Computational Thinking on the Way to a Cultural Technique

Andreas Bollin, Peter Micheuz

Computational Thinking on the Way to a Cultural Technique

A Debate on Lords and Servants

Andreas Bollin, Peter Micheuz
Universität Klagenfurt, Austria, {Andreas.Bollin | Peter.Micheuz}@aau.at

Abstract. Based on a thorough literature review and on personal expertise in different areas of computer science (education) fields, we reflect and debate on computational thinking from different perspectives. One is that of an Austrian teacher who is confronted with a curriculum for a new subject called 'Basic Digital Education', with computational thinking as an explicit part of it. The other view is that from a reflective software engineer with a holistic perspective on computational thinking and concrete ideas about its limitations. The debate concludes with an agreement on computational thinking as a cultural technique and a mutual approach to a refined working definition.

Keywords. Computational thinking, computer science, life-long-learning, engineering, curriculum development

1 Introduction

In an interview with the German "Süddeutsche Zeitung" at the end of January 2018, Armin Grunwald, head of the Office of Technology Assessment at the German Bundestag, said that if "*we just have to work to run after the technology, then something is wrong. Hegel has already put this in a nutshell, with the relationship of master and servant… The more the Lord relies on his servant, the more dependent he becomes on him*" [1]. Indeed, for decades, educators (and politicians) have had to deal with the question of what to teach and how to be able to produce (or perhaps guarantee) politically mature and technologically up-to-date people. But, what does this mean in the context of current developments: robots, drones, artificial intelligence, smart devices and, not to forget, the Internet of things and smart homes? Is digital education following a scattergun approach?

With the publication of a CACM viewpoint article about computational thinking (CT) in 2006, Jeanette Wing popularised the idea of a new fundamental skill used by everyone in the world by the middle of the 21st century [2]. She defined computational thinking as "*the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer – human or machine – can effectively carry out*". In the Gödel Lecture at Vienna University of Technology on June 9 2016, she also shared many examples of where to find computational thinking aspects in different disciplines, be it economics, law, healthcare or geosciences. So, together with introducing computer science to our classes, do we have a silver-bullet for dealing with today's challenges?

In her 2006 seminal paper, Wing did not primarily think of computational thinking in primary and secondary education [2]. It was not foreseeable which worldwide avalanche has been set off by her, especially among educationalists and teachers. Maybe it goes too far to refer to it as a hype. But if not, it could be a worthwhile endeavour to show that Gartner's Hype Cycle can even be applied to this phenomenon, with the peak of inflated expectations being apparently behind us. Currently, we find ourselves at the slope of enlightenment, in the form of reasonable and viable definitions of that all-in-all still fuzzy term. Reviews on existing literature strengthen some core concepts of CT: logical and algorithmic thinking, decomposition, generalisation and pattern recognition, modelling and abstraction [13].

CT is seen as a fundamental set of mental skills used by everybody, as fundamental as reading, writing and arithmetic [12]. Martin describes CT simply in a few words: "It is about connecting computing to the world" [9]. Moreover, it seems to be widely accepted that coding is an indispensable part of CT [10]. It is seen as an aid to learning software development [11], and is thus coupled with software engineering [20, 21].

The list of publications in the context of CT is amazing. But, even more surprising is the fact that the debate about "How much of computer science and computational thinking should be taught?" is camouflaged by the support from technology enthusiasts, industry, and politicians. So, though there are numerous different curricula and definitions of computational thinking around [3 - 7], there still is no common understanding about how far we need to go. This paper, therefore, tries to answer the question of what computational thinking (also at primary and secondary schools) is, and elaborating on it closer, what it *is not*. We try to approach the border between computational thinking (as a set of skills that is needed due to contemporary demands, addressing the characteristics of a new cultural technique) and the skills of an engineering education that are needed by professionals.

The idea behind this paper was born during a trip of the two authors from Klagenfurt to Vienna, where we were trying to define computational thinking in the Austrian context. Both authors have years of teaching experience in computer science, but the first author has a strong engineering background, whereas the second author is involved in political discussions, adapting the school system in Austria for many years. It seems natural to us to approach this topic in the form of a debate, where statements are presented and redefined along the discussion, finally coming up with a reliable definition of the border between CT and not-CT.

