found that teachers saw the value of Programma il Futuro project (the Italian version of Code.org) more in fostering transversal competences or domain-general skills (like promotion of awareness and comprehension of problem solving, logical thinking, creativity, attention, planning ability, motivation for learning, students interest, cooperation) than in teaching Informatics 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, logic, but often forgetting to refer in any way to an *information-processing agent*, giving a very partial view of Informatics. Moreover, many of them mentioned transversal competences, which hints the view of Informatics as an instrumental discipline, not valuable in itself. This is possibly deriving from attempts to convince teachers of the importance of CT by focusing mainly on its value for other disciplines and as a general learning tool.

Moreover, non-specialists may get the wrong direction of the implication: while researchers agree that competences like perseverance, dealing with complexity, and collaboration are essential to succeed in Informatics, which is a challenging subject (Murphy & Thomas, 2008), the opposite implication (CT fostered by these skills) is far from being proved.

For example, Lewis (2017, p. 18) states that "*programming has been speculated to be uniquely qualified to help normalize failure and thus encourage productive learning strategies*". However, research in education tells us that transfer is difficult and unlikely to happen, especially between knowledge domains far from one another, and especially when treating domain-general skills (Guzdial, 2015; Lewis, 2017).

At the moment, there is no proof that transversal skills like perseverance are automatically fostered by learning CT (for examples, we found no difference in students' mindset, with respect of studying or not studying Informatics at school (Lodi, 2019)).

5. Conclusions

The expression “computational thinking” has become a buzzword related to the introduction of CS in K-12 education. Although it had already been used in the 80s by Papert, it started to be massively used in Informatics education after being re-proposed by Wing.

Many authors tried to define CT: despite being quite different, the most famous definitions share many characteristics. All agree CT is a form of thinking for solving problems by expressing the solution in a way that can be automatically carried out by an (external) processing agent. We identified four categories of CT constitutive elements proposed by authors: mental processes, methods, practices, and transversal skills. We argued that this classification could be useful to frame misconceptions about CT.

Mental processes (e.g., problem solving, problem decomposition, abstraction, logical thinking) and transversal competences (e.g., tolerance for ambiguity, perseverance) resonate with the current narrative on the importance of the 21st-century skills, and are probably even one of the reasons of the widespread of CT in education. This is confirmed by large scale qualitative studies (Corradini et al., 2017a), showing that generalist teachers mainly find value in introducing CT in schools for promoting general skills rather than Informatics core concepts.

However, educational research warns about the transferability of this kind of general skills between disciplines, and some even doubt their teachability.

Moreover, putting too much focus on this aspects risks to dilute the fundamental concepts that distinguish CS from other disciplines (e.g., the presence of a precise external executor that solves problems following provided algorithms, the possibility to describe and execute abstractions through specific languages, the possibility to simulate worlds, and so on). The definitions of CT contain many elements directly linked to CS methods (e.g., automation, data analysis, evaluation) and programming practices (e.g., iterating, debugging). However, educators may fail to include references to these CS specific aspects in their definition of CT.

Since even more specific concepts appear to be shared with other disciplines, the message that teaching separately some of these concepts (often in an informatics-unrelated way) - or simply recognising them inside other disciplines (like math for problem solving, geography for giving precise directions, and so on) - will automatically foster CT is spreading between educators.

We therefore argued that all the characteristics, especially the “core” ones, should be read, understood, and taught *inside* the discipline of Informatics. A lot of “thinking” are worth being taught, CT “*is often a welcome addition to other fields, but not a replacement for their ways of thinking and not a meta-skill for all fields*” (Denning & Tedre, 2019, p. 213).

CT should represent the “disciplinary way of thinking” of Informatics: *Informatical thinking*.

Acknowledgements

I would like to thank Prof. Simone Martini for supporting my research.

References

Aho, A. V. (2011). Ubiquity Symposium: Computation and Computational Thinking. Ubiquity, 2011(January). https://doi.org/10.1145/1922681.1922682

Bell, T., & Vahrenhold, J. (2018). CS Unplugged—How Is It Used, and Does It Work? In H.-J. Böckenhauer, D. Komm, & W. Unger (Eds.), Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday (pp. 497–521). Springer International Publishing. https://doi.org/10.1007/978-3-319-98355-4_29

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking (Using artifact-based interviews to study the development of computational thinking in interactive media design). Proceedings of the 2012 Annual Meeting of the American Educational Research Association. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Cansu, S. K., & Cansu, F. K. (2019). An Overview of Computational Thinking. International Journal of Computer Science Education in Schools, 3(1). https://eric.ed.gov/?id=EJ1214682

Chick, N. L., Haynie, A., Gurung, R. A., & Regan, A. (2009). From generic to signature pedagogies: Teaching disciplinary understandings. In R. A. R. Gurung, N. L. Chick, & A. Haynie (Eds.), Exploring signature pedagogies: Approaches to disciplinary habits of mind. (pp. 1–16). Stylus publishing. https://books.google.co.il/books?id=0SWec-nwL4EC

College Board. (2017). AP Computer Science Principles. College Board. https://apcentral.collegeboard.org/pdf/ap-computer-science-principles-course-and-exam-description.pdf

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”. Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, 224–229. 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. Proceedings of the 2017 ACM Conference on International Computing Education Research, 136–144. 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. Computing at School. https://eprints.soton.ac.uk/424545/

Denning, P. J. (2017). Remaining Trouble Spots with Computational Thinking. Commun. ACM, 60(6), 33–39. https://doi.org/10.1145/2998438

Denning, P. J., & Tedre, M. (2019). Computational Thinking. MIT Press.

