Computational Thinking: What and Why?

Jeannette M. Wing
17 November 2010

In my March 2006 CACM article I used the term "computational thinking" to articulate a vision that everyone, not just those who major in computer science, can benefit from thinking like a computer scientist [Wing06]. So, what is computational thinking? Here is a definition that Jan Cuny of the National Science Foundation, Larry Snyder of the University of Washington, and I use; it is inspired by an email exchange I had with Al Aho of Columbia University:

> **Computational Thinking** is 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 [CunySnyderWing10]

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, or more generally, by combinations of humans and machines.

When I use the term computational thinking, my interpretation of the words "problem" and "solution" is broad; in particular, I mean not just mathematically well-defined problems whose solutions are completely analyzable, e.g., a proof, an algorithm, or a program, but also real-world problems whose solutions might be in the form of large, complex software systems. Thus, computational thinking overlaps with logical thinking and systems thinking. It includes algorithmic thinking and parallel thinking, which in turn engage other kinds of thought processes, e.g., compositional reasoning, pattern matching, procedural thinking, and recursive thinking.

Computational thinking is used in the design and analysis of problems and their solutions, broadly interpreted. The most important and high-level thought process in computational thinking is the abstraction process. Abstraction is used in defining patterns, generalizing from 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.

Recursively applying abstraction gives us the ability to build larger and larger systems, with the base case (at least for 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 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 be a correct implementation of the semantics of the language. As another example, the narrow waist architecture of the Internet, with TCP/IP at the middle, enabled a multitude of unforeseen applications to proliferate at the highest layer, and a multitude of unforeseen hardware platforms, communications media, and devices to proliferate at the lowest.

Computational thinking draws on both mathematical thinking and engineering thinking. Unlike in mathematics, however, our computing systems are constrained by the physics of the underlying information-processing agent and its operating environment. And so, we must worry about boundary conditions, failures, malicious agents, and the unpredictability of the real world. But unlike other engineering disciplines, because of software (our unique "secret weapon"), in computing we can build virtual worlds that are unconstrained by physical reality. And so, in cyberspace our creativity is limited only by our imagination.

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 [PITAC 2005]. The expedited sequencing of the human genome through the shotgun algorithm awakened the interest by the biology community in computational methods, not just computational artifacts (computers and networks). The volume and rate at which scientists and engineers are now collecting and producing data—through instruments, experiments, and simulations—are demanding advances in data analytics, data storage and retrieval, and data visualization. The five-year National Science Foundation Cyber-enabled Discovery and Innovation program, with an FY11 budget request of $100M, is in a nutshell "computational thinking for science and engineering." Every scientific directorate and office in NSF participates in CDI.

Computational thinking has also begun to influence disciplines and professions beyond science and engineering. For example, areas of active study include algorithmic medicine, computational archaeology, computational economics, computational finance, computation and journalism, computational law, computational social science, and digital humanities. Data analytics is used in training Army recruits, spam and credit card fraud detection, recommendation and reputation services, and personalizing coupons at supermarket checkout.

At Carnegie Mellon, computational thinking is everywhere. We have degree programs, minors, or tracks in "computational X" where X is applied mathematics, biology, chemistry, design, economics, finance, linguistics, mechanics, neuroscience, physics, statistical learning. We even have a course in computational photography. We have programs in computer music, and in computation, organizations, and society. We have joint programs between computer science and other disciplines, e.g., algorithms, combinatorics, and optimization (computer science, mathematics, business); computer science and arts; entertainment technology (computer science and drama); human-computer interaction (computer science, design, and psychology); language technologies (computer science and linguistics); logic and computation (computer science and philosophy); pure and applied logic (computer science, math, and philosophy); and robotics (computer science, electrical and computer engineering, and mechanical engineering).

Computational Thinking in Daily Life

Computational thinking can be applied in daily life. Faculty in the Carnegie Mellon Computer Science Department provided the following stories to me:

Pipelining. Randy Bryant, Dean of the School of Computer Science, was pondering how to make the Sunday afternoon School's graduation ceremony go faster. By careful placement of where individuals stood, he designed an efficient pipeline so that upon each graduate's name and honors

read by Mark Stehlik, each person could receive his or her diploma, get a handshake or hug from Mark, and get his or her picture taken. This pipeline allowed a steady stream of students to march across the stage. (A pipeline stall would often occur when the graduate's cap would topple while getting hug from Mark.)

