Creating the next generation of coders

JUL 20, 2022, MILES BERRY

My remarks at today's meeting of the APPG for Education Technology

The rationales for teaching programming to all

There are a number of rationales we might offer for teaching programming to everyone. I suspect the economic argument was uppermost in ministers' minds - to maintain a vibrant UK technology and software industry, we need to maintain a workforce of programmers. The software professions are largely meritocratic, quite well paid, and in desperate need of greater diversity. However, it's a weak argument to say that all should learn to program at school so that some can go on to get jobs in the industry - after all, we don't teach poetry to everyone just to maintain a supply of professional poets.

A more convincing argument is that learning to program helps develop thinking or problem solving skills - that it's a medium for teaching logical, systematic thinking. The particular sort of problem solving that looks for automatable solutions has come to be called 'computational thinking', and is the golden thread running through England's computing curriculum. That programming should be taught so as to develop thinking is similar to the arguments that used to be mustered for compulsory Latin, and there is evidence that, if programming is taught well, there is transfer into problem solving in some domains, notably other STEM disciplines. Furthermore, programming is itself a useful skill to acquire for anyone studying or working in numerate fields: it's not just software engineers who need to code, but also data scientists, accountants and so on. In many fields, there's a huge advantage to being able to automate processes through programming computers.

Another argument is that digital technology has now become such a fundamental part of our lives and our society that no child should leave school without, yes, being able to use this to accomplish goals, but also having at least some grasp of the principles on which it depends, of how it works and of how it is made, and this means at least some knowledge, and experience, of computer programming.

Thus programming should be taught not just because we need more coders, but because coding is a medium for creativity and problem solving, and because it provides a unique perspective on how the world and our society works.

A coherent curriculum

Soon after the coalition government took office back in 2010, Tim Oates of Cambridge Assessment, who went on to lead the national curriculum review, reported on how high performing jurisdictions approached curriculum design and implementation, arguing convincingly for a coherent approach, addressing standards, assessment, pedagogy,

teacher training, teaching materials and the incentives and drivers needed in the system. I'm not going to argue that we saw anything like this coherence with the implementation of the then new computing curriculum in 2014, but Tim's list offers a useful structure to look back on what we did, what we're doing, and what still needs to be done.

Standards

Unlike the core subjects of English, mathematics and science, the programmes of study for computing are brief - they take up three and half pages of A4. They start, with a callback to Marx's Theses on Feuerbach, with the noble ambition that a high quality computing education should

equip pupils to use computational thinking and creativity to understand and change the world.

I think it can be helpful to see this pairwise - computational thinking is about understanding the world, creativity is about making a positive difference to the world.

In framing the national curriculum, we were led in part by the work done by the Royal Society in its 'Shutdown or Restart' report, which saw computing as three interrelated areas: computer science, information technology and digital literacy. It's better to see these as the foundations, applications and implications of the subject. A broad, balanced computing education needs to encompass all three - an understanding of the underlying principles and big ideas of computer science, as well as repeated practical experience of applying these to solve problems through programming; skills in getting useful things done with technology, including the much bemoaned 'office' skills of word processing, spreadsheets and presentations, but also creative work across a range of digital media; and some critical consideration of the impact of digital technology - yes, this includes safety, responsibility and 'cyber', but needs to address societal as well as individual issues.

The programmes of study offer a good model of progression, particularly in coding and computational thinking, building on the foundations laid in 'creating and thinking critically' in EYFS. Key Stage 4 though is problematic - ministers made it clear that computing was to be compulsory for pupils aged 5-16, but here the programme of study is split between a minimal set of core requirements for all, and a vague requirement that pupils should also have 'the opportunity' to study IT and/or computer science at sufficient depth. Just under 80,000 sixteen year olds take up the opportunity to study computer science for GCSE - there seems very little computing provision for over 85% who do not.

