EDUCATION:
FUTURE FRONTIERS

OCCASIONAL PAPER SERIES

A Conversation about Computational Thinking

Jeanette M Wing

A paper commissioned by the NSW Department of Education

ABOUT THE AUTHOR

Jeannette M. Wing is Avanessians Director of the Data Science Institute and Professor of Computer Science at Columbia University. She has been Corporate Vice President, Microsoft Research; Department Head of Computer Science, Carnegie Mellon University; and Assistant Director of Computer and Information Science and Engineering, National Science Foundation.

EDUCATION: FUTURE FRONTIERS is an initiative of the NSW Department of Education exploring the implications of developments in AI and automation for education. As part of the Education: Future Frontiers Occasional Paper series, the Department has commissioned essays by distinguished authors to stimulate debate and discussion about AI, education and 21st century skill needs. The views expressed in these essays are solely those of the authors.

The following is an edited conversation about computational thinking with Jeannette Wing.

What is computational thinking and why does it matter?

I defne computational thinking as the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can efectively carry it out.1

I believe that the skills one learns as a computer scientist are incredibly important for anyone working in any job in today's society. It does not matter what feld you study, what profession you pursue, or even in what sector you practise. I see this need in spades in industry. I'm also seeing that many colleges and universities around the world have embraced this belief and realised that the job opportunities for their graduates demand computational thinking. It's more than programming skills that employers are asking of their employees. Ten years ago it might have been a harder argument to make, but now it's a given. Anyone who graduates knowing computational thinking or with the skills of a computer scientist will have an advantage over those who don't and they will be more competitive in the job market.

Computational thinking is sometimes equated with coding or programming. How can the 'computational thinking = programming' trap be avoided?

Computational thinking is more conceptual than programming. In my defnition, I deliberately use the terms 'thought processes' for formulating a problem and expressing a solution—it's what you do in your head. Programming is a way to make the solution concrete so that it can be run on a computer that is a physical machine. So computational thinking frst and foremost is what humans do. Programming is an expression of a solution that a machine can understand. Of course, when you are programming you are using computational thinking, but the opposite is not true: you can be doing computational thinking and not be programming at all.

You have promoted computational thinking for over a decade now.Are you surprised at how infuential computational thinking has become in education?

I'm not surprised it has become infuential in higher education. When I was at the National Science Foundation ten years ago, I helped create a program called Cyber Enabled Discovery and Innovation. It was all about computational thinking for scientists and engineers. So even ten years ago, it was already a given that computing was going to be necessary for conducting research in any science and engineering feld. This recognition meant that graduate students were going to have to learn computational thinking regardless of what feld they studied. Also, ten years ago, for undergraduates I was promoting the idea that introductory computer science courses should focus more on the higher-level concepts of computer science rather than focus primarily on learning a particular programming language or only learning how to write code. That idea was already in the air so I'm not surprised that computational thinking has taken over at the undergraduate level. Now such courses are the most popular on many campuses.

I am surprised at the pace at which we have made inroads at the K-12 level. I need to thank the advisory committee I had while I was at the National Science Foundation for encouraging me to look at K-12, especially early grade levels. While I was promoting computational thinking across the foundation, the advisory committee asked, 'Why don't you tackle K-12?', and I said, 'You've got to be kidding! I know nothing about K-12 education'. Moreover, in the US, doing anything in the K-12 space is a huge undertaking. One reason is that K-12 is extremely decentralised in the US. There are 10000 school districts and to efect any kind of national change you have to go to each district one by one. I didn't fathom tackling that challenge! However, being at the National Science Foundation, I did have a national platform; moreover, the foundation has a directorate focused on education. Thus, I was able to leverage my position at the National Science Foundation in ways that did move the needle.

Specifcally, the lever we used was advanced placement courses, which are college-level courses taken by high school students in order to get college credit. We worked with the Educational Testing Service and the academic community to create a new advanced placement course in computer science. We started to promote this new course as one which high schools should ofer—in addition to the existing course, which was primarily focused on programming. At the same time, colleges and universities were changing their frst-year and introductory computer science courses. By ensuring that the curriculum of the new advanced placement course matched the new college-level curricula, we could efect a change across K-12 in the US in a scalable way. It was an alignment of stars and perfect timing. Exploiting this lever made a dramatic diference above and below.

But to be honest, the real credit for a lot of what was happening at the K-12 level is due to the entire computing community working with educators, especially teachers in high school and elementary school, and even with the Department of Education.