2 A debate on CT in the (Austrian) educational system

"Basic Digital Education" is the name for a new subject which will be introduced in all Austrian lower secondary schools beginning in the school year 2018-2019. There is one curriculum covering four years of lower secondary education (age groups from 10-14 years) and encompassing eight main topics:

- Social aspects of media change and digitisation
- Information, data and media competence
- Operating systems and standard applications
- Media design

- Digital communication and social media
- Security
- Technical problem-solving
- Computational thinking

Obviously, these topics stand for a broad curriculum, which encompasses digital competence, media competence, and political competence as well. Digital competence in particular is expected to empower pupils, based on a comprehensive overview of digital tools (hardware and software), for coping with certain scenarios in educational, vocational and private contexts in a reflective manner.

At first sight, informatics (computer science) does not play a prevalent and visible role. Media pedagogy and digital literacy are apparently better represented than core informatics. At a second glance, computer science is represented explicitly as CT. This term has not been translated into the German synonym "Informatisches Denken" and appears in the curriculum as the (global) driving force for implementing elements of core informatics into a seemingly overcrowded curriculum.

All topics of the curriculum are divided into further subtopics and detailed competence descriptions and learning goals. The main topic CT is split into a basic and advanced level (see Table 1 for more details).

Computational Thinking	Basic Level (2 hours per week)	Advanced Level 1 (+ 1 hour)	Advanced Level 2 (+ 1 hour)
Working with Algorithms	Pupils - name and describe everyday processes - use, build and reflect codes (e.g. secret writing, QR-Code) - reproduce distinct instructions (algorithms) and carry them out - formulate distinct instructions verbally and in written form	Pupils - discover similarities and rules (patterns) within instructions (algorithms) - discover the importance of algorithms in automatic digital processes (e.g. automated proposal of potentially interesting information)	Pupils can evaluate intuitive user interfaces and its underlying processes
Creative Use of Programming Languages	Pupils - produce simple programs or web applications with appropriate tools to solve a problem or to complete tasks - know different programming languages and production processes	Pupils - master basic programming structures (decision, loops, procedures)	Pupils reflect the boundaries and options of simulations

Table 1. CT in the Austrian Curriculum for Basic Digital Education [23]

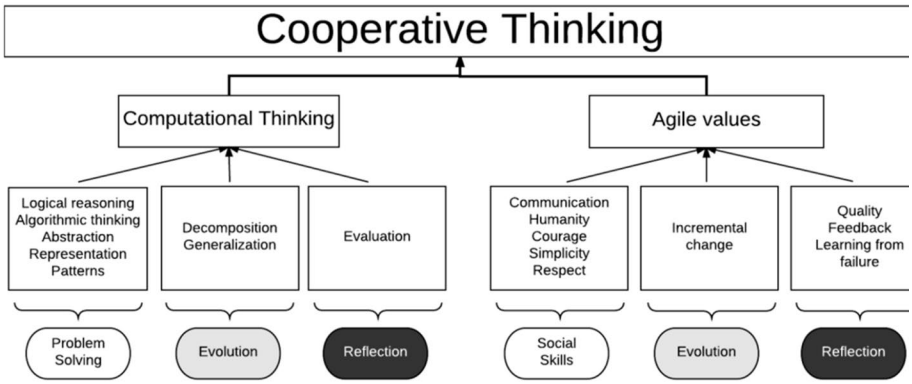

Figure 1. Cooperative Thinking as a combination of CT and Agile Development breakdown (according to Missiroli et al. [14] and following Computing at School [15] and Beck [16])

For the first time in the history of computing education in Austria, all lower secondary schools – "some" teachers and all pupils – are exposed to a binding curriculum wherein computing, algorithms and programming play a more or less clearly defined and specified role. Digital education in general and CT in particular are no longer optional for a special cohort of pupils, but obligatory for all. It would be too much to go into the details of the challenging organisational issues required to implement the curriculum. All there is to say about it is that within the framework of school autonomy, schools can change the number of hours within a certain range. They have to decide autonomously to introduce the curriculum for 'Basic Education' as an independent subject (2 to 4 hours per week, which means an assumed 64 to 128 hours of lessons) within 4 years, in a completely integrative way in other subjects (64 to 128 hours) or all hybrid forms between these extremes. But independent from the justified question of how to cover all topics, with, all-in-all, over one hundred learning goals in a very limited time, the main questions are: to what extent is the Austrian definition of CT in line with international views among experts including scientists and teachers, and does it already answer the question of what CT is definitely not? The explicit topic CT in the Austrian curriculum could lead to the assumption that it has nothing to do with the other topics, like standard applications, technical problem-solving or security or even media design.

