Computational thinking's influence on research and education for all

Influenza del pensiero computazionale nella ricerca e nell'educazione per tutti

Jeannette M. Wing

Columbia University, New York, N.Y., USA, wing@columbia.edu

HOW TO CITE Wing, J.M. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology*, *25*(2), 7-14. doi: 10.17471/2499-4324/922

ABSTRACT Computer science has produced, at an astonishing and breathtaking pace, amazing technology that has transformed our lives with profound economic and societal impact. In the course of the past ten years, we have come to realize that computer science offers not just useful software and hardware artifacts, but also an intellectual framework for thinking, what I call "computational thinking". Everyone can benefit from thinking computationally. My grand vision is that computational thinking will be a fundamental skill—just like reading, writing, and arithmetic—used by everyone by the middle of the 21st Century.

KEYWORDS Computational Thinking, Education, Curriculum.

SOMMARIO L'informatica ha prodotto, a un ritmo sorprendente e mozzafiato, una straordinaria tecnologia che ha profondamente trasformato le nostre vite con importanti conseguenze economiche e sociali. Negli ultimi dieci anni ci si è resi conto che l'informatica non solo offre utili artefatti software e hardware, ma anche un paradigma di ragionamento che chiamo "pensiero computazionale". Ognuno di noi può trarre vantaggio dal pensare in modo computazionale. Mi aspetto che il pensiero computazionale costituirà un'abilità fondamentale - come leggere, scrivere e far di conto – che verrà utilizzata da tutti entro la prima metà del XXI secolo.

PAROLE CHIAVE Pensiero Computazionale, Educazione, Curriculum.

1. WHAT IS COMPUTATIONAL THINKING?

This article describes how pervasive computational thinking has become in research and education. Researchers and professionals in an increasing number of fields beyond computer science have been reaping benefits from computational thinking. Educators in colleges and universities have changed undergraduate curricula to promote computational thinking to all students, not just computer science majors. Before elaborating on this progress toward my vision, let's begin with describing what is meant by computational thinking.

1.1. *Definition*

I use the term "computational thinking" as shorthand for "thinking like a computer scientist" (Wing, 2006). To be more descriptive, however, I now define computational thinking (with input from Al Aho at Columbia University, Jan Cuny at the National Science Foundation, and Larry Snyder at the University of Washington) as follows:

> *Computational thinking is 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.*

Informally, computational thinking describes the mental activity in formulating a problem to admit a computational solution. The solution can be carried out by a human or machine. This latter point is important. First, humans compute. Second, people can learn computational thinking without a machine. Third, "computers" of today combine the intelligence of humans and the intelligence of machines. Also, computational thinking is not just about problem solving, but also about problem formulation.

In this definition I deliberately use technical terms. By "expressing" I mean creating a linguistic representation for the purpose of communicating a solution to others, people or machines. The expressiveness of a language, e.g., programming language, can often make the difference between an elegant or inelegant solution, e.g., between a program provably absent of certain classes of bugs or not. By "effective," in the context of the Turing machine model of computation, I mean "computable" (or "decidable" or "recursive"); however, it is open research to revisit models of computation, and thus the meaning of "effective," when we consider what is computable by say biological or quantum computers (Wing, 2008) or what is solvable by humans (Levin, 2013; Wing, 2008).

1.2. *Abstraction is Key*

Computer science is the automation of abstractions (Aho & Ullman, 1992)[1]. So, the most important and high-level thought process in computational thinking is the abstraction process. Abstraction is used in defining patterns, generalizing from specific instances, and parameterization. It is used to let one object stand for many. It is used to capture essential properties common to a set of objects while hiding irrelevant distinctions among them. For example, an algorithm is an abstraction of a process that takes inputs, executes a sequence of steps, and produces outputs to satisfy a desired goal. An abstract data type defines an abstract set of values and operations for manipulating those values, hiding the actual representation of the values from the user of the abstract data type. Designing efficient algorithms inherently involves designing abstract data types.

