A Plea for Modesty

David Hemmendinger
Department of Computer Science
Union College, Schenectady, NY 12308
hemmendd@union.edu

From time to time a movement arises that promises to save the world, or at least to make it vastly better. The extraordinary achievements of digital computing make it a locus of such movements today. Yet we should be wary; when movements fail they provoke backlash that rejects the more limited gains that they might have afforded. Today "computational thinking" has a considerable following, and I would like to discuss some problems with its discourse. It is too often presented in terms that could be interpreted as arrogant or that are overstated. Its descriptions too often lack appropriate examples, and perhaps as a result, it gets misunderstood in casual writing.

Since Jeannette Wing wrote her CACM essay in praise of computational thinking four years ago [11], we have had numerous discussions of the idea in print and in conferences. Some, such as an ITiCSE panel last summer [5], are devoted to deciding what it is, but nearly all appear to view it as a Good Thing. Consider, for example, a National Science Foundation Q&A:

> Q: Must I include Computational Thinking (CT) in my proposal?
> A: Yes, all CPATH proposals must demonstrate how Computational Thinking is incorporated within the project. Since CT is fundamental to virtually all disciplines, this should be a natural part of any transformative vision focusing on a single discipline or across disciplines [9].

So what is computational thinking? Wing's article offers some characterizations: Computational thinking "is reformulating a seemingly difficult problem into one we know how to solve", "is thinking recursively", "is using abstraction and decomposition when attacking a large complex task", "is separation of concerns", "is using invariants to describe a system's behavior succinctly", "is using heuristic reasoning".

However important all these are in thinking about computing or in thinking with it, they are scarcely peculiar to computing. Reformulating hard problems is typical of all domains of problem-solving. Philosophers have been thinking about thinking — recursively — for a long time. Mathematics surely uses abstraction, and so do all disciplines that build models. Separation of concerns and using heuristics also characterizes problem-solving in general. Invariants: think of the conservation laws in physics, or of formulations of mechanics in terms of Hamiltonians or of the principle of least action. In short, Wing has characterized problem-solving, mathematics, scientific reasoning in general, and the use of models.

Perhaps less seriously, Wing's article adds that computational thinking will be part of our lives when "garbage collection take[s] on the meanings used by computer scientists; and when trees are drawn upside down." But despite the merits of recycling, surely there is some garbage that belongs on the rubbish heap and not on the free heap. And as to trees, we might paraphrase (with apologies to Joyce Kilmer):

> Inverted trees are drawn by fools like me,
> But only god can make an upright tree.

Consider analogous statements made about other disciplines

Chemical thinking will be part of our lives when bonding is understood in terms of valence electrons and when moles no longer burrow underground.

> Mathematical thinking will be part of our lives when theater marqueés use deltas and epsilons to advertise continuous movie showings and when group identities are actual group members.

I doubt that anyone would think that this broad application of technical terms and concepts is either necessary or sufficient to make chemical or mathematical understanding part of our lives, or even that such application is desirable.

Apart from such exaggerations, a problem with Wing's article is the central role that it gives Us — computer scientists. The final paragraph suggests that we should offer a course to first-year college students, "Ways to Think Like a Computer Scientist". A course on what computing can and can't do and on how to use it could indeed be valuable. Many other college departments offer introductions to their disciplines with the goal of showing non-majors what the subject is all about. They are not generally called "Ways to Think Like a {Mathematician, Historian, Chemist, etc}", however. They are have topical titles, or perhaps one as broad as "What is Mathematics", even if one of their goals is to show what characterizes modes of thinking in the discipline.

Another part of the article describes everyday activities in computational terms, as if, like Molière's Monsieur Jourdain who discovered that he'd been speaking prose all his life without knowing it, we may all be computational thinkers in daily life. When a child puts school supplies in a backpack, that's prefetching and caching, and retracing one's steps to find a lost item is backtracking. Describing the backpack as a cache is not a bad idea — in fact, a Web search turns up several explanations of cache memory that use the backpack analogy. As with step-retracing, though, the connection goes in the wrong direction when this is offered as an example of our thinking computationally. Caches in computing are an application of a common practice — keeping frequently used things close at hand; backtracking in computational search is an instance of another common practice. Those practices do not become computational by virtue of finding application in computing.

Describing computational thinking in these terms can have an imperialistic flavor. An interview with Joan Peckham has a passage that illustrates it. She spoke about seeing a video on teaching science and said:

> We all looked at it and thought: "But it's also computational thinking!"
>
> In the course of teaching elementary school students about honey bees, he took them out on the playground and asked them to act out what the honey bees did: leaving the hive, finding the pollen, giving directions to the other bees. Then he brought them back into the classroom, went to a whiteboard, and engaged them in activities that I would identify as modeling, debugging, and drawing finite state diagrams. He didn't call them that, but that's what they were.
>
> Yes he was teaching them science, but the way he was analyzing the subject, and engaging them in analysis, clearly involved a set of computational constructs [10].

