PGCE Secondary Computing Assignment 1

2019-20

Mark Gadsby

The case for computer science being taught in schools is strong. Some problems do exist around establishing the subject in the curriculum with the correct balance of computer science, information technology and digital literacy (Kemp et al, 2016, Royal Society, 2017), but no one is questioning the direction of travel.

It is clear we are living in an increasingly digital age; as many as 90% of newly created jobs require some level of digital literacy (Blackwood,2016). For Grady Booch (2016) we are living through a computational revolution which represents a paradigm shift as profound as the shift to scientific thinking half a millennium ago. He highlights the fundamental changes to the way humans now communicate, consume media and engage with work or culture. Children need to be taught how to make informed decisions about this digital society and be able to conceive and design new digital systems, not just be passive consumers of technology. This goal is encompassed by the aim of the current English computing curriculum to; 'equip pupils to use computational thinking and creativity to understand and change the world' (DfE, 2013).

A digitally literate workforce will bring huge economic benefits to society. Over 75% of respondents in the British Chambers of Commerce Digital Economy Survey (2017) reported a shortage of digital skills in their workforce. If you couple this with the potential for growth in fields of computing such as mobile and cloud, cyber-security and data analytics, it is clear that giving young people digital skills can unlock productivity and economic potential.

Against this backdrop it was equally clear that computing education in this country was not up to the task of meeting this demand. The Royal Society report, Shutdown or Restart (2012) laid bare the deficiencies in the Information and Communication Technology (ICT) based approach, reporting that the subject had been dumbed down and recommending that it be replaced with all pupils being taught computer science 'as a rigorous academic discipline'.

The term computational thinking (CT) was probably first used by Seymore Papert in his seminal 1980 book, Mindstorms. He was trying to describe the skills children develop when learning how to solve problems on a computer. As someone who has been through that learning process myself and then subsequently employed as a senior software engineer for a significant period of time, I have arrived

at my own short definition; CT is being able to recognise the elements of a problem that can be most effectively solved by a computer.

The modern definition for CT used by Jeannette Wing in 2006 is similar to mine but with more differentiation around the term 'computer'; she used the phrase 'computer—human or machine' instead. These words were further refined with input from Cuny, Snyder & Aho to 'information-processing agent' (Wing et al, 2010), and here we arrive at the most common general definition cited over 5000 times in the literature;

'The thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information processing agent.'

Much work has been completed since (National Research Council, 2010; Barr and Stephenson, 2011) in defining the essence of this broad definition so that the key components of CT might be better understood and ultimately taught effectively throughout the key stages of education. The consensus is that CT is a thought process that involves abstraction, decomposition, algorithmic thinking, evaluation and generalization (Selby and Woollard, 2013). The current GCSE specifications further reduce CT down to just three main headings; decomposition, abstraction and algorithmic thinking (OCR, 2016; AQA, 2016).

Having defined what we feel is important to teach, we must also understand how it is that children learn. In the 1950's a child's development was thought to amount largely to the sum of the child's learning, largely from textbooks and lectures. Jean Piaget (1964) turned these ideas on their head with his theory of constructivism which suggested that learning was driven by development. His idea was that children best achieve learning as a by-product of engagement in, and experience of, the world first hand. Constructivism means that knowledge is actively constructed by the individual and has become 'the dominant theory of learning today' (Ben-Ari, 2001)

Piaget's student Papert (1991) took his teacher's theory and brought it further into the real world. He added to Piaget's thinking on how children learn, with the idea of using project based discovery. He invented the Logo programming language with Wally Feurzeig and Cynthia Solomon at MIT. Children draw on a computer screen by moving a small robotic turtle. By solving problems and creating artefacts, within this environment, they can construct an effective model of not just the coordinate system and angles but also how a computer works. This approach also taps into students' innate pleasure in making real things for other people to respond to. Turtle graphics is currently

implemented in the Python and Small Basic programming languages for use as a classroom tool. ‘Perhaps if we wrote programs from childhood on, as adults we'd be able to read them’ (Perlis 1982).

For me CT is a theoretical framework for teaching computer programming to engender an understanding of computer science. I appreciate the goal of computing teachers is not to produce an army of software developers but rather rounded individuals equipped for all fields of work. Mark Guzdial noted in 2015 that if students destined for other fields have an understanding of core programming concepts such as sequence, selection, iteration and simple data structures it will help them to understand computation in their field and elsewhere in their lives. Yet it has been argued by Jeannette Wing that CT is quite possible without a machine at all (2014).

The quote that crystallises our positions most succinctly was made by Alan Perlis in 1982 ‘Everything should be built top-down, except the first time.’ The situation reminds me of being taught object-orientation in the early 90's from object first principles, and the feeling of being ungrounded that went with it, something Ben-Ari called the object-oriented paradox (2001). It asks the question, how do you forget the correct amount of detail to achieve the required level of abstraction when you have never appreciated those details existed in the first place?

