Communications of the ACM
Volume 49, Number 3 (2006), Pages 33-35

Viewpoint: Computational thinking

Jeannette M. Wing

Table of Contents

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our advantage, and whether false positives or false negatives are allowed. Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system's design for simplicity and elegance.

Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable. It is using invariants to describe a system's behavior succinctly and declaratively. It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail. It is modularizing something in anticipation of multiple users or prefetching and caching in anticipation of future use.

Computational thinking is thinking in terms of prevention, protection, and recovery from worst-case scenarios through redundancy, damage containment, and error correction. It is calling gridlock deadlock and contracts interfaces. It is learning to avoid race conditions when synchronizing meetings with one another.

Computational thinking is using heuristic reasoning to discover a solution. It is planning, learning, and scheduling in the presence of uncertainty. It is search, search, and more search, resulting in a list of Web pages, a strategy for winning a game, or a counterexample. Computational thinking is using massive amounts of data to speed up computation. It is making trade-offs between time and space and between processing power and storage capacity.

Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction.

Consider these everyday examples: When your daughter goes to school in the morning, she puts in her backpack the things she needs for the day; that's prefetching and caching. When your son loses his mittens, you suggest he retrace his steps; that's backtracking. At what point do you stop renting skis and buy yourself a pair?; that's online algorithms. Which line do you stand in at the supermarket?; that's performance modeling for multi-server systems. Why does your telephone still work during a power outage?; that's independence of failure and redundancy in design. How do Completely Automated Public Turing Test(s) to Tell Computers and Humans Apart, or CAPTCHAs, authenticate humans?; that's exploiting the difficulty of solving hard AI problems to foil computing agents.

Computational thinking will have become ingrained in everyone's lives when words like algorithm and precondition are part of everyone's vocabulary; when nondeterminism and garbage collection take on the meanings used by computer scientists; and when trees are drawn upside down.

We have witnessed the influence of computational thinking on other disciplines. For example, machine learning has transformed statistics. Statistical learning is being used for problems on a scale, in terms of both data size and dimension, unimaginable only a few years ago. Statistics departments in all kinds of organizations are hiring computer scientists. Schools of computer science are embracing existing or starting up new statistics departments.

Computer scientists' recent interest in biology is driven by their belief that biologists can benefit from computational thinking. Computer science's contribution to biology goes beyond the ability to search through vast amounts of sequence data looking for patterns. The hope is that data structures and algorithms our computational abstractions and methods can represent the structure of proteins in ways that elucidate their function. Computational biology is changing the way biologists think. Similarly, computational game theory is changing the way economists think; nanocomputing, the way chemists think; and quantum computing, the way physicists think.

This kind of thinking will be part of the skill set of not only other scientists but of everyone else. Ubiquitous computing is to today as computational thinking is to tomorrow. Ubiquitous computing was yesterday's dream that became today's reality; computational thinking is tomorrow's reality.

What It Is, and Isn't

Computer science is the study of computation what can be computed and how to compute it. Computational thinking thus has the following characteristics:

- *Conceptualizing, not programming*. Computer science is not computer programming. Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction;
- *Fundamental, not rote skill*. A fundamental skill is something every human being must know to function in modern society. Rote means a mechanical routine. Ironically, not until computer science solves the AI Grand Challenge of making computers think like humans will thinking be rote;
- *A way that humans, not computers, think*. Computational thinking is a way humans solve problems; it is not trying to get humans to think like computers. Computers are dull and boring; humans are clever and imaginative. We humans make computers exciting. Equipped with computing devices, we use our cleverness to tackle problems we would not dare take on before the age of computing and build systems with functionality limited only by our imaginations;
- *Complements and combines mathematical and engineering thinking*. Computer science inherently draws on mathematical thinking, given that, like all sciences, its formal foundations rest on mathematics. Computer science inherently draws on engineering thinking, given that we build systems that interact with the real world. The constraints of the underlying computing device force computer scientists to think computationally, not just mathematically. Being free to build virtual worlds enables us to engineer systems beyond the physical world;
- *Ideas, not artifacts*. It's not just the software and hardware artifacts we produce that will be physically present everywhere and touch our lives all the time, it will be the computational concepts we use to approach and solve problems, manage our daily lives, and communicate and interact with other people; and
- *For everyone, everywhere*. Computational thinking will be a reality when it is so integral to human endeavors it disappears as an explicit philosophy.

Many people equate computer science with computer programming. Some parents see only a narrow range of job opportunities for their children who major in computer science. Many people think the fundamental research in computer science is done and that only the engineering remains. Computational thinking is a grand vision to guide computer science educators, researchers, and practitioners as we act to change society's image of the field. We especially need to reach the pre-college audience, including teachers, parents, and students, sending them two main messages:

- *Intellectually challenging and engaging scientific problems remain to be understood and solved*. The problem domain and solution domain are limited only by our own curiosity and creativity; and
- *One can major in computer science and do anything*. One can major in English or mathematics and go on to a multitude of different careers. Ditto computer science. One can major in computer science and go on to a career in medicine, law, business, politics, any type of science or engineering, and even the arts.

Professors of computer science should teach a course called "Ways to Think Like a Computer Scientist" to college freshmen, making it available to non-majors, not just to computer science majors. We should expose pre-college students to computational methods and models. Rather than bemoan the decline of interest in computer science or the decline in funding for research in computer science, we should look to inspire the public's interest in the intellectual adventure of the field. We'll thus spread the joy, awe, and power of computer science, aiming to make computational thinking commonplace.

Author

Jeannette M. Wing (wing@cs.cmu.edu) is the President's Professor of Computer Science in and head of the Computer Science Department at Carnegie Mellon University, Pittsburgh, PA.