Constructing models, finding and correcting errors, drawing diagrams, analyzing — these are all parts of scientific (and other) activities. They are computational constructs, however, only in that they are also important to computer science. To say that because the teacher was doing what we also do, he was using computational constructs is to say "it's ours!"

At such times, in fact, it sounds as if many of the claims for computational thinking amount to, "If it's a good way of thinking, then it's ours." I once remarked on this to Wing, who of course said that she didn't mean that at all. But the appearance of grand territorial claims risks provoking adverse reactions, and leads to remarks such as Denning quotes, "You computer scientists are hungry! First you wanted us to take your courses on literacy and fluency. Now you want us to think like you!" [6]

There is a systematic ambiguity in statements of the form, "computational thinking is X", as in "computational thinking is using abstraction and decomposition". As a public figure said a few years ago, "It depends what the meaning of ’is’ is." If it is the ’is’ of predication, as in "grass is green", then such statements are quite straightforward: abstraction is one property of computational thinking (and may also be a property of many other sorts of thinking). If it is more nearly akin to the ’is’ of identity, however, then things are more difficult. In that case, "computational thinking is X" appears convertible with "X is computational thinking" But then all of the claims about what computational thinking includes can readily be taken to mean also that each of those activities *belongs* to computational thinking. That is, statements that "computational thinking is ..." recursive thinking, problem reduction using abstraction and decomposition, and all the rest can be construed as: these activities all belong to computational thinking.

Some arguments for the value of everyone’s learning to think like a computer scientist actually take us in the wrong direction. They emphasize algorithmic thinking, describing it in terms that resemble what one writes in a procedural language, with conditionals and iteration. This is not the place to get into a discussion of the virtues of non-procedural programming languages, but these arguments amount to recommending that people learn to think in the way in which they would have to program in a standard machine instruction set; i.e, at a low level. In a recent article in which he discusses how to make computing more accessible, Mark Guzdial describes some computing-education research that suggests that people should not be encouraged to think in such low-level terms [7]. The studies showed that when people were asked to describe tasks in procedural terms, they generally didn’t specify control flow — explicit iteration or conditionals — and that when given a task description that lacked explicit control flow, they had no difficulty in understanding what was to be done. If this is so, then teaching people to think in standard programming-language terms is a striking example of "dumbing down" and scarcely what we want computational thinking to be. If, as Guzdial (and Wing) say, people should become familiar with the metaphors of computation, we need to cast them in the terms that take into account how people rather than machines learn.

A Web site, IAE-pedia, uses a low-level notion of algorithm in its article on computational thinking [8]. It describes doing a dictionary lookup with linear search and observes that this is a poor algorithm. It goes on to say that it can be quite hard to write out the algorithm that we use, and hard for a third-grader to follow it if written, and it concludes that simply looking up a word definition with Google is a good example of computational thinking: it combines our interests with machine search capabilities. It’s easier and faster for a child to learn to use Google than a dictionary.

We need not get into the merits of these ways of looking up definitions. Writing out a procedure for a child to follow, rather than inviting the child to figure out a good way to look up words, or showing by example how to do it, is a very poor way of teaching. Casting computational thinking in such machine-level terms does it a disservice. I don’t intend to suggest that proponents of computational thinking are responsible for what’s in a wiki article, but they need to be aware of how the concept becomes distorted.

Let’s return to the question of what computational thinking is. The Carnegie Mellon Center for Computational Thinking [3] offers an example: an algorithm for matching kidney donors and recipients that can build chains of donor-recipient pairs to find a sequence of compatible transplants. It is a striking example of a computational solution of a difficult and important problem, but it was developed by a quite eminent computer scientist on the CMU faculty. If we are to show the value of computational thinking, we need to find people other than computer scientists who are using it. A much more prosaic example was the subject of a seminar that a mechanical-engineering colleague gave recently: on using machine-learning to improve the performance of control systems. Although control systems have a well-developed theory, here is a small instance of being able to ask new questions and get new answers with computation.

Alan Bundy describes a seminar series on computational thinking at Edinburgh [2]. Like Guzdial, he speaks of the importance of computational metaphors; for example, a psychologist studies facial recognition as a computational process and uses concepts such as "nearest neighbor". A biologist models protein interaction with process algebras, and in addition to the computational models of mind that philosophers have discussed, a philosopher uses the notion of an emulator to study planning.

New wine in old bottles may spoil, but new metaphors in old disciplines can stimulate thinking. We need to distinguish between metaphors and mere jargon, however. Terms like "context switch" and "multitasking" have entered everyday discourse, though garbage collection has not yet acquired its computational meaning. I do not think that using these terms has led people to use the computational ideas or structures that underlie them, and I hope that we will not urge that people adopt the outward garb of computational thinking — the lingo — rather than its substance.

The Edinburgh seminar series had speakers from a wide range of fields. Several astronomers spoke of computer modeling as the closest thing that they had to a laboratory. A musician talked about algorithmic composition both old and new. A linguist described modeling of adaptive systems and a geographer, ways of visualizing information and "zooming in" on both geographical maps and other representations of information (metaphors abound here). Computation is also used to help to remind lawyers and physicians of information relevant to their investigations, and to help recognize art forgeries.