Abstraction gives us the power to scale and deal with complexity. Applying abstraction recursively allows us to build larger and larger systems, with the base case (at least for traditional computer science) being bits (0's and 1's). In computing, we routinely build systems in terms of layers of abstraction, allowing us to focus on one layer at a time and on the formal relations (e.g., "uses," "refines" or "implements," "simulates") between adjacent layers. When we write a program in a high-level language, we are building on lower layers of abstractions. We do not worry about the details of the underlying hardware, the operating system, the file system, or the network; furthermore, we rely on the compiler to correctly implement the semantics of the language. The narrow-waist architecture of the Internet demonstrates the effectiveness and robustness of appropriately designed abstractions: the simple TCP/IP layer at the middle has enabled a multitude of unforeseen applications to proliferate at layers above, and a multitude of unforeseen platforms, communications media, and devices to proliferate at layers below.

[1] In this book, the authors define Computer Science to be "The Mechanization of Abstraction".

2. COMPUTATIONAL THINKING AND OTHER DISCIPLINES

Computational thinking has already influenced the research agenda of all science and engineering disciplines. Starting decades ago with the use of computational modeling and simulation, through today's use of data mining and machine learning to analyze massive amounts of data, computation is recognized as the third pillar of science, along with theory and experimentation (President's Information Technology Advisory Council, 2005).

Consider just biology. The expedited sequencing of the human genome through the "shotgun algorithm" awakened the interest of the biology community in computational concepts (e.g., algorithms and data structures) and computational approaches (e.g., massive parallelism for high throughput), not just computational artifacts (e.g., computers and networks). In 2005, the Computer Science and Telecommunications Board of the National Research Council (NRC) published a 468-page report laying out a research agenda to explore the interface between biology and computing (National Research Council, 2005). In 2009, the NRC Life Sciences Board's study on Biology in the 21st Century recommends that "within the national New Biology Initiative, priority be given to the development of the information technologies and sciences that will be critical to the success of the New Biology" (National Research Council, 2009). Now at many colleges students can choose to major in computational biology.

The volume and rate at which scientists and engineers are now collecting and producing data—through instruments, experiments, simulations, and crowd-sourcing—are demanding advances in data analytics, data storage and retrieval, as well as data visualization. The complexity of the multi-dimensional systems that scientists and engineers want to model and analyze requires new computational abstractions. These are just two reasons that every scientific directorate and office at the National Science Foundation participated in the Cyber-enabled Discovery and Innovation[2], or CDI, program, an initiative started when I first joined NSF in 2007. CDI was in a nutshell "computational thinking for science and engineering."

Computational thinking has also begun to influence disciplines and professions beyond science and engineering. For example, areas of active study include algorithmic medicine, computational economics, computational finance, computational law, computational social science, digital archaeology, digital arts, digital humanities, and digital journalism. Data analytics is used in training Army recruits, detecting email spam and credit card fraud, recommending movies and books, ranking the quality of services, and personalizing coupons at supermarket checkouts. Machine learning is used by every major IT company for understanding human behavior and thus to tailor a customer's experience to his or her own preferences. Every industry and profession talks about Big Data and Cloud Computing. The combination of machine learning, especially deep learning, and large-scale compute infrastructure, built from CPUs, GPUS and FPGAs, underlies the current resurgence of artificial intelligence, with no end in sight. Cognitive tasks, such as image classification and speech recognition, performed computationally now reach human-level performance. Every industry —automotive, finance, healthcare, journalism, law, manufacturing— is being disrupted by technological advances in computer science. People working in those industries are going to have to think computationally.

3. COMPUTATIONAL THINKING AND EDUCATION

In the early-2000s, computer science had a moment of panic. Undergraduate enrollments were dropping. Computer science departments stopped hiring new faculty. One reason I wrote my 2006 CACM article on computational thinking was to inject some positive thinking into our community. Rather than bemoan the

[2] http://www.nsf.gov/crssprgm/cdi/

declining interest in computer science, I wanted us to shout to the world about the joy of computing, and more importantly, about the importance of computing. Sure enough, today enrollments are skyrocketing (again). Demand for graduates with computing skills far exceeds the supply; six-figure starting salaries offered to graduates with a B.S. in Computer Science are not uncommon.

