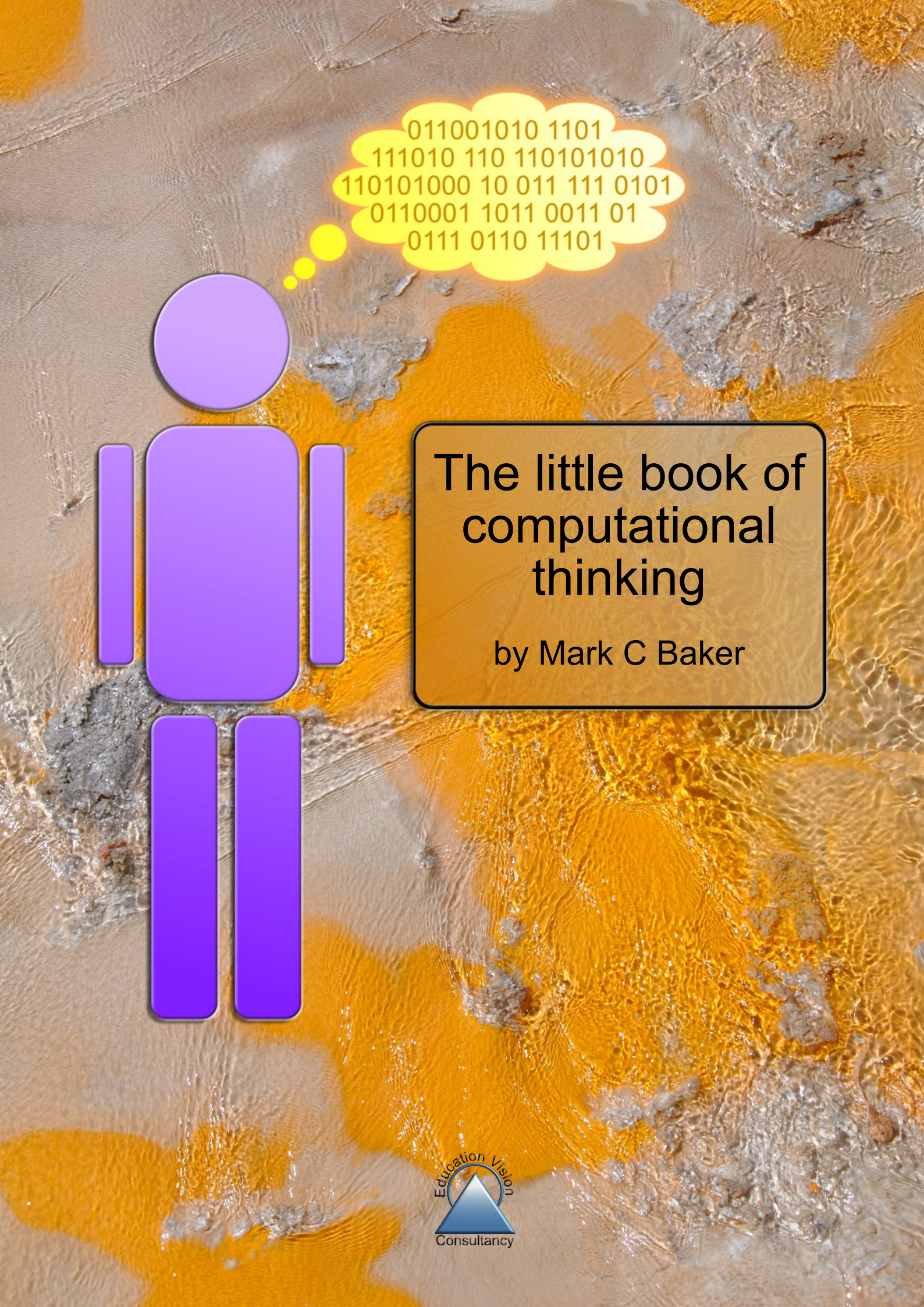


The little book of computational thinking

by Mark C Baker

Education Vision
Consultancy

The little book of computational thinking

Why computational thinking is useful

Computational thinking is usually presented as being what underpins much of computer science[1]. The 2013 UK National Curriculum documents for computing open with the following line.