Denning, P. J., Tedre, M., & Yongpradit, P. (2017). Misconceptions about Computer Science. Commun. ACM, 60(3), 31–33. https://doi.org/10.1145/3041047

diSessa, A. A. (2018). Computational Literacy and "The Big Picture" Concerning Computers in Mathematics Education. Mathematical Thinking and Learning, 20(1), 3–31. https://doi.org/10.1080/10986065.2018.1403544

Duncan, C. (2019). Computer science and computational thinking in primary schools. [PhD Thesis, University of Canterbury]. http://hdl.handle.net/10092/17160

English, L. D. (1997). Mathematical Reasoning: Analogies, Metaphors, and Images. L. Erlbaum Associates.

Feaster, Y., Segars, L., Wahba, S. K., & Hallstrom, J. O. (2011). Teaching CS Unplugged in the High School (with Limited Success). Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, 248–252. https://doi.org/10.1145/1999747.1999817

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. https://doi.org/10.1145/965754.965757

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/

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

Gurung, R. A. R., Chick, N. L., & Haynie, A. (Eds.). (2009). Exploring Signature Pedagogies: Approaches to Teaching Disciplinary Habits of Mind. Stylus Publishing, LLC.

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. (2019, April). A new definition of Computational Thinking: It's the Friction that we want to Minimize unless it's Generative,. Computing Education Research Blog. https://computinged.wordpress.com/2019/04/29/what-is-computational-thinking-its-the-friction-that-we-want-to-minimize/

Hemmendinger, D. (2010). A Plea for Modesty. ACM Inroads, 1(2), 4–7. https://doi.org/10.1145/1805724.1805725

ISTE, & CSTA. (2011a). Computational Thinking teacher resources. https://id.iste.org/docs/ct-documents/ct-teacher-resources_2ed-pdf.pdf?sfvrsn=2

ISTE, & CSTA. (2011b). Operational Definition of Computational Thinking for K-12 Education. https://id.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf?sfvrsn=2

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

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.

Krauss, J., & Prottsman, K. (2016). Computational Thinking and Coding for Every Student: The Teacher's Getting-Started Guide. Corwin Press.

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

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 (pp. 15–34). Springer International Publishing. https://doi.org/10.1007/978-3-319-54226-3_2

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). Introducing Computational Thinking in K-12 Education: Historical, Epistemological, Pedagogical, Cognitive, and Affective Aspects [PhD Thesis, Alma Mater Studiorum - Università di Bologna]. http://amsdottorato.unibo.it/9188/

Lodi, M. (2019). Does Studying CS Automatically Foster a Growth Mindset? Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, 147–153. https://doi.org/10.1145/3304221.3319750

Lodi, M., Martini, S., & Nardelli, E. (2017). Do we really need computational thinking? Mondo Digitale, 72. http://mondodigitale.aicanet.net/2017-5/articoli/MD72_02_abbiamo_davvero_bisogno_del_pensiero_computazionale.pdf

Martini, S. (2012). Lingua Universalis. Annali della Pubblica Istruzione, 4–5, 65–70.

Martins-Pacheco, L., von Wangenheim, C., & Alves, N. (2020). Polemics about Computational Thinking: Digital Competence in Digital Zeitgeist – Continued Search for Answers: Proceedings of the 12th International Conference on Computer Supported Education, 499–506. https://doi.org/10.5220/0009797104990506

Murphy, L., & Thomas, L. (2008). Dangers of a Fixed Mindset: Implications of Self-Theories Research for Computer Science Education. SIGCSE Bull., 40(3), 271–275. https://doi.org/10.1145/1597849.1384344

Pace, D., & Middendorf, J. (2004). Decoding the Disciplines: Helping Students Learn Disciplinary Ways of Thinking: New Directions for Teaching and Learning, Number 98. Wiley.

Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, Inc.

Schoenfeld, A. H. (1985). Mathematical Problem Solving. Elsevier. https://doi.org/10.1016/c2013-0-05012-8

Selby, C., & Woollard, J. (2013). Computational thinking: The developing definition [Project Report]. University of Southampton (E-prints). https://eprints.soton.ac.uk/356481/

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

Stephens, M., & Kadijevich, D. M. (2020). Computational/Algorithmic Thinking. In S. Lerman (Ed.), Encyclopedia of Mathematics Education (pp. 117–123). Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0_100044

Sternberg, R. J., & Ben-Zeev, T. (Eds.). (1996). The Nature of Mathematical Thinking. Routledge.

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS Unplugged and Middle-School Students' Views, Attitudes, and Intentions Regarding CS. ACM Trans. Comput. Educ., 12(2). https://doi.org/10.1145/2160547.2160551

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

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

Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society 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, 20–23.

M. Lodi is B.S., M.S., and Ph.D. in Computer Science. He is currently Post-doc research fellow and Adjunct Professor of Computer Science Education at Alma Mater Studiorum - Università di Bologna, Italy. His research interest is in Computer Science Education - especially on computational thinking with a constructivist and constructionist approach, teacher training, transfer of learning, computer science mindset, and epistemological aspects of Computer Science as a discipline. He is author of more than ten publications in conferences and journals on Computer Science Education, and a book in Italian for primary school teachers. He is actively involved in nation-wide initiatives to introduce CS in Italian K-12 curriculum. https://lodi.ml