What's all the fuss about coding?

Professor Tim Bell
University of Canterbury, New Zealand

Professor Tim Bell is a professor in the Department of Computer Science and Software Engineering at the University of Canterbury in Christchurch, New Zealand. His main current research interest is computer science education; in the past he has also worked on computers and music, and data compression. His Computer Science Unplugged project is widely used internationally, and its books and videos have been translated into more than 18 languages. He has received many awards for his work in education, including the ETH Zurich ABZ International honorary medal for fundamental contributions in computer science education in 2013, and the Institute of Information Technology Professionals New Zealand Excellence in IT Education award and President's Award for Contribution to the IT Profession in 2014. Recently he has been actively involved in the design and deployment of new computer science curriculum in New Zealand high schools.

Abstract

The idea of teaching 'coding' to school students has become popular, and the term appears in the names of many initiatives, such as Hour of Code and Code Club. But what do we really mean by 'coding', and why would you want every child to learn it? Won't it be outdated soon? This paper looks at these issues, and why topics such as computer science are being taught to all students. This includes an assessment of misunderstandings around the idea of compulsory programming for every student, and the challenges that accompany the introduction of such topics into schools.

Introduction

The term 'coding' has become a catchword for an international movement to give school students the opportunity to explore technical computing topics. Using the word 'coding' gives an air of mystery to the topic, and this can be useful for attracting students' attention. In this paper we will unpack what is really meant by the term, and why it is being introduced into curricula around the world.

One of the drivers of exposing students to coding is to help them be *creators* of software, rather than just *users*. There are several motivations for this, but a key point is that being a mere 'user' in an increasingly digital world means that one is completely dependent on others to provide suitable software, which takes away individual freedom, since you can only consume what others choose to provide. Rushkoff uses the phrase 'program or be programmed' to capture this issue (Rushkoff, 2010). Lee et al. (2014) also highlight the sense of ownership that students get when they know how to modify and create technology. Furthermore, a country that doesn't produce and sell software is missing out on an important export market, which provides an economic incentive to increase the exposure to coding in schools.

Understanding the nexus of human life and the discipline of programming is essential; in the 21st century, computer programs (also referred to as apps or software) permeate daily life. Programs bring life to smartphones, provide access to information online, mediate much of human communication, run our transport, monitor our fitness, track our health, and protect our finances. Hence, computing is primarily about people, rather than computers. The computer is just the general-purpose tool we use to solve human issues, whether for something as noble as supporting democracy, or simply for pure entertainment in the form of games.

Not only do programs need to be written to address human needs, the process of writing programs involves an awareness of what those needs are. For all but the smallest projects, programming involves collaborating with others to deliver the software in a timely fashion; putting all this together explains why 'many skills of a professional programmer are related to social context rather than the technical one' (Blackwell, 2002). Coding, whatever it is, is more about people than about computers.

What is coding?

The term 'coding' has become widely used in recent years, largely through the names of websites that promote programming (for example, Code.org, Codecademy.com, Codeclub.org.uk). Coding has become a brand, relating to moving students from consuming digital technology to producing digital technology, and giving them a sense of agency.

Coding in popular culture has come to mean what is more accurately called programming, and, more generally, software development. The term 'coding' has traditionally referred to only a small part of the whole process of software development. Creating new software involves several aspects, including:

- analysis: identifying the needs for which a solution will be developed
- design: sketching how the solution will work
- coding: converting the proposed solution to a computer language
- testing: checking that the solution works as intended, including being reliable and usable
- debugging: tracking down why parts of it don't work as intended.

Those who advocate teaching 'coding' are invariably intending to refer to the broader ideas of software development listed above, but even this is a smaller part of the wider field of computer science. Programming is a key tool in computer science, but the bigger issues are knowing how to develop (rather than just use) fast algorithms, usable interfaces, intelligent systems, reliable networks, computer vision, innovative graphics software, and so on. New curricula appearing internationally take account of these broader issues, and in this context we can see that coding is simply a small but critical part of the whole idea of developing software to meet a human need. It has been compared to the telescope in astronomy; one could be forgiven for thinking that astronomy is about telescopes, since they are such a key tool, but that would be missing the point (Fellows, 1991).

While 'coding' has become common as a sound bite term to advocate for this new discipline, official curricula tend to use broader terminology. In the US, the term 'computer science' is more commonly used (for example, one of the main organisations is the Computer Science Teachers' Association). In the UK, 'computing' has been chosen. In Europe, the German term 'Informatik' (and various translations[1]) describes the field well, and in Australia and NZ, 'digital technologies' is the name of the new curriculum. A key point is that

1 Note that the English term 'informatics' doesn't have the same meaning as the European 'Informatik', and, confusingly, is closer to traditional curricula that are focused on using computers rather than developing new software.

all of them have moved away from very broad terms like 'information and communications technology' (ICT). A 2012 report from the Royal Society (UK) pointed out that a focus on learning to be a computer *user* rather than a *developer* 'has led to many people holding a very negative view of "ICT", to the extent that terminological reform and careful disaggregation is required.' (Furber, 2012). Traditional ICT in schools might be easier to teach, but is often focused on learning to use particular software, which means the knowledge could date rapidly. Of course, the new curricula don't throw out the baby with the bathwater; it's still important for students to learn to use existing systems effectively.