"A high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world."[2]

Seymour Papert is credited with originating the term in 1980[3], but Jeanette Wing is often cited as being the person who popularised it in her 2006 paper[4]. She felt that all learners should be exposed to computational thinking, not just those specialising in computer science. She concludes her paper with the following.

"We should look to inspire the public's interest in the intellectual adventure of the field. We'll thus spread the joy, awe, and power of computer science, aiming to make computational thinking commonplace."

For anyone teaching computing, or a related subject, the concept of computational thinking is especially useful for helping to understand some of the key ideas that underpin computer science. For many, it is the key driver when undertaking curriculum design.

What is computational thinking?

In the early days of the 2013/14 UK computing curriculum, computational thinking seemed to get used increasingly in order to explain and even to justify, this new subject. The term ended up being banded about quite a lot, although I was not always convinced that those who used it had a really clear idea of what it was. I certainly felt somewhat in the dark, until I came across the *Computational Thinker: Concepts and Approaches* diagram published by Barefoot Computing (see below).

This clearly identifies six fundamental concepts, such as abstraction and decomposition, as well as some key underlying approaches. Very helpful though this diagram is, the pedantic side of my nature kicked in and I felt uncomfortable about the approaches side of the diagram. Persevering is a characteristic, rather than an approach and debugging is a skill.

This prompted me to investigate further, to see if I could come up with a diagram that I felt better represented the fundamentals of computer science. A quick online search showed that there is no single definition of computational thinking that is widely agreed upon, although the concepts *decomposition*, *pattern recognition*, *abstraction* and *algorithms* are commonly mentioned.

The Google for Education website[5] has a longer list of concepts, specifically mentioning abstraction, algorithm design, automation, data analysis, data collection, data representation, decomposition, parallelization, pattern generalization, pattern recognition and simulation.

Diagram reproduced under the terms of the Open Government Licence V2.0
http://www.nationalarchives.gov.uk/doc/open-government-licence/version/2/

A more reduced characterisation is the three As, *abstraction* (problem formulation), *automation* (solution expression) and *analyses* (solution execution and evaluation). The website ComputationalThinking.org opts for *Define*, *Translate*, *Compute*, *Interpret*.

Why computational thinking is unhelpful

Steve Easterbrook wrote a paper in 2014[6] that was somewhat damning of computational thinking. He felt that it had been widely adopted in the US as a course marketing tool.

"Since the concept was introduced, there has been remarkably little critical thinking about computational thinking. The few critiques that have been written tend to focus on either the vagueness of the term, or on a concern that the field of computer science should not be reduced to just one of its practical tools."

He views computational thinking as inherently reductionist, leading to problems that do not have a computational solution being ignored (e.g. those will an ethical dimension or requiring value judgements to be made). His paper is particularly concerned with sustainability and one of his conclusions is as follows.

"Over-reliance on computational thinking is likely to lead not just to computational solutions that ignore social and environmental sustainability, but often to solutions that actively undermine such sustainability."

Easterbrook also considers the importance of communication in the formulation of solutions.