Seth Goldstein observed: "A buffet line. Why do they always put the dressing *before* the salad? The sauce *before* the main dish? The silverware at the *start*? They need some pipeline theory."

Hashing. After a talk I gave at a department faculty meeting on my computational thinking vision, Danny Sleator came up to tell me that at his home they use a particular hashing function to store away Lego pieces. According to Sleator, they hash on these categories: rectangular thick blocks, other thick (non-rectangular) blocks, thins (of any shape), wedgies, axles, rivets and spacers, fits on axle, ball and socket, and miscellaneous. They even have rules to classify pieces that could fit into more than one category. He said that "Even though this is pretty crude, it saves about a factor of 10 when looking for a piece." Avrim Blum overheard Danny telling me this story and chimed in to say "At our home, we use a different hash function."

Sorting. The following story is taken verbatim from an email sent by Roger Dannenberg, a professional trumpeter and computer scientist. "I showed up to a big band gig, and the band leader passed out books with maybe 200 unordered charts and a set list with about 40 titles we were supposed to get out and place in order, ready to play. Everyone else started searching through the stack, pulling out charts one-at-a-time. I decided to sort the 200 charts alphabetically O(N log(N)) and then pull the charts O(M log(N)). I was still sorting when other band members were half-way through their charts, and I started to get some funny looks, but in the end, I finished first. That's computational thinking."

Benefits of Computational Thinking

Computational thinking is the new literacy of the 21st Century. It enables you to bend computation to your needs. Why should everyone learn a little computational thinking? Cuny, Snyder, and I advocate these benefits [CunySnyderWing10]:

Computational thinking for everyone means being able to

- Understand what aspects of a problem are amenable to computation
- Evaluate the match between computational tools and techniques and a problem
    - Understand the limitations and power of computational tools and techniques
- Apply or adapt a computational tool or technique to a new use
- Recognize an opportunity to use computation in a new way
- Apply computational strategies such divide and conquer in any domain

Computational thinking for scientists, engineers, and other professionals further means being able to

- Apply new computational methods to their problems
- Reformulate problems to be amenable to computational strategies
- Discover new "science" through analysis of large data
- Ask new questions that were not thought of or dared to ask because of scale, easily addressed computationally
- Explain problems and solutions in computational terms

Computational Thinking in Education

Campuses throughout the US and abroad are revisiting their undergraduate curriculum in computer science. Many are changing their first course in computer science to cover fundamental principles and concepts, not (just) programming. For example, at Carnegie Mellon we recently revised our undergraduate first-year courses to promote computational thinking for non-majors [Link10].

Moreover, the interest and excitement surrounding computational thinking has grown beyond research and undergraduate education. The potential of spreading computational thinking broadly across the population has motivated several recent projects, sketched below. Most focus on K-12. Sponsors range across professional organizations, government, academia, and industry. Computational thinking has also spread internationally.

The College Board, with support from NSF, is designing a new Advanced Placement (AP) course that covers the fundamental concepts of computing and computational thinking (http://csprinciples.org). Five universities are piloting versions of this course this year: University of North Carolina-Charlotte, UC Berkeley, Metropolitan Sate College of Denver, UC San Diego, and University of Washington. The plan is for more schools–high schools, community colleges, and universities–to participate next year.

The National Academies' Computer Science and Telecommunications Board held a series of workshops on "Computational Thinking for Everyone" with a focus on identifying the fundamental concepts of computer science that can be taught to K-12 students. The first workshop report [NRC10] provides multiple perspectives on computational thinking.

On May 29, 2009, an event on The Hill sponsored by ACM, CRA, CSTA, IEEE, Microsoft, NCWIT, NSF, and SWE , called for putting the "C" (computer science) into "STEM." The US House of Representatives designated the first week of December as Computer Science Education Week (http://www.csedweek.org/). CSEdWeek is sponsored by ABI, ACM, BHEF, CRA, CSTA, Dot Diva, Google, Globaloria, Intel, Microsoft, NCWIT, NSF, SAS, and Upsilon Pi Epsilon. On July 30, 2010 Rep. Jared Polis (D-CO) introduced the Computer Science Education Act (H.R.5929) to strengthen K-12 computer science education.

