Computational thinking
is more about humans than computers

TIM BELL WITH JOSIE ROBERTS

Set interviews computer scientist Professor Tim Bell to figure out how computational thinking differs from digital literacy, and why both might be important for today's society. Tim explains his mission to introduce teachers and students to computational thinking, even without a computer in sight. His work with schools—from junior primary to senior secondary—shows that computational thinking augments a range of learning areas and competencies.

Q. Are there important differences between digital literacy and computational thinking?

That's a big question!

When I was visiting New York City a few years ago, a friend very kindly shouted me tickets to a jazz concert; all I had to do was pick up the tickets from the ticket booth. When I got to the concert about 20 minutes early, there was a queue going out the door. By the time I got to the front of the queue I was a few minutes late for the concert, and surrounded by agitated customers! I hastily asked, "Do you have some tickets for Bell?" The attendant asked me what time I made the reservation, which I didn't know—even if I had booked them myself I probably wouldn't have remembered! She rolled her eyes and gave me a "yet another one" look; she had the tickets in a box in the order that people phoned in. With a sigh, she started at the back of the pile, and went through every ticket until she found mine. As I hurried into the concert, I could hear her sighing because the person behind me also couldn't remember the exact time they booked their tickets.

This is a somewhat stark example of not using good computational thinking—perhaps putting the tickets into alphabetical order would have saved a lot of time? But how do you quickly put a few hundred envelopes into alphabetical order? And how much time would it save? Seconds? Minutes? Hours? These are the questions a computational thinker would be asking.

Computational thinking has emerged as a useful toolkit for dealing with problems where the solution is a *process,* rather than a product. It can be applied to all sorts of situations, not necessarily involving a computer. When you make a computational process happen on a computer, that's called *programming*, but if you don't start with good computational thinking, you can end up with a slow app that frustrates the users because it's doing things the wrong way. Many people have come across systems that can take way too long to respond because of some inefficiency in how they were designed, or don't give effective access to the information that you know is there.

While computational thinking isn't directly about programming, when you write a program it provides

http://dx.doi.org/10.18296/set.0030

a thorough test of your computational thinking—the computer is completely unforgiving and will follow your set of instructions exactly, so students receive instant feedback if their computational thinking is sound. For some students this is frustrating, and for others it's liberating! And so we end up with the strange connection where computational thinking (and the closely related field of computer science) are not particularly about programming, yet programming can be a key focus for computational thinking.

Digital literacy, on the other hand, focuses on *using* a digital system effectively. This is also important, since digital devices have become such ubiquitous tools, but it generally treats the software or app as the starting point, whereas computational thinking explores how such apps could be designed. Not understanding the difference between students being a *user* (digital literacy) and *creator* has delayed the introduction of computational thinking into curricula, and officials find these hard to distinguish, believing that introducing devices will automatically support computational thinking. Having BYOD [bring your own device] and high-speed internet is about digital literacy, which can be applied across many subjects, but computational thinking is quite different. Overseas the introduction of BYOD has even hindered getting students involved in activities such as programming; just when schools have started to teach programming (which requires large screens and the ability to run new programs), administrators have removed computer labs and replaced them with locked down tablets with small screens!

Q. Why might teachers want to consider computational thinking?

The big picture is that society is becoming increasingly digital, and much of what happens for us as humans is based around computation, whether it is communicating with others, transport, shopping, financial transactions, or entertainment. Issues like privacy and security develop a new dimension in a digital context. Instead of just being a user at the mercy of those developing the systems, students can start to understand what is happening, and even have a hand in making it happen.

We've been running pilot programmes in local primary schools looking at what works and what doesn't for introducing computational thinking and programming into New Zealand schools. The interest in the programme has been snowballing, and the great thing about working with primary school teachers is that they already have much of the background needed to engage with teaching computational thinking. Teachers in the pilot have been embracing the new topics, not just because they are seeing a high level of engagement with students, but because of the surprisingly strong cross-curricula benefits. Students have demanded to learn concepts from geometry in order to move objects around on the screen, they have found connections to health and PE, and they have developed their overall literacy as they communicate with others about what they want to do, or have achieved.

We've also found that many teachers who approached this topic very nervously (with no previous experience) have found it exciting; they can understand the concepts if given an appropriate introduction, and they find their students very engaged as they explore how to create digital systems rather than just use them.

Q. Why is it important for students to learn how to think like a computer? Isn't it enough for them to be able to use one?

Computational thinking isn't about thinking like a computer; it's about getting control over digital devices by understanding them. This requires a higher order of thinking and reasoning than a computer can do, and a different kind of reasoning to what we are used to in the physical world. For example, computer programming isn't about just writing the correct "code". It involves finding out what you want to write, testing it, and debugging it (tracking down the part of the program that isn't doing what you intended). All this is in a digital domain where there are no physical objects to observe, but virtual objects can be created at a whim, including scaffolding to help you develop your own program, the opportunity to use automation to reach to the other side of the world in a fraction of a second and collect information, and easy mechanisms to distribute millions of copies of a program internationally with a few minutes' work! This is quite a different view of the world to physical systems that students usually interact with, even though digital devices have become a huge part of their physical world.