3.1. *Undergraduate Education*

Campuses throughout the United States and abroad are revisiting their undergraduate curriculum in computer science. They are changing their first course in computer science to cover fundamental principles and concepts, not just programming. For example, Carnegie Mellon revised its undergraduate first-year courses to promote computational thinking for non-majors (Bryant, Sutner, & Stehlik, 2010). Harvey Mudd redesigned its introductory course with stellar success, including increasing the participation of women in computing (Klawe, 2013). At Harvard, «*In just a few short years CS50 has rocketed from being a middling course to one of the biggest on campus, with nearly 700 students and an astounding 102-member staff*» (Farrell, 2013). Harvard's CS50 is now Yale's most popular class (Annear, 2015). For MIT's introductory course to computer science, Eric Grimson and John Guttag say in their opening remarks «*I want to begin talking about the concepts and tools of computational thinking, which is what we're primarily going to focus on here. We're going to try and help you learn how to think like a computer scientist*» (Grimson & Guttag, 2008).

Many such introductory courses are now offered to or required by non-majors to take. Depending on the school, the requirement might be a general requirement (CMU) or a distribution requirement, e.g., to satisfy a science and technology (MIT), empirical and mathematical reasoning (Harvard), or a quantitative reasoning (Princeton) requirement.

3.2. *What about K-12?*

Not till computational thinking is taught routinely at K-12 levels of education will my vision be fully realized. Surprisingly, as a community, we have made faster progress at spreading computational thinking to K-12 than I had expected. We have professional organizations, industry, non-profits, and government policymakers to thank.

The College Board, with support from NSF, designed a new Advanced Placement (AP) course that covers the fundamental concepts of computing and computational thinking. This new course, called Computer Science Principles[3], launched in Fall 2016. Not coincidentally, the changes to the Computer Science AP course are consistent with the changes in introductory computer science courses taking place now on college campuses.

Industry is also promoting the importance of computing for all. Since 2006, with help from Google and later Microsoft, Carnegie Mellon has held summer workshops for high school teachers called "CS4HS." These workshops are designed to deliver the message that there is more to computer science than computer programming. CS4HS spread in 2007 to UCLA and the University of Washington. Since 2009, under the auspices of Google, CS4HS[4] had spread to over 40,000 teachers in over 40 countries reaching over one million students.

Launched in 2013, code.org[5] is a non-profit organization promoting making computer science accessible to all students. It is supported by donations from companies, foundations, and individuals. Many companies

[3] https://advancesinap.collegeboard.org/stem/computer-science-principles

[4] https://www.cs4hs.com/

[5] https://code.org/

support code.org out of their own need for more professionals trained with computer science skills. Code.org hosts a rich suite of educational materials and tools that run on many platforms, including smart phones and tablets. It lists local high schools and camps throughout the US where students can learn computing.

Computer science has also gotten attention from government leaders. President Barack Obama, in his 2016 State of the Union address, advocated for the nation to provide «*pre-K for all, offering every student the hands-on computer science and math classes that make them job-ready on day one*». Included in the Computer Science for All Initiative[6] he announced on January 30, 2016, is $120 million from the National Science Foundation. This money will be used to train as many as 9,000 more high school teachers to teach computer science and integrate computational thinking into their curriculum.

Computational thinking has also spread internationally. In January 2012, the British Royal Society published a report that says that «*Computational thinking' offers insightful ways to view how information operates in many natural and engineered systems*» and recommends that «Every child should have the opportunity to learn Computing at school» ("School" in the UK is the same as K-12 in the US.) Since that report, the UK Department for Education changed its national curriculum to include the mandatory study for computing starting Fall 2014 by all K-12 students in the UK (UK Department for Education, 2013). Much of the legwork behind this achievement was accomplished by the grassroots effort called "Computing at School"[7].

More recently, the Danish Growth Council published a 120-page report (The Danish Growth Council, 2016) with recommendations to the Danish government that include making computational thinking a mandatory component in education at all levels, from primary school through adult education. The report recognizes that «*the proper digital and analytical level of knowledge [be] secured among teachers throughout the educational system" and that "computational thinking must be part of the educational food chain*».

Other countries are also making rapid strides in the same direction. The Australian National Curriculum includes a Digital Technologies subject for K-10 students in which they «*use computational thinking and information systems to define, design and implement digital solutions*». China's Ministry of Education is discussing plans for making computational thinking be a required core competency for high school graduation. I am aware of similar efforts in Japan and Korea.

4. PROGRESS SO FAR AND WORK STILL TO DO

Ten years after the publication of my CACM Viewpoint, how far have we come? We have come a long way, along all dimensions: computational thinking has influenced the thinking in many other disciplines and many professional sectors; computational thinking, through revamped introductory computer science courses, has changed undergraduate curricula. We are making inroads in K-12 education worldwide.