2.1 Proper or -mproper

The opening statement of A. Bollin: The article of Bocconi et al. [8] about computational thinking approaches and orientations in K-12 education brings us exactly to the point: there are still different views on what computational thinking is. Some of them include programming; some of them do not. For motivational reasons, curricula sooner or later will introduce coding or programming in their lecture units. According to the new Austrian curriculum, pupils should be able to produce simple programs or web-applications and should know different programming languages. This seems to be in-

line with the current trend, but when it is not introduced for good reasons and correctly, this is putting the cart before the horse.

In the vision of Wing, everyone should be able to use well-established techniques that have been applied by engineers for a long time already. It is about formulating problems in such a way that, maybe with the help of others (or even machines), problems can be solved easier. Thus, it is not surprising that the problem-solving activity includes logical reasoning, algorithmic thinking, abstraction, decomposition, generalisation, pattern detection and languages (notations) for communication and representing information. But, this is quite individualistic, and nowadays by far not enough to solve larger (and more complex) problems. Missiroli et al. [14] thus suggest combining computational thinking skills and team-based skills, as needed by software developers, when developing software in an agile manner. Figure 1 summarises their concept, combining problem solving and social skills to a new literacy they call "cooperative thinking".

So, when introducing programming, it should not be done to just to illustrate how algorithms can be used and executed. We should not be dumb Lords. One should be honest and state that nowadays it is about solving real-world problems. But, this then includes more than just being able to write some lines of (computer-readable) text. It is also about including some software engineering skills.

2.2 Step-by-step

The follow-up statement of P. Micheuz: "The secret of getting ahead is getting started. The secret of getting started is breaking your complex overwhelming tasks into small manageable tasks and starting on the first one." This little-known quote from poet Mark Twain on problem-solving can be seen as a remarkable historical precursor, long before CT began to rack the brains of thousands of educational experts in the field. Every software engineer working on solving so-called real-world problems in teams, and with the aid of digital tools, has learned "simply to go" by practicing and internalising basic concepts and, to a reasonable extent, also some (especially among many educationalists), disreputable rote learning. There is another quote, "He who wants to build high towers must dwell with the fundament for a long time" from the Austrian composer Anton Bruckner. It supports the truism that CT as a cultural technique needs an early beginning, a coherent and sustainable construction of skills and competences in the form of a spiral curriculum.

Looking at the ambitious and overloaded curriculum, with CT as a comparatively small part, and even under the assumption that motivated and CT-proven teachers follow the intended curriculum, it is rather unlikely that all pupils can meet all goals of the whole curriculum in 64 to 128 hours of lessons within four years.

It is self-evident that CT for primary, lower/upper secondary and tertiary level (must) have different characteristic forms. I can live with the fact that advanced aspects of software engineering and bigger projects should play a role, at the earliest, at upper and tertiary level, but for primary and lower secondary level that would go too far. Nonetheless, cooperative thinking, or better, cooperative acting, can be harnessed as a valuable general teaching method, especially in programming and CT-related lessons. I doubt that software engineering in its full definition is an adequate term for lower age

groups. That would overstrain teachers who are already struggling with CT, but willing to undergo professional development in that field. But, often things are not as bad as they seem. From this perspective, the fact that CT lacks a precise definition must be considered predominantly an advantage, as it is scalable and adaptable for various age groups and even interdisciplinary implementations.

The explicit learning goal of producing (simple) programs confronts Austrian teachers and pupils with a fait accompli. CT in the new curriculum covers more than mere algorithmic thinking, but less than dealing with (complex) real-world problems.