Assessment

Formative assessment for computing is well supported - we've crowdsourced over 10,000 multiple choice questions, there's immediate feedback from the computer when pupils run the programs they write, and there's some, but perhaps not enough, use made of automated assessment, e.g. coding jigsaws ("Parsons's problems"), or unit testing to check that a program gives the answer it should.

At A-Level, independent projects count towards the grade awarded, with some pupils using this as an opportunity to develop their own machine learning applications. However, the A Level specifications seem to have been written with progression onto CS degrees at

university in mind, and thus they are rather narrower than that which a facilitating subject for any STEM discipline would be. Take up has improved greatly, but was still under 14,000 students last year, less than 15% of them girls. As far as I'm aware, no university requires A Level CS for degrees in CS - with such a small pool of potential candidates, why would they?

GCSE is even more problematic. For a start, the abolition of GCSE ICT had a significant negative impact on the numbers of pupils studying computing at this level, and skewed the gender balance of the subject - only about 20% of entrants are now girls. The exam specifications take a very narrow, and perhaps rather dull, view of computer science; assessed project work has been done away with, with the market leading exam board now assesses programming through its own, made up, exam reference language; and top grades are hard to obtain - pupils typically get about two thirds of a grade lower than they do in GCSE physics. There's a popular perception that GCSE computer science is hard, narrow and dull.

Alongside GCSE and A-Level there is a diverse set of qualifications in IT and digital media, although it's much harder to get a clear picture of how many pupils are engaged with these. The BTEC in computing has sat happily alongside A Levels, and has appealed to some as a more practical preparation for university or employment than A Level. The T Level in digital production, design and development has been well thought through, but this will be incompatible with A Levels.

Pedagogy

The best teaching in computing makes the subject accessible to all, recognises the full breadth and relevance of the subject, and makes this one of the most exciting subjects on the school curriculum. There is some phenomenal work happening in schools, with teachers making the most of the brevity of the national curriculum programmes of study to make computing engaging, creative and relevant.

Whilst longer established subjects have a broad evidence base on which to draw to inform teachers in how to teach, computing teachers are breaking new ground, finding out what works for themselves and readily sharing that with others. Computing teachers make excellent use of the tools available - Bee Bots and Scratch Jr make coding accessible to children who are still learning to read and write. MIT's Scratch provides a block based, multilingual environment for building, rather than writing, programs, as well as as a global community for inspiration, collaboration and support. The micro:bit and the associated makecode editor have made 'physical' computing possible in primary or lower secondary education. Online editors such as Replit, Jupyter Notebooks and p5.js provide an anytime - anywhere means for writing, and sharing, code, as well as tools for exploring data science and algorithmic art.

Much of the early work in the pedagogy of teaching programming was done by Seymour Papert using 'turtle' graphics and the Logo language, and I think his insight that people learn particularly effectively through making things to show to others remains as relevant today as it was in the 70s and 80s. The government funded National Centre for Computing Education has articulated a set of twelve, research based, principles for teaching computing, including reading code before writing code, collaboration and the use of worked examples - I know computing teachers have found this very helpful.

Teacher training

Back in the early days of implementing the computing curriculum, teachers' subject knowledge was a huge challenge - in primary, few teachers had done any programming, as they'd never been taught this at primary school themselves. Even in secondary schools, the sudden change from ICT to computing resulted in many teachers feeling disempowered and inadequate. Early initiatives were community focussed, as peer to peer support in Computing At School or through local authority based networks. In primary, Barefoot Computing, which is still going strong, created example lessons, linked to other curriculum topics, to illustrate the concepts and approaches of programming and computational thinking.

By the time of the Royal Society's After the Reboot report, it was apparent that the implementation of the computing curriculum was 'patchy and fragile', and a more top-down, properly funded approach was needed. Philip Hammond announced £84m in the 2017 budget to ensure that every secondary school would have a fully qualified GCSE computer science teacher; a year later the contract for the new National Centre for Computing Education was awarded to a consortium of STEM Learning, Raspberry Pi and the BCS. Nearly four years on, some 6,000 teachers have done at least ten hours of targeted GCSE training, and about 50,000 teachers have had some sort of support from the NCCE. Bids are currently being evaluated for a scaled back NCCE2, expected to carry this work forward in April next year. Ofsted's recent review of the research in computing education makes clear that computing teachers should continue to receive subject-specific professional development.