By gaining mastery over the basic ideas of digital systems, students are empowered to understand the digital world in which they live. It is possible for relatively young students to grapple with ideas like encryption, which affects our privacy—if a wireless laptop is transmitting all the data it is sending to the internet, how could it possibly be private? They can explore the limits of computation—could a computer ever program itself? Are there things that we might think are possible to do with a computer, but actually aren't? Are there things we could do with computation, but shouldn't?

In the same way that students need to understand some science to form a view on climate change, or they need to understand social and cultural issues to form a view on politics and conflicts, they need to know some

basics of the concepts underlying digital technologies to make reasonable decisions about the digital systems that interact with almost every move we make.

The opposite of this is "screen essentialism"—the idea that what's on the screen is the whole thing, and we take anything behind the screen for granted. Computational thinking gets students to look behind the screen at what is really happening, and empowers them to know that they can influence it, and even create things behind the screen for themselves.

Q. Can you tell us about the Computer Science Unplugged (CSU) resource for teachers and what led you to develop it? What do you mean by the trailer tag-line "computer science is no more about computers than astronomy is about telescopes"?

CS Unplugged (csunplugged.org) started over 20 years ago when my son's J1 class invited parents to talk about their jobs. I was at a loss for how to present computer science to 5 and 6 year olds, particularly when the previous talks had been from a policeman with a police car, and a nurse with fake blood and bandages. So I made the radical decision to not use a computer, and developed some games and a magic trick to get across the *concepts* that I worked with, rather than the end result (which would be a fast, easy to use, secure, reliable computer program).

It turned out to be engaging for the students, and reinforced other curriculum areas, and I was subsequently invited back to try it with other classes. Soon after that I came across Mike Fellows, in Canada, who was doing something similar, and we pooled our ideas, releasing them as "CS Unplugged" (it was the early 1990s, and Eric Clapton's *Unplugged* album had just been released).

Since then, CS Unplugged has been used all around the world, translated into about 20 languages, and has had a renaissance in countries where computer science or computational thinking are part of the junior curriculum. Teachers have found it empowering: they already know how to work with cards, string and chalk, and how to teach young children, so it provides the glue for them to do something without having to worry about digital devices crashing or being incompatible with the school system. Of course, we don't advocate it as a complete computational curriculum, but it's a very useful component that gets students away from their screens and thinking about key concepts. The idea of "computational thinking" became popular about a decade ago (through an influential article in 2006 by Jeanette Wing), and as that became popular, we realised that the Unplugged approach had captured much of the essence of computational thinking. The relationship between computer science and computational thinking is very intricate, and could be the subject of a whole article, but to simplify things, at a primary school level these concepts largely converge, so we had developed an approach to computational thinking before it became a buzzword.

The phrase "computer science is no more about computers than astronomy is about telescopes" was coined by Mike Fellows (although if you Google it, you'll have to do a thorough job to establish who said it first, as it was later used by Dijkstra, but that's another story.) Mike also drew analogies with chemistry being about test tubes, and biology about microscopes. These disciplines aren't defined by their key tools, but by the great ideas behind them. Many of the key ideas in computer science existed before computers did; for example, the main logic that is the basis of all digital computers is Boolean algebra, developed by George Boole, who was born 201 years ago. The word *algorithms* is derived from the name of a 9th-century author, Muḥammad ibn Mūsā al-Khwārizmī.

To follow the analogy, astronomers seem to use telescopes a lot, and spend a lot of money on them, but usually it's not because they're interested in telescopes. In computer science, we also use computers a lot, but we're more interested in what we can make them do, and also what we *can't* do with them. In fact, if computer science is about anything, it's about *humans*: how do we develop software and apps that don't keep people waiting? That don't flatten your smartphone battery by doing unnecessary calculations? That can operate and store data in the limited space of a wristband? That will be reliable even if the hardware isn't? That have an interface that matches the way that people think? Just as "telescope scientists" are more focused on the stars, computer scientists are more interested in what a digital device can do for humans (and in fact, some of those who are more interested in the device than humans have been responsible for some pretty annoying systems that are far from "user friendly"!)

Q. Is computational thinking just another educational fad? How does computational thinking and the CS Unplugged resource interplay with the New Zealand Curriculum and its learning areas?

While there are good arguments for teaching computational thinking in its own right, I think its longevity rests on whether systems based on computation (i.e., digital technologies) are a fad, or are likely to continue to permeate society, *and* whether or not schools should prepare students to be informed citizens in a democratic society. If digital systems are a passing fad, it's not clear what they'd be replaced with (sure, quantum

computing might be a thing, but that's still based on computational thinking), or one could imagine a future where digital systems have collapsed and we revert to 19th-century technology (there are already people who would advocate this!) In the meantime, we live in a society that is increasingly controlled by digital systems, whether it is the hundreds of computers in your car, the digital mobile phone system, your bank account, or your online purchases. Your access to services and right to privacy are heavily dependent on the digital world. I also hope that we will continue to be a society in which the education system prepares students to be informed citizens!