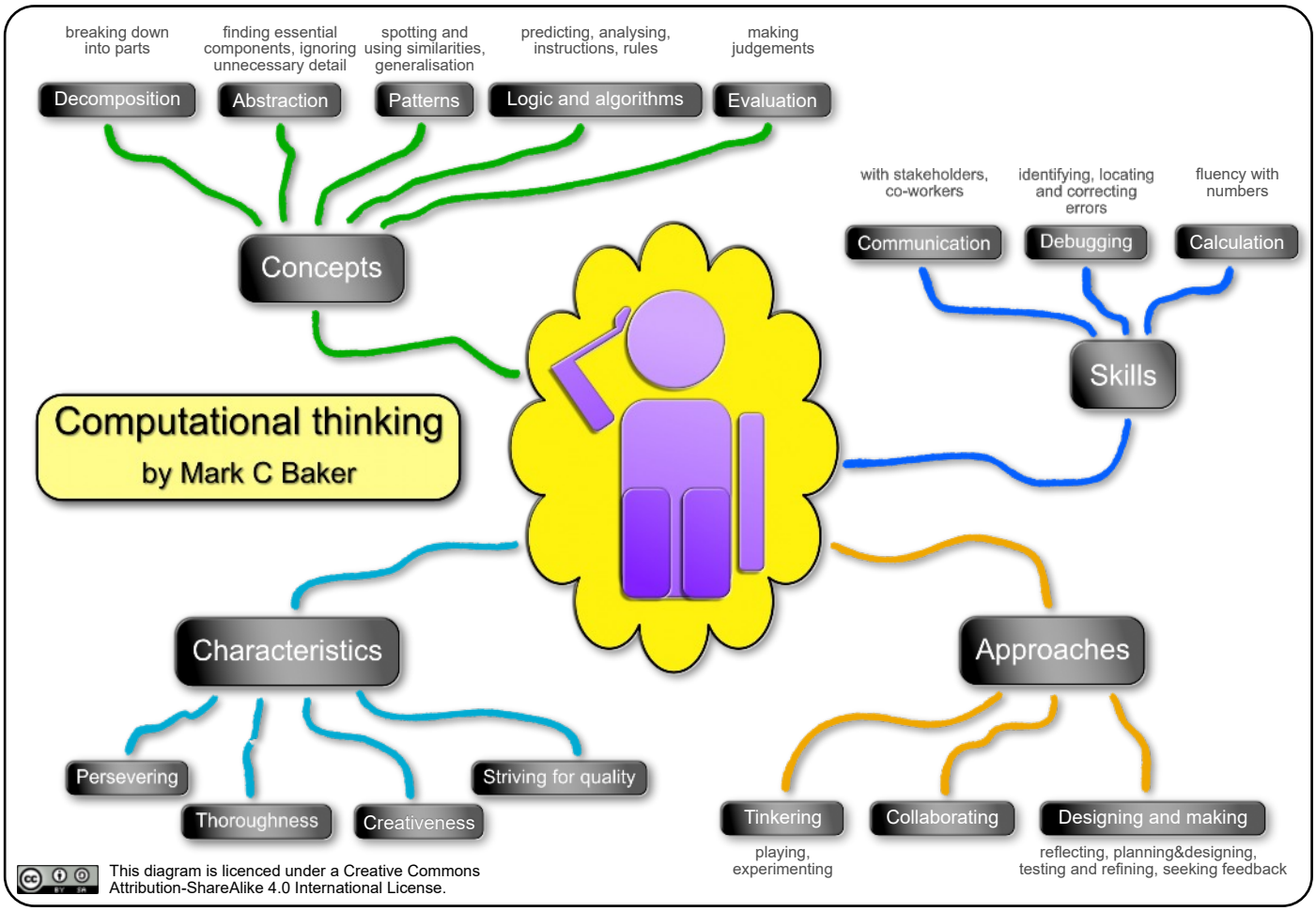
"Computational thinking thus ignores the fact that any particular expression of the "the problem to be solved" is the result of an ongoing negotiation between the competing needs of a variety of stakeholders."

The idea of an ongoing evolution of computing solutions is certainly key within agile programming approaches.

The reductionist nature of computational thinking is perhaps more of an issue within higher education, where learning is much more specialised and focused. It is less of an issue within schools, where any learner in receipt of a well-rounded education should be exposed to a range of thinking styles and problem solving approaches. However, it is a timely reminder that computational thinking as a concept should be challenged and explored periodically. Computational thinking seems to have achieved the status of an education orthodoxy, whilst being variously defined and understood. It behoves us as professionals to periodically challenge the accepted orthodoxies, if we are to move the subject of computing forwards and avoid the "shallow manipulations" referred to in Papert's 1996 paper[7].