While we have made incredible progress, our journey has just begun. We will see more and more disciplines make scholarly advances through the use of computing. We will see more and more professions transformed by their reliance on computing for conducting business. We will see more and more colleges and universities requiring an introductory computer science course to graduate. We will see more and more countries adding computer science to K-12 curricula.

Practical challenges and research opportunities remain. The main practical challenge is that we do not have enough K-12 teachers trained to teach computer science to K-12 students. I am optimistic that, over time,

[6] https://www.whitehouse.gov/the-press-office/2016/01/30/fact-sheet-president-obama-announces-computer-science-all-initiative-0

[7] http://www.computingatschool.org.uk/

we will solve this problem.

There also are interesting research questions that I would encourage computer scientists to pursue, working with the cognitive and learning sciences communities. First, what computer science concepts should be taught when, and how?

Consider an analogy to mathematics. We teach numbers to 5-year-olds, algebra to 12-year-olds and calculus to 18-year-olds. We have somehow figured out the progression of concepts to teach in mathematics, where learning one new concept builds on understanding the previous concept, and where the progression reflects the progression of mathematical sophistication of a child as he or she matures.

What is that progression in computer science? For example, when is it best to teach recursion? Children learn to solve the Towers of Hanoi puzzle (for small n) and in history class we teach "divide and conquer" as a strategy for winning battles. But is the general concept better taught in high school? We teach long division to 9-year-olds in 4th grade, but we never utter the word "algorithm." And yet the way it is taught, long division is just an algorithm. Is teaching the general concept of an algorithm too soon for a 4th grader? More deeply, are there concepts in computing that are innate and do not need to be formally learned?

Second, we need to understand how best to use computing technology in the classroom. Throwing computers in the classroom is not the most effective way to teach computer science concepts. How can we use technology to enhance the learning and reinforce the understanding of computer science concepts? How can we use technology to measure progress, learning outcomes and retention over time? How can we use technology to personalize the learning for individual learners, as each of us learn at a different pace and have different cognitive abilities?

We have made tremendous progress in injecting computational thinking into research and education of all fields in the last ten years. We still have a way to go, but fortunately, academia, industry and government forces are aligned toward realizing the vision of making computational thinking commonplace.

5. PERSONAL NOTES AND ACKNOWLEDGEMENTS

This piece is based in large part on a January 10, 2014 blog post "Computational Thinking Benefits Society"[8], written to celebrate the 40th anniversary of the book *Social Issues in Computing* by C.C. Gotlieb and A. Borodin, published in 1973. In turn, that post takes content from an article I wrote for Carnegie Mellon School of Computer Science's publication *The Link* (Wing, 2011), which was based on earlier unpublished writings authored with Jan Cuny and Larry Snyder. I thank them for letting me them use our shared prose for the blog post and for their own efforts in advocating computational thinking.

Looking back over how much progress has been made in spreading computational thinking, I am grateful for the opportunity I had while I was the Assistant Director of the Computer and Information Science and Engineering (CISE) Directorate of the National Science Foundation. I had a hand in CDI from its start. I also helped start the Computing Education for the 21st Century program in 2010, designed to help K-12 students, as well as first- and second-year college students, and their teachers develop computational thinking competencies. These two programs allowed me—through the reach of NSF—to spread computational thinking directly to the science and engineering research (CDI) and education (CE21) communities in the US. Jan Cuny's initiative and persistence led to NSF's efforts with the College Board and beyond.

I thank Michael E. Caspersen[9], Professor in Computing Education and director of Center for Computational Thinking at Department of Computer Science, Aarhus University, Denmark, for pointing me to the Danish

[8] http://socialissues.cs.toronto.edu/index.html%3Fp=279.html

[9] http://www.cs.au.dk/~mec/

Growth Council report and providing the translations of the relevant passages.
Since the publication of my CACM article (Wing, 2006), which has been translated into Chinese, French, German, Italian, Japanese, Korean, and Portuguese, I have received hundreds of email messages from people of all walks of life—from a retired grandfather in Florida to a mother in central Pennsylvania to a female junior high school student in Seattle, from a Brazilian news reporter to the head of a think tank in Sri Lanka to an Egyptian student blogger, from artists to software developers to astrophysicists—thanking me for inspiring them and eager to support my cause. I am grateful to everyone's support.
Besides the citations I gave in text, I recommend the following references: CSUnplugged (Bell, Witten, & Fellows, 2015) for teaching young children about computer science without using a machine; the textbook used in MIT's 6.00 Introductory to Computer Science and Programming (Guttag, 2016); a book on the breadth of computer science, inspired by Feynman lectures for physics (Hey & Papay, 2014); a framing for principles of computing (Denning, 2010); and two National Research Council workshop reports (National Research Council, 2010; 2011), as early attempts to scope out the meaning and benefits of computational thinking.