Meanwhile, computing technology continued to pervade our everyday lives. Young children took technology for granted and were growing up more tech-savvy than their parents. People recognised the importance of having K-12 students learn computing skills. At the same time, companies in the IT industry, such as Microsoft, Facebook, Google, Apple and so on, were desperately trying to hire people with computing skills. The demand far outweighed the supply. These companies realised they needed to look one level earlier in the pipeline and to encourage more students to take computer science in high school. The huge demand for talent by industry helped drive the awareness of computer science education at the K-12 level.

When I frst started talking about computer science at the K-12 level, I said that there are two very fundamental questions that need further research by the education community. The frst is, what are the concepts to teach and when? My analogy is mathematics, where we fgured out that by the time you are fve years old you have enough mathematical sophistication to understand numbers and relations, such as greater than and less than; by the time you are twelve years old, you have the mathematical sophistication to learn algebra; and by the time you are eighteen years old, you have the mathematical sophistication to learn calculus. Somehow we have learned from teaching mathematics for centuries and studying mathematics education both how the brain develops and gains the sophistication to do mathematical reasoning, and how can we align the teaching of mathematical concepts to that growth in reasoning capability.

So, ten years ago, my question to the computer science community working with educators was 'What is the analogy in computer science?'. This question had never been asked before. I strongly believe it's important to do research to fgure this out. In the beginning, I was pretty adamant that we should understand the science underlying how to teach computer science to young children—to do the research—before we go out and invent a lot of curricula that are not grounded in science. But there was so much momentum around me that people just went out and started inventing curricula. Fortunately, the education community is pursuing this line of research now. Also, new technology, such as massive online learning, enables us to do large-scale experimentation as part of the needed research in education.

There defnitely is a lot we don't know that will take time to fgure out. Analogously, we still have what we call in the US 'maths wars', where we continue to tinker with teaching mathematics in K-12. I anticipate that, decades from now, we will still be trying to fgure out how best to teach computer science to K-12 students.

The UK, through their Computing At School initiative, has introduced computing at all levels. It is a very courageous efort. The UK is my exemplar. I hope countries around the globe look to the UK as a leader and learn from them as they push the frontiers of education in computer science.

The second fundamental question is how best and when should we use 'the computer' in the classroom to teach and reinforce computational thinking concepts? Here my concern is throwing technology into the classroom and thinking the students are going to learn anything, let alone computing. We need further research on how computing technology can be used efectively for learning and not hinder the learning process. We also need research on how such technology can help reinforce the learning of computational thinking specifcally.

Some commentators have argued that computational thinking mainly benefts students in statistical or scientifc environments, and that the benefts of computational thinking in other disciplines such as creative arts or humanities have not been empirically substantiated. Do you have thoughts on this?

It's a fair statement to say the benefits of computational thinking in arts, humanities and social sciences have not been 'empirically substantiated', primarily because it's too early to tell—only now are researchers exploring the power of computation in these subjects. However, when I look at felds such as economics and social science specifcally, and even the humanities, computational methods are transforming these felds. New programs around the country and around the world recognise the prevalence and importance of the digitisation of data. With the help of computational power, you can do a lot with digitised data that you couldn't do as a human being. And so the digitisation of data is bringing computational methods to all felds where you can search, manipulate, analyse and visualise the data. These methods will enable us to make new discoveries, to fnd patterns and to suggest new questions that people would never have thought to ask before.

For example at Columbia University we have a history professor who has been looking at massive amounts of declassifed government documents and analysing them in new ways. By using computational methods and tools, he is able to make new discoveries about law, policy and history. As a human being, you could not make these discoveries on your own because you could not read all the data, you could not digest all the data, you could not remember everything you've looked at, and so, you could not fnd specifc patterns across all those documents. And that's just an easy example. At Columbia and elsewhere, people in all felds are recognising the value of data to making new discoveries and making predictions. I was just talking to a colleague in Economics this morning and he was rattling of many examples of his work with data, all of which have important implications for economic policy, decision-making and prediction. We are at the tip of an iceberg considering all the data that is being digitised and people in all felds now having access to online datasets that didn't exist before.

More specifc to the creative arts is the ability to use technology to digitise artefacts, media and structures. Here I'm talking about emerging felds such as digital art, digital humanities and digital archaeology. For example, we can digitise historic relics—what you might see in museums—and then provide anyone around the world access to explore these artefacts. You don't have to travel to a remote place to enjoy the beauty and culture of other regions around the world. It's a diferent kind of globalisation if you like—it's one way to bring diferent cultures together through shared access of digital data.

Finally, I would like to add that computational thinking is itself a very creative process. As with any problem-solving, it relies on human ingenuity, fashes of insight and taste in design.

You touched on this earlier when we were talking about the K-12 computational thinking concepts. One of the challenges is how can it be measured or assessed, particularly in non-computing disciplines.