Devising a new diagram

Computational thinking is still a useful idea for classroom practitioners, as it helps us to understand and explore the subject of computing. Since there are many different definitions,



none of which I am entirely happy with, I decided to produce my own diagram, to see if it would help move the debate forward.

The resulting diagram (see above) should not be filled with so much as to obscure the key areas, nor should it be so reduced as to be unhelpful. As a subject specialist, the temptation was to include all positive thinking approaches and characteristics, to grab as much as possible for "my subject." I have tried to resist this! However, this diagram, developed from those that came before it, is the work of just one person. If it has value, it is as stimulus material, to encourage thought and debate about computing, especially in schools.

Concepts

Decomposition

One of the first steps to be taken when writing a complex program or creating a complex system, is to break down what is required into smaller, more manageable chunks. If I wanted to send people to the moon and return them safely to Earth afterwards, I might decompose the problem into the following stages; LAUNCH FROM EARTH, TRAVEL TO MOON, LAND ON MOON, EXPLORATION ON MOON, LAUNCH FROM MOON, TRAVEL TO EARTH, LAND ON EARTH. I could then take any one of these and decompose it further, for example, TRAVEL TO MOON might include *propulsion*, *life support*, *navigation*, and *communication*. Decomposition continues until the chunks are small enough to be tackled and solutions formulated. The individual solutions can then be integrated to form the solution to the entire problem of getting people to the moon and back.

It should be noted that decomposition is a general problem-solving technique and not one that is uniquely applied within computer science. The same could be said of the other key concepts, it is more how they blend together and are applied, together with the importance of digital technologies, that gives the subject its unique flavour and value.

Abstraction

Even relatively simple real-life situations involve a great many factors and variables and it is usually impossible to take everything into account, e.g. when writing a program to assist in a particular situation. It is usually necessary to create a simplified version of reality, identifying and focusing on the essential items, whilst ignoring as much of the detail as possible. The London Underground or Tube map is a good example of an abstraction. The detailed geographical information is ignored, with just the order of stations and the places where travellers can change lines being retained. Some accessibility information is also added, to produce a simplification that is extremely useful to travellers.

Many factors may interact in very complex ways to produce the skill set of an individual footballer. A programmer working on a computer game may decide to list some skill areas, such as passing, dribbling and shooting and allocate a score out of ten for each skill area, to each player. Each time a player passes, dribbles or shoots, the appropriate score is used to help decide whether the skill is executed successfully - the higher the score, the greater the chance of success. This is an example of a data abstraction, the way that numbers are to be used to represent a simplified version of reality.

Patterns

Spotting repeating patterns is key to writing efficient algorithms/programs, which can exploit patterns by using repeating loop structures or reusable modules, such as procedures and subroutines.

Logic and algorithms
The ability to think logically is an essential part of understanding and writing algorithms, the sets of instructions/rules that make up computer programs. I therefore chose to put these two together, although some models of computational thinking separate them.

Learners should be able to read, understand and appreciate algorithms written by other people, as well as being able to construct their own. Standard algorithms, such as those for sorting or searching data, are worthy of study.

Evaluation
Algorithms should be evaluated to try and determine if improvements can be made that lead to greater efficiency. It is very unlikely that the first attempt at writing an algorithm produces the best possible solution. Small, iterative improvements can often be made, sometimes wholesale rewriting will be necessary. Solutions that are to be used by other people should be evaluated prior to and after full deployment, in order to improve them. This may involve making changes to one or more underlying algorithms, it may simply be a case of improving how people interact with a program or system, the so-called human-computer interface.

Skills

Communication
Communication skills are crucial, given the need to coordinate the work within teams and to understand the needs and wants of end users.