2.3 Past and future

The follow-up statement of A. Bollin: Interestingly, history is repeating. The introduction of spreadsheets decades ago led to a situation where every user was using sheets without noticing that he or she was (and is) in fact programming [19] (and not following quality standards a software engineer would naturally follow). Debugging aids for spreadsheets are getting better nowadays, but still a lot of erroneous spreadsheets are around, forming the basis for (private as well as industrial) disastrous decisions. We should not educate pupils in a way where oversimplifications potentially lead to a misuse or misunderstanding of reality.

The "proper or improper" claim does not mean that even more skills are to be packed into the tight schedule in schools under the umbrella of computational thinking, but it is a hint toward a problem that we are running into. In one work, Hermans and Aivaloglou [17] show that using block-oriented languages (which are quite often chosen for novice programmers) dramatically hampers learning programming later at universities. Without taking care of software engineering practices from the beginning (as examined in their paper), a lot of effort, time and resources are needed to produce the engineers that the industry is longing for. Now, not everybody will (and should) strive for a career as a (software) engineer. But, this is not a reason for showing programming in a way nobody ever would and should program. The problem focus and the context are missing, and in our classrooms, we continue having a lot of bored pupils.

This viewpoint is partially also supported by neuro-didactic findings: people have enormous difficulty learning when either parts or wholes are neglected. Hence, the learning brain needs the whole and the details; it requires both a big picture and paying attention to the individual parts [18]. In our case, the real world is needed, and the individual parts could be programming tasks or the meaningful use of technology.

To me, programming already is part of the engineering discipline, and CT covers parts of the skills an engineer needs to successfully solve problems and to create something new. Sure, for didactic reasons, one might start to introduce small programs to show the application of CT techniques in a bigger context. It is also clear that one has to start step-by-step. But, as another comparison with a cultural technique shows, when we start learning to read, we do not stop after recognising the letters. We continue with combining the letters together and with learning to recognise syllables, words, etc., until we reach some level of proficiency.

To summarise, when defining programming as not being part of CT, then we definitely should add computer science (and software techniques) to our curricula in order to show the whole picture. When we say that programming is part of CT, then we should

not do it in an inappropriate (context-neglecting) way and need to add more hours to our syllabi. It also means investing more resources and training our teachers (including all the lecturers at universities) accordingly. This is the only way that pupils (and parents and teachers) will responsibly know why to decide either for or against a technical study or technique-related working place later on. It is also the only way to keep technological change being the servant and not allow it to take over.

2.4 In the right place at the right time with a sense of proportion – P. Micheuz

The final statement of P. Micheuz: Digital education in its full complexity will remain a big challenge in traditional formal educational settings. So will the recent decision from an expert group to embed CT explicitly into the Austrian curriculum. It does not solve the problem of age-appropriateness and does not even guarantee dedicated lessons for computing. It transfers the responsibility for its implementation - to what extent and at which age level - to schools and teachers. In contrast to traditional subjects such as language education and mathematics, the drivers of the main cultural techniques of reading, writing and arithmetic, currently CT cannot rely on sequenced and coherent age appropriate lessons. Accordingly, there is legitimate concern that within 4 years of lower secondary education, CT will not be taught properly. It may be assumed that in the initial phase of executing the curriculum of 'Basic Digital Education' in some schools, CT will play little role.

Regarding the introduction of programming with a block-based approach (Scratch or similar development environments), I am quite optimistic. The question remains when to switch to the first steps of textual coding. As for today, the Austrian curriculum for 'Basic Digital Education' has no answer for that.

It cannot be expected that (m)any teachers at this school level see the whole picture of software development. Basic CT education with first steps in problem-solving, algorithmic thinking and first programming experiences on a small scale should be also feasible without having deep experiences in software engineering.

But even these first steps cannot be taken for granted. CT education for pupils needs CT-educated teachers and professional development in that field on a large scale. In the next years, nationwide measures in the form of pre-service and in-service training in various formats will need to be taken.

3 A working definition of computational thinking

In the previous section, we tried to take two positions, one from the viewpoint of a software engineer, and one from the viewpoint of teachers who need to make the next generations fit for exciting technological changes and current threats. Now, we try to converge and to find a working definition in the context of the Austrian situation and with regard to CT as a cultural technique.

3.1 About cultural techniques

When reflecting on cultural techniques, we think primarily of the cognitively most fundamental cultural techniques of reading, writing and arithmetic, and of its lengthy and laborious acquisition in dedicated school subjects. But, in our increasingly digitally penetrated culture, there are demands for extended skills and competences as widely elaborated in the Digital Competence Framework for Citizens [22].