In September 2010, NSF started the Computing Education for the 21st Century (CE21) program to develop computational thinking competencies for K-14 students and teachers. CE21 builds on the successes of the NSF CISE Pathways to Revitalized Undergraduate Computing Education (CPATH) and Broadening Participating in Computing (BPC) programs. CE21 has a special emphasis on activities that support the CS 10K Project, an initiative launched by NSF through BPC that aims to catalyze a revision of high school curriculum, with the proposed new AP course as a centerpiece, and to prepare 10,000 teachers to teach the new courses in 10,000 high schools by 2015.

In August 2010, the British Royal Society announced that it is leading an 18-month project to look "at the way that computing is taught in schools, with support from 24 organisations from across the computing community including learned societies, professional bodies, universities, and industry" (http://royalsociety.org/Education-Policy/Projects/). One responder is the UK Computing At School (CAS), which is a coalition run by the British Computing Society and supported by Microsoft Research and other industry partners.

Since 2006, with help from Google and later Microsoft, Carnegie Mellon has held CS4HS summer workshops for high school teachers to take away an understanding that there is more to computer science than computer programming. CS4HS spread in 2007 to UCLA and the University of Washington. By 2010, under the auspices of Google, CS4HS spread to 20 schools in the US and 14 in Europe, the Middle East, and Africa.

Since 2007, Microsoft Research has funded the Carnegie Mellon Center for Computational Thinking: http://www.cs.cmu.edu/~CompThink/ . The Center supports both research and educational outreach projects.

In October 2010, Google launched the Exploring Computational Thinking website (http://www.google.com/edu/computational-thinking/index.html). It has a wealth of links to further web resources, including lesson plans for K-12 teachers in science and mathematics.

Computer Science Unplugged, http://csunplugged.org/, created by Tim Bell, Mike Fellows, and Ian Witten, teaches computer science without the use of a computer. It is especially appropriate for elementary and middle school children. Several dozen people working in many countries, including New Zealand, USA, Sweden, Australia, China, Korea, Taiwan and Canada, contribute to this extremely popular website.

Additionally, panels and discussions on computational thinking have been plentiful at venues such as SIGCSE and the ACM Educational Council. The CRA-E presented a white paper [CRA-E10] at the July 2010 CRA Snowbird conference, which includes recommendations for computational thinking courses for non-majors. CSTA produced and disseminates *Computational Thinking Resource Set: A Problem-Solving Tool for Every Classrooom (*http://www.csta.acm.org/).

Final Remarks

Computational thinking is not just or all about computing. The educational benefits of being able to think computationally transfer to any domain by enhancing and reinforcing intellectual skills.

Computer scientists see the value of thinking abstractly, thinking at multiple levels of abstraction, abstracting to manage complexity, abstracting to deal with scale, etc. We know the value of these capabilities. Our immediate task ahead is to better explain to non-computer scientists what we mean by computational thinking and the benefits of being able to think computationally. Please join me in helping to spread the word!

Bibliography

[CRA-E10] Computer Research Association, "Creating Environments for Computational Researcher Education," August 9, 2010.
http://www.cra.org/uploads/documents/resources/rissues/CRA-E-Researcher-Education.pdf

[CunySnyderWing10] Jan Cuny, Larry Snyder, and Jeannette M. Wing, "Demystifying Computational Thinking for Non-Computer Scientists," work in progress, 2010.

[NRC10] Report of a Workshop on the Scope and Nature of Computational Thinking, National Research Council, 2010 http://www8.nationalacademies.org/cp/projectview.aspx?key=48969

[Link10] A Deans' Perspective, The Link, September 2010.

[PITAC05] President's Information Technology Advisory Council, "Computational Science: Ensuring America's Competitiveness," Report to the President, June 2005.

[Wing06] Jeannette M. Wing, "Computational Thinking," Communications of the ACM, CACM vol. 49, no. 3, March 2006, pp. 33-35.

Acronyms:

ABI: Anita Borg Institute for Women and Technology
ACM: Association for Computing Machinery
BHEF: Business-Higher Education Forum
CISE: Computer and Information Science and Engineering
CRA: Computing Research Association
CRA-E: Computing Research Association-Education
CSTA: Computer Science Teachers Association
CSTB: Computer Science and Telecommunications Board
IEEE: Institute for Electrical and Electronic Engineers
NCWIT: National Center for Women and Information Technology
NSF: National Science Foundation
SIGCSE: ACM Special Interest Group on Computer Science Education
SWE: Society for Women Engineers