Debugging
Debugging takes mental effort and children will often want their teachers to do the debugging for them. They must therefore be encouraged to be more independent and helped to learn what is a multi-stage skill. Errors in algorithms must first be recognised, then located and finally fixed, before retesting to ensure that all is now well and that the 'fix' has not introduced more errors.

It often helps if algorithms are developed incrementally, in small steps, so that they can be checked and debugged in an on-going fashion. If a lot of code is written in one go, the number of problems or bugs that are introduced can be overwhelming to a novice programmer and stop them from experiencing success.

Calculation
Writing algorithms often means representing reality using numbers, therefore numerical fluency is important.

Approaches

Designing and making
In common with other school subjects, the approaches of the design and make cycle, such as planning and seeking feedback, are important when writing programs and designing systems.

Collaborating
Working together is often a key part of computational thinking and widely seen in professional situations. Whilst it is often seen in the classroom too, the difficulty with assessing it and the fact that it is not therefore part of the formal examination system, means

that it does not receive the attention it deserves, especially as children progress through their education.

Tinkering
Playful experimentation is often apparent in those that are the more gifted computational thinkers. The readiness to engage in tinkering has been suggested in the past as one reason why boys have performed better than girls in computing related subjects. Certainly, the amount that can be learned in computing by playing around and seeing what happens, should not be underestimated.

Characteristics

There is room for much difference of opinion with a diagram such as the one above, especially in the *characteristics* section. However, it is an interesting challenge to try and identify a small number of key characteristics. Here is my choice.

Striving for quality
The best solutions tend to result from teams and individuals who are not prepared to accept second best, but are keen to find the best answers, presented to potential users in the most effective ways.

Creativeness
Following rules and conventions is often a key part of computing solutions. Whilst being mindful of these, computational thinkers need to be prepared to "think the unthinkable" and throw out the rule book if they are to discover new techniques and achieve big steps forward. Ideally they will develop a good instinctive 'feel' for when it is appropriate to be creative and when it is better to follow existing conventions.

Thoroughness
The pedantic nature of computer programming and the nature of the solutions it can produce mean that being thorough and attending to detail is an essential component for a successful computational thinker. Whilst there is some overlap with *striving for quality*, I felt this deserved a separate mention.

Persevering
Successful computational thinking requires persistence; first attempts often fail and even when they work, there may be better solutions waiting to be discovered. It can be mentally taxing and a determination to find answers is often needed.

In conclusion

Reflecting on computational thinking has led me to the following conclusions.

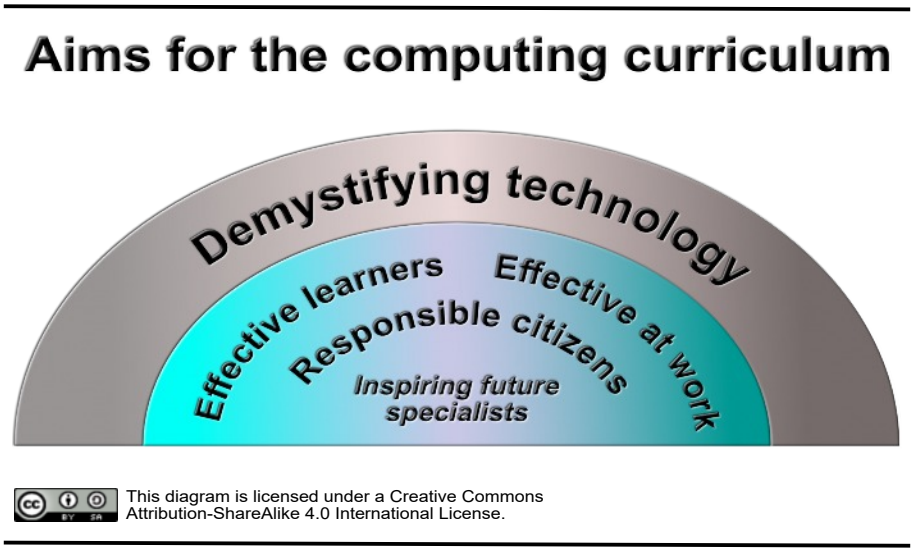
- Computational thinking is not the be all and end all of computing or computer science
- Computational thinking is currently ill-defined
- Despite the first two bullet points, computational thinking is a valuable tool for thinking about computer science and what it should be about
- Greater debate about the nature of computational thinking would be beneficial to those in schools who work in this area

