Time to re-load?
Computational Thinking and Computer Science in Schools
Executive Summary

The technology revolution has changed the way many of us work and interact, it has generated new industries and new businesses, and it is natural that we now look to schools, teachers and the education system to help us to understand how we might prepare our children to live, work and make effective use of what computer technology offers. But how best can we do this?

A mess? 2012 has seen the Secretary of State for Education state that “ICT in schools is a mess” and he has called for a new approach with the hope that technology can be used creatively to develop curricular content: the ‘wiki’ curriculum. What is happening with ICT and computer science education in schools has also been the subject of a 2011 Naace report entitled “The Importance of Technology”, an Ofsted report on ICT in schools, and the importance of providing young people with the skills required by the new workplace is captured by Nesta’s Next Gen report. Clearly there is growing concern and government commitment to change, so what change should we make and why?

Is Computer Science the answer? Computer science is an important element of the debate. The Royal Society’s 2012 ‘Shut down or restart?’ report suggested that a sound understanding of computer science concepts enables people to get the best from the systems they use, and to solve problems when things go wrong. However, computer science is evolving rapidly and its interdisciplinarity means that its evolution touches on many domains and every day life. There are significant challenges for those interested in how best to include it in the curriculum.

Are we sure we know what we want to change? There is already some excellent teaching of ICT and computer science in some schools within the current curriculum and programme of study, so not everything is wrong. Care needs to be taken that the changes we make do lead to a better learning experience at school: an experience that inspires and educates. But, are we clear about what is wrong with computer science and ICT in schools now? Can we be precise about the rationale for what learners at different stages need to be taught? What do we want learners to be able to achieve as a result of studying computer science? Where do ICT and computer science fit in the structure of the school curriculum: media, design, science, cross-curricular?

How can learners tap into the power of computational thinking? The skills of computational thinking can be taught with or without computers, by exploring how processes work, looking for problems in everyday systems, examining patterns in data, and questioning evidence. With a computer, learners can put their computational thinking into action. Could a focus on computational thinking better equip learners to use their understanding effectively and to learn how to apply a range of computing tools? Writing the code that makes a computer behave in a particular way is a creative pursuit: reflecting on what you have constructed is a key part of learning. We may therefore valuably ask: How can we develop good computational thinking for children?

Are we looking in all the right places? Are there less obvious areas of research that might help us answer some of these questions? For example, many people encounter the experience of Flow and are all too familiar with the experience of losing themselves in a task. Might the idea of Flow itself help us understand the learning process in computational thinking and computer science? Researchers in the psychology of programming have spent decades exploring how people learn to code, surely their expertise needs to be drawn into the debate?

There are no short cuts to answering these questions. The process of addressing them requires an interdisciplinary and participatory approach that involves groups from across the sectors that is inclusive in nature and powerful in design. This will require an approach that is new to society, schools, teachers and learners: a process that must be both flexible in its thinking and realistic in its understanding of the role of schools.
Computational Thinking and Computer Science in Schools

“We need computational thinking to extend previous forms of thinking, if we are to understand the universe, and how we fit into it, including how we and other living things came to exist on a planet that originally had no life.” Aaron Sloman 15 April 2012

1. Introduction

In Chicago today the Obama re-election campaign is set to be the most technically sophisticated ever seen with voters being wooed via Twitter and Facebook, and digital technology along with those who understand how to build and use it set to play a key role in influencing people’s decision making. Across the Atlantic in the UK we face an abundance of choices about how to exploit and use technology, and this poses an enormous challenge for both the current and future education of our children. The realisation that we need people who can produce as well as consume technology has brought a new energy and excitement about computer science and computational thinking, which is being heralded by some as the new literacy of the 21st century. The technology revolution has changed the way many of us work and interact, it has generated new industries and new businesses, and it is natural that we now look to schools, teachers and the education system to help us to understand how we might best prepare our children to live, work and make best use of what computer technology offers.

2. What is computer science?

Computer science is a discipline that involves the understanding and design of computers and computational processes. In its most general form it is concerned with the understanding of information transfer and transformation. In the early days of computer science education, a degree in computer science covered a large amount of what there was to be known about the design and construction of a computer leading to an engineering qualification, or what there was to be known about how computers worked leading to a science qualification. The science qualification would have likely included elements of theoretical computer science, discrete mathematics, machine code, data structures and functional programming, computer architecture, operating systems and high level programming languages. However, by the early 1990’s different types of computer science degree started to emerge with interdisciplinary subjects like Software Engineering and Design, Artificial Intelligence, Human Computer Interaction and Artificial Life becoming popular. The advent of the Internet brought a new set of concepts about networked computing, security and privacy and the popularity and growth of computer games spawned degrees in game design. A strong interdisciplinary perspective on computing continues to evolve through, for example, the emergence of bioinformatics, digital humanities and digital ethics.

