Reclaiming the Roots of CT

Irene Lee

There has been much debate about the definition of computational thinking (CT) and the relative merits of different definitions. In this article, I argue for a focused definition of CT that clearly distinguishes it from other forms of thinking.

CT was popularized by Jeannette Wing in 2006 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." Lee and Martin, CSTA CT Task Force co-chairs, further simplified this definition in 2015 to "CT refers to the human ability to formulate problems so that their solutions can be represented as computational steps or algorithms to be carried out by a computer." For the remainder of this article this will be referred to as the "thought processes" definition of CT.

The Computing at School (CAS) curriculum supports this description of CT. It explicitly states, "the thinking that is undertaken before starting work on a computer is known as CT" (*barefootcas.org.uk*). Further, in UK's Computing at School's "Computational thinking: A guide for teachers," CT is clearly described as "a thought process, not the production of artefacts or evidence" (*community.computingatschool.org.uk/files/6695/original.pdf*).

Abstraction, automation, and analysis, the three pillars of CT as described by Cuny, Snyder, and Wing in 2011, have been observed in students as young as middle school. Students have demonstrated that they can develop abstractions and automations as they study and solve real-world problems in modeling and simulation projects and robotics projects. Key to this definition is that CT takes place when students are "looking at a real-world problem in a way that a computer can be instructed to solve it." In the context of modeling and simulation, students were actively engaged in CT when they selected features of the real world to incorporate into their models (abstraction), determined which elements of the model need to be updated as simulation time advanced (automation), and analyzed the model's inclusion the features necessary to mimic the real world (analysis).

One of the merits of the "thought processes" definition is that it is very specific to humans harnessing computers as information processing devices. This specificity makes it different from critical thinking, mathematical thinking, and scientific thinking. A difficulty with this definition is that a teacher or student new to computer science (CS) may not be able to relate to this definition of the practice. How does one formulate a problem and its solution so that it can be carried out by a computer if one does not know what a computer is capable of doing and how to give a computer instructions? What differentiates a poor formulation from a strong one?

Thus, after Wing's definition was publicized, other groups published their own interpretations of CT, including the ISTE/CSTA Operational Definition, "Computational Thinking Practices" (*AP CS Principles* and *Exploring CS*, 2012), "CT concepts, practices, perspectives" (Brennan & Resnick, 2012), "CT Patterns" (Repenning, 2012), and "CT" (*Exploring CT*, Google, 2014).

Common among these definitions is an expansion of CT to include many other practices. For example, the ISTE/CSTA "operational definition of CT" was constructed to aid teachers in seeing themselves as already teaching skills that are components of CT.

While well-intentioned, the ISTE /CSTA operational definition has caused confusion. It is not well understood that an "operational definition" is intended to be a definition of the operations that make up a practice. Each operation is a part of the larger practice but does not by itself equal the practice. Thus the operational definition of CT describes various operations that make up CT but conducting a single operation does not equal "doing CT."

Unfortunately, all too often, the interpretation of the operational definition is that if you are doing any one of the listed operations, you are a computational thinker. This is not correct. For example, logically organizing and analyzing data is an operation described in the operational definition of CT. But a student who organizes data, without consideration of how a computer program would direct a computer to read in, store, and manipulate the data, is not doing CT.

In a similar vein, the AP CS Principles and Exploring CS curricula describe "CT Practices" that extend beyond the original "thought processes" definition of CT. The CT Practices include "communicating computational thought processes" and "collaborating with peers on computing activities." While these are both valuable practices in CS education, they are not necessarily part of "formulating a problem and

It's Election Time

CSTA BOARD OF DIRECTORS CANDIDATES

9 –12 Representative Candidates

Stacey Kizer
Chinma Uche

At-Large Representative Candidates

Myra Deister
Michelle Lagos

International Representative Candidates

Miles Berry
Michael Jones

State Department Representative Candidates

Anthony Owen
Doug Paulson

University Faculty Representative Candidates

Darcy G. Benoit
Fred G. Martin

Read their personal statements on page 10

its solution so that the solution can be carried out by a computer."

The expansion of definitions of "CT" and definitions of "CT Practices" have led to an erosion of the integrity of the "thought processes" definition of CT. Some have come to believe that CT means everything and, consequently, nothing at the same time.

Furthermore, the two terms "computational thinking practices" and "computational thinking" often get conflated (or taken to mean the same thing). In some circles, CT has come to encompass "everything people think kids should learn in CS," including the iterative development of software artifacts.

This losing of the original definition of CT has serious ramifications. 1) We lose what is special about CT—that the human is formulating a problem and its solution so that the solution can be carried out by a computer (not by a human); 2) CT can be viewed as any task that involves students thinking while on a computer—troubleshooting hardware involves thinking and computers, is it computational thinking? and 3) We lose sight of the power of CT to study and solve real-world problems. If a student is doing CT by writing and debugging some code, why go further and address real-world problems?

I believe that CT is a skill that is developed through repeated exposure to how real-world problems are represented, studied, and solved using computers as information processing devices, and progressively deeper understanding of what computers are able to do and how to instruct them. Students can develop CT skills through opportunities to map real-world problems into abstractions and algorithms that can be represented and operated upon on a computer.

ISTE/CSTA "Operational Definition of Computational Thinking for K-12 Education" (2011)

1. **Formulating problems in a way that enables us to use a computer and other tools to help solve them**
2. **Logically organizing and analyzing data**
3. **Representing data through abstractions such a models and simulations**
4. **Automating solutions through algorithmic thinking (a series of ordered steps)**
5. **Indentifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources**
6. **Generalizing and transferring this problem-solving process to a wide variety of problems**

CT Driving Computing Curriculum in England

John Woollard

Computational thinking (CT) has come to the fore for many teachers in England with the advent of the new *National curriculum in England: computing programmes of study* in September 2013 (*goo.gl/SklB9O*). It is explicitly and thoroughly embedded in the curriculum for K–12. The first sentence states, "A high quality computing education equips pupils to use computational thinking and creativity to understand and change the world."