If we accept CT as a new cultural technique, it might be helpful to look closer at what a cultural technique really is. Cultural techniques are a set of skills, concepts and competences that help human beings "function properly" in a given culture. They help in dealing with tasks and solving problems in different situations of life, like making a fire, using a calendar, or being able to communicate in social networks.

As cultural techniques are solution concepts for tasks and problems of human beings, we have to be clear about current human needs (cf. Maslow's hierarchy). In the context of new technological demands and secondary education, this list of needs encompasses being able to:

- Communicate with others using state-of-the-art communication technology.
- Search for, assess and work with available information.
- Solve tasks in a sustainable manner with the help of new state-of-the-art technology.
- Protect him- or herself against fraud.

Apart from these needs, which are well covered also in the new Austrian curriculum for 'Basic Digital Education' (see Section 2), one also needs to know about the limits of the solution concepts and being able to protect oneself from a misuse, so the list has to be extended by:

- Knowing what computer science and software engineering is about.
- Being aware of potential limitations and side-effects.

Last, but not least, computational thinking definitely does not include the cultural technique of typewriting or information technology (IT)-literacy such as using office software and digital devices at a cursory level. And, it is rather unthinkable that there are computational thinkers who are not fluent in harnessing computers, but it is quite possible that fluent computer users are not yet educated computational thinkers.

With all these reflections in mind, one can give a quite crisp definition of what computational thinking in secondary education is and what it is not.

3.2 A working definition

Computational thinking **is a cultural technique** consisting of a set of skills needed to complete a task in a responsible, sustainable manner including problem-solving, evolutionary and reflection steps. These steps encompass *logical reasoning, algorithmic thinking, abstraction, generalisation, decomposition, design/solution patterns, evaluation techniques*, and as computers might be involved in the solution process, *different representation forms*. It also includes knowing about related disciplines like computer science and software engineering. As such, it should be thought about to its fullest extent, but in an age-appropriate manner, at the secondary level in Austria.

Computational thinking **is not about** being able to work like a *software engineer or computer scientist*. It is not necessarily about *finalising (software) products in a correct, efficient and cost-oriented manner*. It is also not about *proving and searching for algorithmic properties or creating new physical devices*. But, it is knowing about the limits of one's own solution ideas.

Now, as our debate was also about programming and coding, it is up to the educator to what extent to include programming languages (in graphical or textual form) to motivate for a technique or skill. However, it is then his or her responsibility to make the difference to software engineering clear. The implications for a teacher (and for teacher education) are obvious: he or she needs to know more about computer science and software engineering, as at least a portion of teaching CT is about teaching the differences/boundaries to neighbouring disciplines.

4 Summary and outlook

In this paper, we tried to further the approach to defining computational thinking, reflecting on discussion among scientific and educational experts in the educational field of computing. It stresses the fact that CT is at the border of engineering disciplines, and, when coding is involved, it is also close to the border of software engineering.

More than a decade after the seminal work of Jeanette Wing [2], the wave of its public perception reached Austria. Since computational thinking is an explicit part of the new curriculum 'Basic Digital Education', it will be a starting point of many discussions and debates.

The debate in this paper results in a working definition from the perspective of software engineering and CT as a cultural technique, having in mind the limits and challenges of school education in general and the introduction of a new subject in particular.

We agree that CT in the way we see it must be considered an important cultural technique in the 21st century. But, we are realistic enough to know that CT as imagined in the heads of many educational experts, including our abstract working definition above, still has a long and difficult way to go from the conception stage to its implementation.

There is hope that its future will not be that of the term and subject of informatics in Austrian lower secondary schools where, according to a very broad interpretation of its definition, the subject informatics in Austrian schools created its own reality. Maybe a reality with some CT-related parts were included, but from a vast majority of teachers they were not realised as CT defined above. Only when carefully knowing the borders, were we able to deal with current and future human needs. And thus, it is more likely to keep the role of a Lord not being dependent on his or her (technical) servants.