Postscript

Readers must decide for themselves whether the proposed new computational thinking diagram is a significant improvement over the earlier Barefoot Computing diagram, or indeed, if some other definition of computational thinking is even more useful.

However, examining computational thinking in this way does throw up the question of how does computational thinking fit into the National Curriculum subject of C*omputing*?

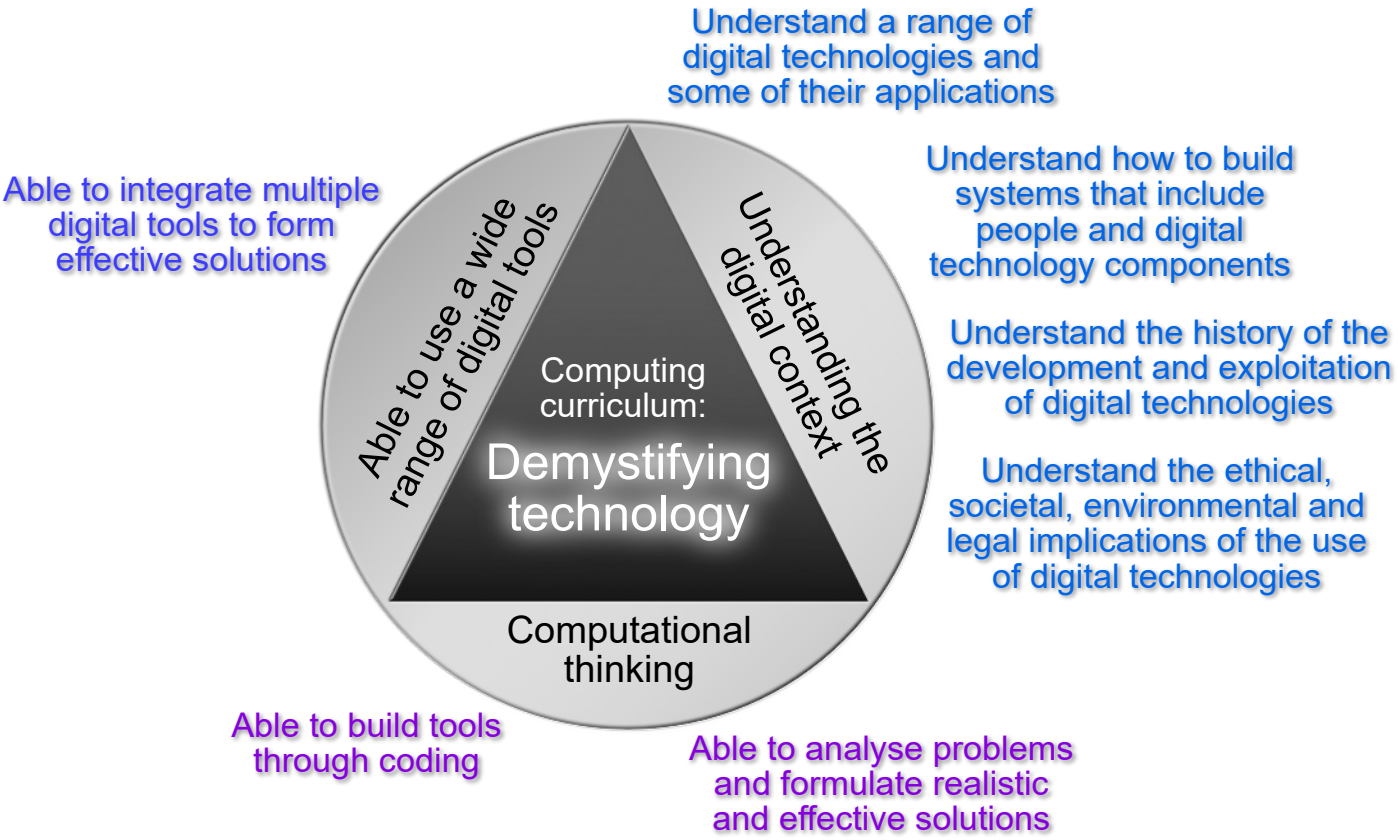
I believe that we need to develop a much more digitally confident and competent population to meet the needs of the future, not just for use at work, in education and within our everyday lives. We must also ensure the widest possible participation in national debates around the uses of digital technologies, the way in which they impact on people's lives and the sorts of controls and regulations that need to be put in place. This will not happen if there is a widespread feeling that digital technologies are complex and beyond the ability of many people to understand. Therefore I would place demystifying technology at the heart of *Computing*, as its primary aim.