A concept that has become widely used to capture the idea of a more empowering computing curriculum is 'computational thinking' (CT). Rather than focus on particular technical skills, it captures ways of thinking that students should develop, such as decomposing large problems, designing algorithms, and abstracting concepts (Voogt et al., 2015; Wing, 2006). In principle, these concepts can be applied without even using a computer, but computer programming is a very direct way of exercising these ideas, and quickly exposes any weaknesses in a student's expression of how to solve a problem.

Why teach coding?

As discussed earlier, when popular culture talks about adding 'coding' to a curriculum, we should expand this to the general idea of computational thinking and the corresponding disciplines (for example, computer science or digital technologies). Guzdial (2015) gives several reasons that students benefit from learning computing.

- Jobs: For some students it will be important to discover early that this is in fact a rewarding career for them; at present, many students miss out on this opportunity simply because they don't know what it involves, and this has created a desperate shortage of suitably qualified software engineers. However, this shouldn't be the main driver; the goal is not to prepare all students for the software industry, in the same way that teaching art isn't intended to prepare all students to become artists.
- Learning about their world: Now that society is so digital-centric, we have created many issues such as risks involving privacy, security, artificial intelligence, intellectual property and computer reliability; but there are also positive opportunities such as access to information, efficiency gains and better communication. In the same way that some understanding of biology helps us to be informed about controversies such as genetic modification, understanding computing will help us be informed about drivers behind our changing society.
- Computational thinking: The skills learning through CT can generalise to non-computing problems that we face.
- Productivity: Understanding and being in control of the devices we use enables us to use them more effectively.
- Broadening participation: Women and minority ethnic groups are notably absent from the business of software development, and yet the industry is crying out for diversity in order to produce better products. Increased participation can largely be traced to stereotypes created by society that are very hard to overcome if a student hasn't tried the discipline for themselves. There is evidence that it is particularly helpful for students to gain experience in programming before their adolescent years (Duncan et al., 2014), which crudely translates to learning 'coding' in primary/elementary school.

Each of the above reasons have an impact on a student's self-efficacy: the idea that they can understand and even control or change their digital world is important, to avoid developing a society of technocrats and their users.

As noted earlier, programming isn't an end in itself. Programming is used to make the world a better place for humans (and understanding programming helps us to evaluate better if each program that is written actually does improve our world, be it a social network, encryption, or artificially intelligent system). This view is particularly important for engaging women in computer science; Margolis points out that 'for most women students, the technical aspects of computing are interesting, but the study of computer science is made meaningful by its connections to other fields' (Margolis & Fisher, 2003).

Much of what is already available in school curricula is foundational to computer science, and includes skills and dispositions around interpersonal communication, teamwork, mathematical reasoning, understanding society, and creative thinking. Introducing 'coding' to a curriculum should not push out these existing subjects, and, in particular, experience in areas like music has a positive impact on the ability of a student to function effectively in a creative team.

Of course, this raises the question of what might be removed from an overcrowded curriculum, but in our experience, adding computer science concepts to a primary classroom can help to teach other areas faster. For example, with students programming in Scratch, one of the initial exercises is often to draw a square, with 90-degree angles. Students soon want to find out how to draw other shapes, and end up demanding to know how to work out the angle for a three- or five-sided figure, and soon encounter the idea that a full turn is 360 degrees. We have seen this happen with a variety of topics; the mathematical links are more obvious (coordinate geometry, arithmetic, number representation and so on), but topics like interface evaluation require some concepts from psychology and sociology, and since output from a computer is sensed by human beings, this leads to considering how eyes and ears work (for example, red/green/blue cones in the eye explain the use of red/green/blue (RGB) colour models on computers; and the 20 kilohertz (kHz) limit of human hearing explains why 44.1kHz is a common audio sampling rate).

The challenge of introducing computer science

We are living through a digital revolution that has impacted almost every aspect of our lives. Many aspects of education have been through change in parallel with other changes in society; there is an increasing use of mobile devices, use of the internet to access information, and use of productivity software to improve the way we work with information. However, these are all significant changes in education, and although 'e-learning' has made a significant impact, it is primarily used to teach the same subjects that we would have taught without it, and teachers are mainly having to develop their pedagogical knowledge rather than their subject knowledge. In contrast, computer programming and related topics are (for most schools) a completely new curriculum subject, and will require considerable professional development for teachers to gain both content knowledge *and* pedagogical knowledge. This is often overlooked, or confused with e-learning; a school might mistakenly think that because students are now bringing their own devices for all classes, then they are learning computer science, whereas this often means the opposite and reinforces the notion of being a user rather than a creator.