What do all of these examples of thinking with computation. have in common? I don't want to offer yet another definition of computational thinking, which is as varied as the examples are. Some of the capabilities that computation offers, however, appear common. Modeling is ubiquitous in problem-solving domains, and modeling benefits greatly from the speed and data-handling capacity of digital computers, allowing us to build new sort of models readily. Machine learning can play a role — again, employing massive data processing to extract information that we would not be able to obtain in other ways. Computation lets us display information effectively with high-performance graphics.

Of course thinking with computation uses abstraction and resolution of complex problems into simpler ones — as do all sorts of thinking. Of course computational thinking uses algorithms. As every algorithms text tells us, they aren't new; the word comes from the name of a 12th century Persian mathematician, and we find procedural thinking laid out in Babylonian tablets nearly four millennia old. What the theory of computation has contributed to algorithmic thinking is to make precise various notions of complexity, something that had not been a part of the mathematical study of algorithms until recently. Whether complexity is measured by space or time required or by power dissipated, computational thinking leads us to pay attention to scalability and feasibility, as Wing says. Thinking with computation requires us to be resource-aware.

"Ways to Think Like a Computer Scientist" may be an interesting course, just as it may be interesting to learn how mathematicians or historians think. Teaching computational thinking, however is something else; not to lead people to think like us — which is pretty varied anyway. Instead, it is to teach them how to think like an economist, a physicist, an artist, and to understand how to use computation to solve their problems, to create, and to discover new questions that can fruitfully be explored. Computer scientists can contribute, but we should be careful not to speak as if we are the ones to lead people to a promised land. In the end, though, perhaps we should talk less about computational *thinking*, and focus more on computational *doing* — carrying out one's work (and one's play!) in new ways by using computational tools. As Owen Astrachan wrote recently, "let's make sure we don't lose sight of computational doing" [1].

My plea for modesty, then, is that we talk about the value of computation and of its methods without appearing to lay claim to very broadly applicable ways of thinking. It is that we write papers and give talks about striking examples of thinking computationally, but without "computational thinking" in the title. It is that we avoid suggesting that if only people would be more like us, the world would be so much better. It is that we focus on *showing* examples of what can be done by thinking with computation rather than by talking about its attributes — and as both Denning and Astrachan urge, to give due attention to computational *doing*.

There is a nice quote by Leo Cherne, a twentieth-century economist and public servant, one that numerous Web pages erroneously attribute to Einstein:

> The computer is incredibly fast, accurate, and stupid. Man is unbelievably slow, inaccurate, and brilliant. The marriage of the two is a force beyond calculation [4].

Jeannette Wing writes in similar terms:

> 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.

Let us agree on this, and agree also that thinking well is not the province of any one discipline.

References

1. Astrachan, O., "Out-of-the-box: cogito ergo hack," *inroads (ACM SIGCSE Bulletin)* **41**(2), p. 80 (June, 2009). `http://doi.acm.org/10.1145/1595453.1595476`.
2. Bundy, A, "Computational thinking is pervasive," *Journal of Scientific and Practical Computing* **1**(2), pp. 67-69 (2007).
3. Center for Computational Thinking, Carnegie Mellon University. `http://www.cs.cmu.edu/˜CompThink` (retrieved 4 Jan. 2010).
4. Cherne, L., remarks at the Discover America meeting, Brussels, June 27, 1968. as cited in "Computer science: A neglected area in schools of education," Gary D. Brooks, *The Phi Delta Kappan 53*, 2 (Oct., 1971), pp. 121-122. `http://www.jstor.org/stable/20373101?seq=2`.
5. Curzon, P., Peckham, J., Settle, A., and Roberts, E., "Computational Thinking (CT): On Weaving It In," *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education,* pp. 201-202 (2009). `http://doi.acm.org/10.1145/1562877.1562941`.
6. Denning, P. J., "The profession of IT: Beyond computational thinking," *Communications of the ACM* **52**(6), pp. 28-30 (June, 2009). `http://doi.acm.org/10.1145/1516046.1516054`.
7. Guzdial, M., "Paving the way for computational thinking," *Communications of the ACM* **51** (8), pp. 25-27 (Aug., 2008). `http://doi.acm.org/10.1145/1378704.1378713`.
8. IAE-pedia, Computational Thinking. `http://iae-pedia.org/Computational_Thinking` (retrieved 20 January 2010).
9. National Science Foundation, *CISE-CNS-CPATH FAQ.* `http://www.nsf.gov/cise/funding/cpath_faq.jsp` (retrieved 31 Jan. 2010).
10. Udell, Jon, *Talking with Joan Peckham about computational thinking.* `http://blog.jonudell.net/2009/05/04/talking-with-joan-peckham-about-computational-thinking` (retrieved 31 Jan. 2010).
11. Wing, J. M., "Computational thinking," *Communications of the ACM* **49**(3), pp. 33-35 (April, 2006). `http://doi.acm.org/10.1145/1118178.1118215`.