Responsible citizens will need to have a solid grounding in computing in order to play their role in national debates on digital issues. The ability to use and, where necessary, to integrate, a wide range of digital tools, will make people more effective as learners and at work. At the same time, the curriculum should inspire the minority (albeit, in all probability, a growing minority) who will go on to become the digital experts of the future.

With demystifying technology as the central aim, I see this being made up of three key components, *computational thinking*, *being able to use a wide range of digital tools* and *understanding the digital context* (see over).

Components of the computing curriculum



There is a danger that schools in England have become too focused on computer science, driven by changes to the examination system and that the choices for KS4 students are now too narrow. A curriculum that puts demystifying technology at its heart is not only more inclusive and relevant, but also more ambitious. Some pupils will want to focus more on one of the component areas than the others, this is not a problem provided they receive enough of a balance to meet the central aim of demystifying technology.

Most importantly of all, examination courses should be of high quality, designed to provide an exceptional educational experience, rather than being geared towards the maximum accumulation of league table points. The focus should be clearly on the types of young people that the system should be capable of producing. Education should not be solely about examinations, but in reality, examination specifications drive much of what happens, especially within the secondary phase.

Sources

[1] *Computational thinking: A guide for teachers* by Andrew Csizmadia et al, published by Computing at School, November 2015.
https://community.computingatschool.org.uk/resources/2324/single
See also https://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computational-thinking/

[2] *UK National Curriculum computing programmes of study*
https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

[3] Wikipedia entry for *Computational thinking*
https://en.wikipedia.org/wiki/Computational_thinking

[4] *Computational Thinking*, J. Wing, Communications of the ACM, vol. 49, no. 3, pp. 33–35, 2006. http://www.cs.cmu.edu/~./15110-s13/Wing06-ct.pdf

[5] *Exploring Computational Thinking*, Google Education
https://edu.google.com/resources/programs/exploring-computational-thinking/index.html#!ct-overview

[6] *Computational Thinking to Systems Thinking: A conceptual toolkit for sustainability computing* by Steve Easterbrook, Dept of Computer Science, University of Toronto. From the proceedings of the 2nd International Conference on ICT for Sustainability (ICT4S 2014). Published 22/08/2014, Atlantis Press. doi:10.2991/ict4s-14.2014.28 https://www.atlantis-press.com/proceedings/ict4s-14/13446

[7] *An Exploration in the Space of Mathematics Educations*, by Seymour Papert, International Journal of Computers for Mathematical Learning, Vol. 1, No. 1, pp. 95-123, 1996.
http://www.papert.org/articles/AnExplorationintheSpaceofMathematicsEducations.html

For other papers by Mark C Baker go to the Resources section at www.educationvision.co.uk

Books by Mark C Baker, available worldwide from Amazon and other booksellers:

Living in a digital world: Demystifying technology

The Kennet and Avon Canal in pictures