What do you see as the way forward on this?

Any educator would ask this standard question: How do we measure or assess whether one has learned a concept or not? Early on, I encouraged computer scientists to work with education, learning and cognitive scientists to fgure out answers to this question. When I teach college students, I know how I might test a particular concept such as whether someone can write and analyse an algorithm, or whether someone can look at code and argue whether it does the right thing. There are various ways to test and measure the understanding of computational concepts. The bigger picture is still up in the air: How do we measure and assess at the K-12 level?

That's why, as much as I am very excited to see the progress we have made in the K-12 space, we need to temper our enthusiasm because we are still exploring and experimenting. We really do not know when is the right age to teach what concept or what is the degree of reasoning capability a child needs to learn a given concept. I don't have good answers to these questions, but as long as the education and computer scientists are working together, we will make progress.

What are your thoughts on the growing use of and interest in AI and data science?

The progress we are witnessing today in AI is due to the convergence of 'big data' and 'big compute'. What do I mean by that? The AI-based algorithms that people use routinely today in industry are successful because they can be fed with lots and lots of data, so that's the 'big data' concept. The second part is that these AI-based algorithms are compute hogs, meaning that they take lots and lots of processing capability that is best run in the cloud. The cloud provides huge numbers of servers, including huge numbers of central processing units, graphical processing units and other kinds of specialised processors. AI is successful today because algorithms can be fed with lots of data and can be run on these huge computing clusters.

Advances in AI today come from having data. Thus, in terms of the future, data science is even more fundamental to society's digital transformation than just AI. The amount of data we produce continues to grow exponentially. Since we are going to be generating more and more data, we will be analysing more and more data. More data will certainly empower AI to be more sophisticated and more capable. This trend is not going to end, and so we need to adapt to it.

We also need to think about the consequences and implications of more and more of our world being driven by AI-based software. This world is very diferent from the world of today or yesterday where we had software all around but it was designed to be as predictable as possible. For AI-based algorithms the answers are probabilistic. A prediction or classifcation by an algorithm is made with some associated probability, leaving room for uncertainty. Thus, given the output of these AI-based algorithms, any decision you make or action you take is based on likelihoods. Probabilistic reasoning is very diferent from purely logical reasoning, the basis of traditional computing: 0s and 1s, on or of, right or wrong, yes or no.

We need to embrace uncertainty. There is uncertainty everywhere. There is uncertainty in datasets: they can have missing, imprecise or inaccurate values; they can have noise. Mother nature is unpredictable, the physical world is unpredictable and humans are unpredictable. Yet our software systems are going to have to operate in these unpredictable environments and interact with each other and with us humans. The way that we embrace uncertainty in computer science is to use probabilistic reasoning. Probabilistic and statistical reasoning underlies all modern machine-learning techniques and tools. Since these technologies are not going away, we need to consider what needs to be taught in school. We should emphasise not just discrete mathematics but also probability and statistics. Expecting knowledge in these subjects has implications in terms of school education.

In a 2006 article you wrote,'Computational thinking is a way humans solve problems; it is not trying to get humans to think like computers'.2 Eleven years later, with the rapid development of AI, it seems we are getting closer to making computers think like humans. Is it likely that computers will soon do computational thinking better than humans; for example, self-coding AI?

It's a great question and the whole idea of self-coding AI is a new, active area of research. It helps to distinguish between the AI we can do today and the holy grail of AI. In a 1965 conference at Dartmouth, very prominent computer scientists got together and founded the whole area of AI. Their vision was to build a machine that could mimic human intelligence. This vision is the holy grail. Very early on, however, they realised that the general AI goal was way too big a problem to tackle. Instead, the research community divided the intelligence of humans into subcategories: speech, vision, language, planning, decision-making, mobility (e.g., walking or manipulation; for instance, with your fngers) etc. Each of those subcategories then became its own big feld within computer science.

It was only in the early 2000s that all of these separate strands of AI started coming together because many of them were using common techniques, specifcally machine learning. If you use the same technique for vision as you do for speech, as you do for natural language processing, as you do for machine translation, as you do for robotics, then all of a sudden there is something quite tantalising in thinking we can go after the 'general AI problem'.

To be honest, solving general AI is really far of, if you look at what we can do with today's AI. We can train a machine to process images to recognise objects; it's a human-level task, but it is just a single task that humans happen to be good at. We can also use loads of data and compute power to train a model that can recognise English speech; it's a human-level task, but again it is just a single task that a human can do. We cannot build a machine today that can do all of the things that a human can do all at once. We can build little machines, each of which can do a single task that humans are good at. So we are far from solving the general AI problem.