As a topic in schools, there's the concern that it might push out other important learning areas, but our experience is that relatively little extra time is needed because it exercises other areas of the curriculum. There are obvious connections to numeracy and literacy, but we are also finding meaningful ways to have computational thinking reinforce areas as diverse as music, and health and PE (bear in mind that music and fitness are now very digital—how often do you see someone carrying a music player or wearing a fitness tracker, or both at the same time!?) So the relationship of computational thinking to the curriculum is a bit like that of maths or English; you could argue that these needn't be taught in their own right because they would be used by other subjects anyway, but at some point you need to acknowledge that there are some valuable concepts that students might not encounter by chance, and ensure that they are covered. In reality, integrating topics is a great way to teach them, and real problems will draw on many disciplines, as well as exercising the key competencies.

Q. What exciting developments have you witnessed in New Zealand primary and secondary schools, and where do you see things heading next?

New Zealand was a very early adopter of computer science as a formal high school topic (as far as I know, it was the first English-speaking country to do so), in the form of NCEA achievement standards starting in 2011. This generated a lot of interest around the world, and since then we have seen the United Kingdom and Australia adopt forms of this, not only at high school level, but in the last couple of years as compulsory primary school topics. The process in New Zealand has been a grassroots movement, as teachers have embraced the new opportunities (many of them had been holding out for something like this); however, not all schools have adopted the standards, and often management, parents and students haven't fully understood what it is about, leading to mismatches in resourcing for PD or students taking on computer science without an adequate background. Despite this, New Zealand now has hundreds of teachers who have upskilled in this area (mainly thanks to sponsorship from industry, who are motivated by the severe lack of graduates in this area, including concerns about diversity).

At the university level we have seen increases in the quantity and quality of students arriving, but most importantly, an increase in diversity. Traditionally very few women have taken computer science, and yet they often do better than men both in employment, and in academic results (girls have done better than boys in several key computer science NCEA standards). If computer science is ultimately about people, then we need developers who represent the diverse range of people that will be using the software/apps being produced, and it's heartening to see things moving in that direction, although there's a long way to go yet.

We know that to really influence diversity, it's important to give students opportunities to find out what the subject is before they reach their adolescent years, where decisions might be influenced more by social pressure and less by what they are actually good at. Introducing computational thinking or computer science into primary schools is taking off around the world. Here in Christchurch we're in our third year of running formal pilots in local primary schools, working with typical teachers and typical students (a lot of previous work has been done with self-selected clubs or special events).

We are finding that the teachers in the pilot are embracing the new material. Many have reported that through computational thinking activities they are also teaching other curriculum areas, and hence the impact on teaching time is relatively low. We have observed that because teachers have been teaching computer science and programming, they now are integrating this into their inquiry units because programming gives students an open opportunity to demonstrate their high order thinking in any subject by creating new artefacts such as quizzes or animations to demonstrate their learning as examples. Some teachers have observed that students who were previously disengaged with their learning are drawn to the computational thinking exercises because they are using materials and movement to solve problems. This isn't about using e-learning tools, but doing computational thinking, and through it exercising numeracy and literacy, and other topics, including physical education (e.g., by writing and testing software for a beep test).

Overall, the value of computational thinking for students isn't just about particular skills and knowledge that they might pick up, but finding out if this is something they are good at, and appreciating what

the supporting skills are. For example, it's very hard to explain how maths is crucial to computing (often students or parents see maths as just arithmetic, which of course the computer can do for you), but when you've done programming you can appreciate that you need to learn how to work accurately with symbols and apply reasoning to formal systems, and use geometric ideas to create great graphics.

So while it's great to be able to *use* digital devices, understanding them and creating *new* digital systems is empowering for students (and great for our economy). Since I started with an example of a lack of computational thinking working *against* me enjoying a performance, can I give a link to some recent Oscar presentations where the presenters acknowledged how these skills benefit the creative world? One of our ex-students, with several others working at Weta Digital in Wellington, recently won an Oscar. It wasn't for acting, but for developing the software that provided the innovative graphics for the film *Avatar* (see https://www.youtube.com/watch?v=C54bEFUXBnc). It's one thing to know how to *use* software, but it's getting beyond just what's on the screen that gets international attention.

Reference

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

The next article in this issue of *set* explores how to teach coding, a specific activity associated with computational thinking (see Falloon et al., p. 8). The team show that coding is achievable for very young students and helps them to build a range of general and higher order thinking skills.

Tim Bell is a professor in computer science at the University of Canterbury. His Computer Science Unplugged project (csunplugged.org) for students of all ages is widely used internationally, and its books and videos have been translated into about 20 languages. Its sister project, the CS Field Guide (csfieldguide.org.nz) presents computer science for high school students. Tim's work is widely published and he has received national and international awards in computer science education. He is also a qualified musician, and performs regularly on instruments that have black-and-white keyboards.