References

1. Bauchmüller, M., Braun, S.: SZ Online, http://www.sueddeutsche.de/wirtschaft/gefahren-der-digitalisierung-die-leute-merken-nicht-mehr-wie-fragil-das-system-ist-1.3842973-3 (Accessed: 11.1.2019)

2. Wing, J.: Computational Thinking. CACM Viewpoint. 33-35 (2006)
3. Webb, M., Davis, E.N., Bell, T., Katz, Y.J., Reynolds, N., Chambers, D.P., Sysło M.M.: Computer science in K-12 school curricula of the 2lst century: Why, what and when? Education and Information Technologies. 22(2), 445-468 (2017)
4. Gallenbacher, J.: Abenteuer Informatik. Hands-on exhibits for learning about computational thinking. Paper presented at WiPCSE'12. Germany (2012)
5. Dierbach, C., Hochheiser, H., Collins, S., Jerome, G., Ariza, C., Kellevher, T., Kelinsasser, W., Dehlinger, J., Siddharth, K.: A model for piloting pathways for computational thinking in general education. SIGCSE '11. New York, NY, ACM. 257-262 (2011)
6. Seiter, L., Foreman, B., Carroll J.: Modeling the Learning Progressions of Computational Thinking of Primary Grade Students. 9th international ACM Conference on International Computing Education Research. New York, NY. 59-66 (2013)
7. Cole, E.: On Pre-requisite Skills for Universal Computational Thinking Education. In Proceedings of ICER'15. Omaha, NE, 253-254 (2015)
8. Bocconi, S., Ferrari, A., Kampylis, P.: Developing Computational Thinking: Approaches and Orientations in K-12 Education, EdMedia 2016. Vancouver, BC. 13-18 (2016)
9. Martin, F.: Rethinking Computational Thinking: http://advocate.csteachers.org/2018/02/17 /rethinking-computational-thinking (Accessed: 11.1.2019)
10. Prottsman, K.; Krauss, J.: Computational Thinking and Coding for Every Student: The Teacher's Getting-Started Guide. SAGE Publications (2017)
11. Beecher, K.: Computational Thinking. A beginner's guide to problem-solving and programming. BCS Learning & Development Ltd. UK (2017)
12. Wing, J.: Computational Thinking benefits Society. Social Issues in Society. http://socialissues.cs.toronto.edu (Accessed: 11.1.2019)
13. Selby, C., Woollard, J.: Computational Thinking: the developing definition (2013). https://eprints.soton.ac.uk/356481. University of Southampton (Accessed: 11.1.2019)
14. Missiroli, M., Russo, D., Ciancarini, P.: Cooperative Thinking, or: Computational Thinking meets Agile. Proceedings of the 30th IEEE Conference on Software Engineering Education and Training. Savannah, GA. 187-191 (2017)
15. Beck, K., Andres C.: Extreme programming explained. Addison-Wesley (2004)
16. Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Selby, T., Ng, C., Woollard, J.: Computational thinking: A guide for teachers. Computing at Schools E-Book (2015)
17. Hermans, F., Aivaloglou, E.: Do code smells hamper novice programming: A controlled experiment on Scratch Programs. In Proceedings of the 24th IEEE International Conference on Program Comprehension. Austin, TX. 1-10 (2016)
18. Caine, R.N., Caine, G.: Understanding a brain-based approach to learning and teaching. Educational Leadership. 48(2), 66–70 (1990)
19. Mittermeir, R., Clermont, M., Hodnigg, K.: Protecting Spreadsheets against Fraud. Proceedings of the European Spreadsheet Risks International Group (2005)
20. Bollin, A., Sabitzer, B.: Teaching Software Engineering in Schools – On the right time to introduce Software Engineering Concepts. 6th IEEE Global Engineering Education Conference, EDUCON. 511-518 (2015)
21. Bollin, A., Pasterk, S., Antonitsch, P., Sabitzer, B.: Software Engineering in Primary and Secondary Schools – Informatics Education is more than Programming. IEEE 20th Conference on Software Engineering Education and Training, CSEE&T. 132-136 (2016)
22. Digital Competence Framework for Citizens: https://ec.europa.eu/jrc/en/digcomp (Accessed: 11.1.2019)
23. Digital Basic Education (in German): https://bildung.bmbwf.gv.at/schulen/schule40/dgb/index.html (Accessed: 11.1.2019)