Digital technology has had a huge impact on society, and introducing programming – while urgent and important – is a large transition for schools and staff. Relying on the idea that students have devices and that teachers can simply start teaching programming can lead to the initiative backfiring.

For example, computer programming in industry is generally done on large desktop machines with multiple screens. It is particularly unfortunate that programming is being introduced into schools at a time when computer labs are being removed, and students are getting devices with smaller screens! Furthermore, programming involves running completely untested programs on a computer (that is, the students' own programs), and with one-to-one devices, often school policies or even device manufacturers make it difficult to run such programs!

There is also an unfounded concern that these ideas might be too difficult for young students. This would be equivalent to saying we shouldn't teach maths, science or music based on how complex those topics are at an advanced level, when of course they need to be adapted to be age-appropriate so that a foundation can be built early. Engaging tools for teaching computer science have been developed for teaching programming and related subjects to primary-aged students. There are dozens of programming languages designed for children (Duncan, 2014). Students can also engage with many of the concepts of computing and computational thinking without using a computer; approaches like Computer Science Unplugged (Bell et al., 2012) can provide students with the opportunity to think deeply about issues in computing without having to learn to program first. Unplugged exercises aren't enough on their own – after all, students need to find out how programming actually works first-hand – but programming on its own isn't enough either, since it isn't an end in itself, but a tool for implementing new ideas.

Another issue is around the choice of a programming language to teach students. There are many factors to consider here, but a key point is that we should be teaching *programming*, not a particular language. The issue is similar to choosing a car for a student to learn to drive in; while the typical career expectations for a professional driver might involve a bus, truck or courier van, the first principles are easily learned in a small hatchback. Similarly, programming is best taught in languages that have good pedagogical support, including books or websites, and are motivating in an age-appropriate way.

Conclusion

Digital technologies now permeate our lives, and it is important that we grow a diverse generation of students who are empowered to understand what is really going on, are able to make informed decisions, and have the opportunity to pursue the remarkable career opportunities that we have. There are deep ideas that students need to understand that haven't been taught previously in schools. Fortunately there are age-appropriate ways of engaging students with these ideas, so long as we are clear about what the key concepts are, we are clear about our purpose in mandating that they be taught, and we resource the transition to teaching this engaging subject.

References

Bell, T., Rosamond, F. & Casey, N. (2012). Computer Science Unplugged and related projects in math and computer science popularization. In H. L. Bodlaender, R. Downey, F. V Fomin & D. Marx (Eds.). *The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday* (Vol. LNCS 7370, pp. 398–456). Heidelberg: Springer-Verlag; Berlin, Heidelberg. doi: dl.acm.org/citation.cfm?id=2344236

Blackwell, A. (2002). What is programming? In *14th Workshop of the Psychology of Programming Interest Group* (pp. 204–218).

Duncan, C., Bell, T. & Tanimoto, S. (2014). Should your 8-year-old learn coding? In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education – WiPSCE '14* (pp. 60–69). New York, New York: ACM Press. doi: 10.1145/2670757.2670774

Fellows, M. (1991). Computer SCIENCE in the Elementary Schools. In N. Fisher, H. Keynes & P. Wagreich (Eds.). *Proceedings of the Mathematicians and Education Reform Workshop, Seattle, 1991* (Vol. 3, pp. 143–163). Conference Board of the Mathematical Sciences.

Furber, S. (Ed.). (2012). *Shut down or restart? The way forward for computing in UK schools.* London: The Royal Society. http://royalsociety.org/education/policy/computing-in-schools/report

Guzdial, M. (2015). Learner-Centered Design of Computing Education: Research on Computing for Everyone. *Synthesis Lectures on Human-Centered Informatics, 8*(6), 1–165. doi: 10.2200/S00684ED1V01Y201511HCI033

Lee, I., Martin, F. & Apone, K. (2014). Integrating Computational Thinking Across the K–8 Curriculum. *ACM Inroads, 5*(4), 64–71. doi: 10.1145/2684721.2684736

Margolis, J. & Fisher, A. (2003). *Unlocking the clubhouse: Women in computing*. Cambridge, Massachusetts: MIT press.

Rushkoff, D. (2010). *Program or be programmed: Ten commands for a digital age*. New York: Or Books.

Voogt, J., Fisser, P., Good, J., Mishra, P. & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies, 20*(4), 715–728. doi: 10.1007/s10639-015-9412-6

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.