I have observed decomposition being illustrated in the classroom by exploring the steps involved in preparing a breakfast, getting dressed in morning, or selecting a film to watch that evening. None of these examples have a ready computational solution. Without the ability to follow an example through to working code I don't think any insight can be conveyed, and quite possibly an incorrect model of how computers work is constructed in the students’ minds. When it comes to assessment I have seen 'A' level papers requiring candidates to explain the concept of abstraction (for 4 marks), I ascertain from the mark scheme that what's required includes ignoring the unnecessary and using symbolic representation to produce effective and focused programs. All of which can be learnt by rote and says little about the candidates propensity to apply the skill.

Even though bold claims are still being made such as; ‘Computational thinking also gives a new paradigm for thinking about and understanding the world’ (Csizmadia et al, 2015), there is little evidence in the literature to support the notion that you can uproot the principles of CT from the computer and apply them to other areas successfully (Guzdial, 2015). However, CT still has a magnificent contribution to make to other subjects; it is just that claims must be substantiated by tracing computation back to the machine. Aho (2012), for example, considers the implementations of exascale computing and how sequential machines running concurrently on a vast scale have

transformed computational biology to the point where he concludes that new computational models are required.

Assessing the learning of CT concepts remains a challenge to educators. In keeping with the spirit of Papert's work Brennan and Resnick (2012) share digital artefacts, produced by children using the Scratch programming language, via YouTube and the Scratch on-line community. They go on to examine these projects for evidence of sequences, loops, parallelism, events, conditionals, operators, and data. A similar approach was attempted by Werner et al (2012) with their Fairy Assessment, written in the Alice programming language. Working with a key stage 3 aged cohort this study asked students to attempt solutions to a set of part solved problems in order to identify students' understanding and use of CT concepts.

My approach to a analytical framework by which to assess practice is to establish a scheme of work designed to teach the skills of computer programming through cycles of completing engaging coding challenges with increasing complexity. Assessment is then achieved by examining the elegance of these solutions and gauging the quality of the programming through code comments and observation.

Aho, A.V., 2012. Computation and computational thinking. The Computer Journal, 55(7), pp.832-835.Barr, V. and Stephenson, C., 2011

AQA, 2016. GCSE GCSE Computer Science (9-1) - 8520, section 3.1.1

Barr, V. and Stephenson, C., 2011. Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?. Inroads, 2(1), pp.48-54.

Ben-Ari, M., 2001. Constructivism in computer science education. Journal of Computers in Mathematics and Science Teaching, 20(1), pp.45-73.

Berland, M. and Wilensky, U., 2015. Comparing virtual and physical robotics environments for supporting complex systems and computational thinking. Journal of Science Education and Technology, 24(5), pp.628-647.

Blackwood, N., 2016. Digital skills crisis: second report of Session 2016–17: report, together with formal minutes relating to the report: ordered by the House of Commons to be printed 7 Jun 2016.

Booch, G., 2016. The Computational Human. IEEE Software, 33(2), pp.8-10.

Brennan, K. and Resnick, M., 2012, April. New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada (Vol. 1, p. 25).

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C. and Woollard, J., 2015. Computational thinking-A guide for teachers.

Cuny, J., Snyder, L. and Wing, J.M., 2010. Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress, referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

DfE, 2013. Computing programmes of study: Key stages 3 and 4.

Guzdial, M., 2015. Learner-centered design of computing education: Research on computing for everyone. Synthesis Lectures on Human-Centered Informatics, 8(6), pp.1-165.

Kemp, P.E., Wong, B. and Berry, M.G., 2016. The Roehampton Annual Computing Education Report. London: University of Roehampton.

National Research Council, 2010. Report of a workshop on the scope and nature of computational thinking. National Academies Press.

OCR, 2016. GCSE Computer Science (9-1) - J276, section 2.1

Papert, S., 1980. Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc..

Papert, S. and Harel, I., 1991. Situating constructionism. Constructionism, 36(2), pp.1-11.

Perlis, A.J., 1982. Special feature: Epigrams on programming. ACM Sigplan Notices, 17(9), pp.7-13.

Piaget, J., 1964. Part I: Cognitive development in children: Piaget development and learning. Journal of research in science teaching, 2(3), pp.176-186.

Royal Society, 2012. Shut down or restart?: The way forward for computing in UK schools. Royal Society.

Royal Society, 2017. After the reboot: Computing education in UK schools. Policy Report.

Selby, C. and Woollard, J., 2013. Computational thinking: the developing definition.

Werner, L., Denner, J., Campe, S. and Kawamoto, D.C., 2012, February. The fairy performance assessment: measuring computational thinking in middle school. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (pp. 215-220). ACM.

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

Wing, J.M., 2014. Computational thinking benefits society. 40th Anniversary Blog of Social Issues in Computing, 2014.