Recruitment to initial teacher training courses has been consistently below target since 2013, with the exception of a, probably COVID-related, peak in 2020-21. The drop out rate from computing teacher training courses is also higher than for other subjects. Not all CS graduates are cut out to be great teachers, but the pay doesn't help with recruitment - a good CS graduate has some far more lucrative career options than teaching. That said, teaching can offer much as a second career, for those who want to put something back, and have practical experience in the IT industry and correspondingly well-developed people skills, but even here the culture shock presented by many schools can be big.

Teaching materials

I've already mentioned some of the technologies used for teaching programming, but teaching materials matter too. As computing was largely a new subject on the curriculum, there were relatively few textbooks or schemes of work already available. Many teachers developed materials of their own, sharing these with their colleagues. The publishing industry stepped up too, and many primary schools found the example lesson plans in Barefoot a good starting point.

Alongside its training courses, the NCCE developed its own Teach Computing curriculum, with a well thought through, broad and detailed curriculum from 5 to 16, with pathways in Key Stage 4 for those taking GCSE CS and, importantly, those who were not. With the move to remote education in response to COVID-19, these lesson plans and resources were supplemented with video lessons from skilled teachers. As publicly funded content, this material is covered by an open government license for teachers to use and adapt freely.

For A-Level computer science, the NCCE, collaborating with Cambridge University, created Isaac Computer Science as an online teaching resource for students. More recently, GCSE content has been added to this platform. 3,500 teachers and 61,000 students have made use of this, and I think the quality of these materials is appreciated by all.

Incentives and drivers

The recent Ofsted research review for computing makes it clear that schools need to maintain a broad, balanced computing provision up to the end of Key Stage 4 - that the requirements of the Key Stage 4 programme of study need to be honoured in the observance, not the breach. They also argue that the subject needs sufficient time to do it justice at Key Stage 3 - given the ambitions of the programme of study, an hour a week is barely sufficient, and whilst the DfE's workforce census suggests things are getting better, the average allocation is around 52 minutes - in some schools, this will be much less. If we're serious about increasing the numbers taking computer science at GCSE and A Level, particularly amongst girls, and giving the subject the time it needs at Key Stage 3, then we'll need to retain and recruit more computing teachers, and what we've tried so far seems to have had little impact.

There is a role for ed tech in supporting the work of computing teachers - many are already making effective use of video tutorials (check out Daniel Shiffman's The Coding Train and Nottingham University's Computerphile to see how good these can be), interactive exercises, online communities like that for Scratch, and automating some aspects of assessment undoubtedly help, but none of these are a substitute for the interaction, responsiveness and motivation that a trained, knowledgeable and talented teacher offers.

So what next?

For me, the biggest stumbling blocks at the moment are qualifications which are no longer fit for purpose.

I'd replace GCSE Computer Science with a GCSE in computing, ensuring the qualification covers the foundations, applications and implications of the subject, just as the national curriculum does. I'd make sure this had a significant practical component, including both actual, not pretend, programming and digital media, and focussed on solving interesting, non-trivial problems. Can it really be right that after 11 years of learning computing, GCSE questions are no more engaging than calculating the amount of time people play a computer game, and saying if this is too much. I'd want the new computing GCSE to be one that, say, 80% of schools could expect 80% of their students to take. The subject is that important.

I'd also look seriously at A Level, reframing this to sit alongside A Level maths, not as preparation for a CS degree, although it would be, but as enough programming and computational problem solving to serve as an excellent preparation for a degree, or employment, in any STEM discipline, as well as plenty of social science or creative subjects.

Overall, I think the state of computing education is not quite as patchy and fragile as it was in 2017, but nor has it yet lived up to the promise it offered when we put it onto the national curriculum as an entitlement for all.