CT lies at the heart of the computing curriculum but it also supports learning and thinking in other areas of the curriculum. CT gives a new paradigm for thinking about and understanding the world more generally. Simon Peyton-Jones, chair of Computing At School (CAS), succinctly explains why learning computer science (CS) and CT are core life skills, as well as being eminently transferable, in a talk filmed at TEDx-Exeter (*bit.ly/13pJLCR*).

CT skills are the set of mental skills that convert "complex, messy, partially defined, real-world problems into a form that a mindless computer can tackle without further assistance from a human," the Chartered Institute for IT (*bit.ly/1Li8mdn*).

In the UK, the term CT has been described in different ways for different audiences but there is a growing consensus that CT is a cognitive or thought process involving logical reasoning by which problems are solved and artifacts, procedures, and systems are better understood. It embraces:

- the ability to think algorithmically;
- the ability to think in terms of decomposition;
- the ability to think in generalizations, identifying and making use of patterns;
- the ability to think in abstractions, choosing good representations; and
- the ability to think in terms of evaluation.

CT skills enable pupils to access parts of the computing subject content. Importantly, they relate to thinking skills and problem solving across the whole curriculum and through life in general.

Where these thinking skills are being promoted we see the pupils adopting approaches to problem solving such as tinkering, creating, debugging, persevering, and collaborating. These are key features associated with successful learning in computing and across the curriculum. Computing, computer programming in particular, enables tinkering to occur. Learners are genuinely learning through trial and improvement. We all know that perseverance is necessary when debugging programs and we appreciate the reward and feeling of satisfaction when creating and collaborating.

A number of techniques can be employed to enhance CT. Think of these as "computational doing," the computing equivalent of "scientific methods." They are the tools by which CT is operationalized in the classroom, workplace, and home: reflecting, coding, designing, analyzing, and applying. These techniques enable CT skills to be developed.

Reflection is the skill of making judgements (evaluations) that are fair and honest in complex situations that are not value-free. Within CS this evaluation is based on criteria used to specify the product, heuristics (or rules of thumb), and user needs to guide the judgements.

An essential element of the development of any computer system is translating the design into code form and evaluating it to ensure that it functions correctly under all anticipated conditions. Debugging is the systematic application of analysis and evaluation using skills such as testing, tracing, and logical thinking to predict and verify outcomes.

Designing involves working out the structure, appearance, and functionality of artifacts. It involves creating representations of the design, including human readable representations such as flowcharts, storyboards, pseudo-code, systems diagrams, etc. It involves activities of decomposition, abstraction, and algorithm design.

Analyzing involves breaking down into component parts (decomposition), reducing the unnecessary complexity (abstraction), identifying the processes (algorithms), and seeking commonalities or patterns (generalization). It involves using logical thinking, both to better understand things and to evaluate them as fit for purpose.

Applying is the adoption of pre-existing solutions to meet the requirements of another context. It is generalization—the identification of patterns, similarities and connections—and exploiting those features of the structure or function of artifacts. An example includes the development of a subprogram or algorithm in one context that can be re-used in a different context.

Computing At School, as a grass-roots and free teacher-membership organization, has been at the forefront of advising on the changes to the curriculum and in providing much needed support to both primary and secondary teachers faced with the challenge of bringing into being a new subject in UK schools.

LEARN MORE:
CT: A guide for teachers: *www.computingatschool.org.uk/news_items/26*
CAS Barefoot: CPD for K–5 teachers: *barefootcas.org.uk*
CAS Tenderfoot: CPD for 6–12 teachers: *www.computingatschool.org.uk/tenderfoot*
CAS Community: *community.computingatschool.org.uk*
CAS Resources for computational thinking: *community.computingatschool.org.uk/resources/2324*
CAS Network of Excellence: *community.computingatschool.org.uk/resources/802*

Meet the Authors

Lissa Clayborn
Deputy Executive Director/Chief Operations Officer, CSTA
Lissa has worked for over 20 years in the nonprofit educational technology sector, including ISTE.

Daryl Detrick
Warren Hills HS, NJ
Daryl is a CS educator and past President of CSTA Central NJ and is Co-chair of the CSNJ.

J. Philip East
University of Northern Iowa
Philip has been teaching computing for over 35 years. He is the Program Chair for the CSTA Annual Conference.

Shuchi Grover
SRI International
Shuchi is a research scientist at SRI's Center for Technology in Learning and a member of the CSTA CT Task Force.

Stephanie Hoeppner
Williamsburg Local Schools, OH
Stephanie has taught CS for 16 years. She has served on the CSTA Board of Directors and is the Workshop Chair for the CSTA Annual Conference.

Joe Kmoch
CSTA Wisconsin
Joe is a retired CS teacher now working as a CS consultant. He is a member of the CSTA CT Task Force.

Irene Lee
MIT
Irene is a research scientist in the Scheller Teacher Education Program and Education Arcade. She serves as a Co-chair of the CSTA CT Task Force.

Fred G. Martin
University of Massachusetts Lowell
Fred is a CS professor. He serves as a Co-chair of the CSTA CT Task Force.

John Woollard
CAS
John is a leading member of Computing At School in the UK. He serves as the Chair of the Assessment working group and Coordinator of the Tenderfoot project.