Even so, some machines are as good as humans at performing some tasks, such as object recognition or speech recognition. Some, such as the Go computer program that beats human Go players, are even better. But most of our current AI machines or agents are still worse than humans. So we don't have general AI yet, and even most human-level tasks that we are nailing today with machines are still not as well done as by humans. In short, we have a long way to go before we have anything resembling a machine that has the general intelligence of humans.

To focus specifcally on self-coding AI, there is defnitely interesting research going on at Microsoft Research and other places, where people are using AI techniques such as machine learning, and deep learning specifcally, to synthesise code and programs. Once we can succeed at this task, an interesting question is whether these AI agents will replace programmers as we know them today. I think replacing programmers is a ways of because current research is barely scratching the surface, though the results show feasibility. Even so, the task of programming is only one small part of software engineering, what is practised in industry. Much individual human thought, human-to-human communication and teamwork are needed to build large software systems. I don't see software engineering jobs being replaced anytime soon.

You asked me about whether computers could do computational thinking better than humans. Given that computational thinking is really about tapping into the creativity of humans to understand problems and express solutions so that a computer can carry them out, I don't think we are there yet. Perhaps what you are really asking is: Can these AI agents think creatively? It's hard to do technically. More difcult is to defne what creativity is, let alone measure it.

Accenture released fndings from a global study earlier this year outlining the potential jobs that could be created by AI.3 It highlighted trainers, explainers and sustainers of AI. Do you think education systems are focused enough on developing the computational thinking that students will need for the jobs of the future which will require them to work alongside machines?

This question needs to be unpacked because there are a lot of questions within it. First of all, do I think that education systems are focused enough on developing computational thinking? As we discussed, more and more countries are looking at their K-12 education and trying to promote the teaching of computer science. This transformation will happen over time because of demand and because these skills are teachable to K-12 students.

About jobs of the future, it is true that advances in AI are going to automate some jobs that today are done by humans—no question. Technology has always caused the loss of some jobs, but it has also created new kinds of jobs. We should be thinking about what those new jobs might be and what are the skills we need to teach children today or retrain current workers to learn so that they can do these new jobs. A relevant economic and societal concern is that as automation takes over a job previously done by a human, the person who no longer has a job may not have the new skills for the new jobs or have the desire to learn the new skills needed. It's important for society to prepare students properly for the new jobs that will emerge, and also to think carefully about how to encourage and help people who have lost their jobs to automation to learn new skills.

The third part of the question has to do with humans working alongside machines. Machines are never going to replace humans completely, but more and more humans are going to have to work alongside smarter and more capable machines. For them to work efectively together, humans and machines will need to communicate at a higher level of discourse than they do today. Right now, machines produce answers, perhaps probabilistic, that a human needs to interpret and then make a decision or take some action. If the human doesn't understand how to properly interpret the answer the machine produces, then something can go wrong. Similarly, the way in which humans communicate with machines requires either simple spoken commands or low-level instructions written in a machine-interpretable language. Raising the level of communication between humans and machines is a research problem.

Another emerging phenomenon is the combination of humans and machines that can solve problems that neither can solve alone. This combination requires humans and machines to understand what each other can and cannot do and to understand what each other knows and does not know. A nice example of this combination is a kind of robot called CoBot, which a colleague of mine at Carnegie Mellon University built. It's called a CoBot because the robot knows what it doesn't know, and when it needs help, it turns to the human and asks for help. Specifcally, this CoBot can roam the hallways, deliver water and mail, and escort visitors to their host. But when it gets to an elevator door, since it doesn't have hands, it needs help from a human to press the elevator button. So it stops and turns its cute robot head to the human alongside it and says, 'Would you please press the elevator button?'. The elevator opens and the CoBot walks into it. And then someone has to push the foor button. This kind of interaction that the CoBot has with a human shows that the robot knows what it doesn't know, and when it needs help it asks the human.

Notes

1. Defnition with input from Al Aho at Columbia University, Jan Cuny at the National Science Foundation and Larry Snyder at the University of Washington. For further information, see: Wing, Jeannette M (2014). Computational Thinking Benefits Society. Social Issues in Computing, 40th Anniversary Blog, 10 January. http://socialissues.cs.toronto.edu/index.html%3Fp=279.html

2. Wing, JM (2006) Computational Thinking. Communications of the ACM, 49 (3): 33–5.

3. Wilson, HJ, P Daugherty and N Bianzino. The Jobs that Artifcial Intelligence Will Create. MIT Sloan Management Review, 58 (4): 14–17.

Education: Future Frontiers | Occasional Paper Series