6. REFERENCES

Aho, A. V., & Ullman, J. D. (1992). *Foundations of computer science*. Rockville, MD: Computer Science Press. Retrieved from http://infolab.stanford.edu/~ullman/focs.html

Annear, S. (2015, November 17). Yale's most popular course is a Harvard class. *Boston Globe*. Retrieved from https://www.bostonglobe.com/metro/2015/11/17/yale-most-popular-course-harvard-class/OsJs36GYID8t-ViXKDUEaqM/story.html

Bell, T., Witten I., & Fellows, M. (2015). Computer Science Unplugged. Retrieved from http://csunplugged.org/wp-content/uploads/2015/03/CSUnplugged_OS_2015_v3.1.pdf

Bryant, R. E., Sutner, K., & Stehlik, M. J. (2010). *Introductory computer science education at Carnegie Mellon University: A deans' perspective*. Pittsburgh, PA: Carnegie Mellon University. Retrieved from http://repository.cmu.edu/compsci/2554/

Denning, P. J. (2010). The great principles of computing. *American Scientist*, *98*, 369-372. Retrieved from https://www.americanscientist.org/libraries/documents/20108101750328103-2010-09denning-computingscience.pdf

Farrell, M. B. (2013, November 16). Computer science fills seats, needs at Harvard. *Boston Globe*. Retrieved from http://www.bostonglobe.com/business/2013/11/26/computer-science-course-breaks-stereotypes-and-fills-halls-harvard/7XAXko7O392DiO1nAhp7dL/story.html

Grimson, E., & Guttag, J. (2008). 6.00 Introduction to Computer Science and Programming, Fall 2008 [Course]. Retrieved from https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00-introduction-to-computer-science-and-programming-fall-2008/

Guttag, J. V. (2016). *Introduction to computation and programming using Python with application to understanding data, Second Edition*. Cambridge, MA: MIT Press.

Hey, T., & Papay, G. (2014). *The computing universe*. Cambridge, UK: Cambridge University Press.

Klawe, M. (2013, December 18). Increasing the participation of women in computing careers [Blog post]. Retrieved from http://socialissues.cs.toronto.edu/2013/12/women/

Levin, L. A. (2013). Universal heuristics: How do humans solve “unsolvable” problems?. In D. L. Dowe (Ed.), *Algorithmic probability and friends. Bayesian prediction and artificial intelligence* (pp. 53-54). Berlin, DE: Springer.

National Research Council (2005). *Catalyzing inquiry at the interface of computing and biology*. Washington, DC: The National Academies Press. doi:10.17226/11480

National Research Council (2009). *A new biology for the 21st century*. Washington, DC: The National Academies Press. doi:10.17226/12764

National Research Council (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: The National Academies Press. doi:10.17226/12840

National Research Council (2011). *Report of a workshop on pedagogical aspects of computational thinking.* Washington, DC: The National Academies Press. doi:10.17226/13170

President’s Information Technology Advisory Council (2005). *Computational science: Ensuring America’s competitiveness*. Washington, DC. Retrieved from http://www.dtic.mil/docs/citations/ADA462840

The Danish Growth Council (2016). *Report on qualified labour*. Copenhagen, DK. Retrieved from http://danmarksvaekstraad.dk/file/634221/Rapport_om_kvalificeret_arbejdskraft.pdf

UK Department for Education (2013). *Computing programmes of study for Key Stages 1-4*. Retrieved from http://media.education.gov.uk/assets/files/pdf/c/computing%2004-02-13_001.pdf

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

Wing, J. M. (2008). Five deep questions in computing. *Communications of the ACM, 51*(1), 58-60. doi:10.1145/1327452.1327479

Wing, J. M. (2011, March 06). Computational thinking: What and why. *The Link*. Retrieved from http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why