The field is unrecognizable from the days of Alan Turing’s influential description of a computer in ‘On Computable Numbers’ published in 1936. It is important to recognize the rapid evolution of computer science and the implications of this for teaching and learning. It is also important to remember that there are some central concepts within computer science that can offer a useful focus. Computer science is concerned with the representation, storage, manipulation and presentation of information with "the study of symbol-manipulating machines, with communication between man and machine and with the application of these machines" (http://www.cs.mtu.edu/~john/whatiscs.html). The subject of computer science has a central focus upon the handling and manipulation of information and integrates the theoretical study of computer science with its practical application.

Computer science is not limited to the sciences, but is very much part of culture, media and the arts too. For example, see work by digital artists such as Ben Hooker who use programming in their work (see for example, http://hookerandkitchen.com/theweekend/). Notice too the sophisticated animations and filmic quality of many computer games and the rise in popularity of areas such as music informatics and 3D sound and vision.
‘Shut down or restart?’ the Royal Society’s 2012 report into computing in UK schools, highlights another benefit of computational thinking. A sound understanding of computer science concepts enables people to get the best from the systems they use, and to solve problems when things go wrong. Citizens able to think in computational terms are able to understand and rationally debate issues involving computation, such as identity theft and electronic voting systems for elections.

What are the challenges?
Computer science is evolving rapidly and its interdisciplinarity means that its evolution touches on all domains and more importantly upon every day life, it brings with it significant challenges for those interested in how best to include it in the curriculum. For example:

• Where do the visions, such as the semantic web, ubiquitous computing, anywhere anytime computing fit within society’s understanding of computing? Such visions create new ways of thinking about technology and thus the embedding of ‘intelligent technology’ is possible.

• How and should teachers, children and society keep up with the newest computer science developments and understanding? Is this desirable, is it possible and what aspects of computer science need to be taught in schools and to what needs to be the age of the learners?

How can we tackle these challenges?
The skills of computational thinking can be taught with or without computers, by exploring how processes work, looking for problems in everyday systems, examining patterns in data, and questioning evidence. With a computer it is possible to teach computational thinking by programming computers, but these programs must be based on some principles and processes that the learner understands and put into effect through the program they write. This helps empower the individual and is important because through that understanding comes the possibility to think differently when making decisions that involve smart technology. Programming is not just grappling with long lines of code. Since the 1960s, there have been a series of attempts to design new kinds of programming languages and systems that are designed for the novice and non-programmers – for everyone in fact. If computer science in schools is to succeed in “giving everyone a chance- stretching the minds of great thinkers” (Sloman, interview in this document) then we must understand what is happening in schools and ‘the context of the teacher and learner’ (Mee, interview in this document).

The process through which we can identify and agree about the core elements of computer science for schools requires an interdisciplinary and participatory approach that involves groups from across the sectors and that is inclusive in nature and powerful in design. This requires an approach that is new to society, schools, teachers and learners: a process that must be both flexible in its thinking and realistic in its understanding of the role of schools.

What do we know about Policy, Curriculum, the Government OFSTED and ICT in schools today?
In his speech at the BETT show1 this year Michael Gove said that “ICT in schools is a mess” and called for a new approach in which he proposed that the existing ICT Programme of Study should be withdrawn from September 2012, but that ICT would remain a compulsory subject at all key stages and every stage of the curriculum. He opened a consultation on this proposal and stated that: “no English school will be forced to follow it any more. From this September, all schools will be free to use the amazing resources that already exist on the web.” Mr Gove expressed the desire for new courses to be developed by universities, businesses and others including a new high quality Computer Science GCSE, and new curricula for schools. He acknowledged that the

essential requirements of the National Curriculum need to be specified in law, but hoped that technology would be used creatively for the development of curricular content: “we need to consider how we can take a wiki, collaborative approach to developing new curriculum materials”.

Mr Gove stated that his motivations came from the likes of Bill Gates, who had expressed the view that it was more important for children to understand computer programming now than it had been in his youth; and Google chairman Eric Schmidt, who had suggested that England had allowed its education system to “ignore our great heritage and we are paying the price for it.” There have also been a range of influential reports that express concern about computing in school, for example: The Royal Society Report ‘Computing in Schools: Shut down or restart?’; the Next Gen report by Livingston and Hope for Nesta, the Naace report entitled “The Importance of Technology”, and the Ofsted report on ICT in schools, which reported inadequate ICT achievement in nearly a fifth of the secondary schools their team visited.

However, it is not all doom and gloom, Mr Gove did recognize in his Bett speech that “some ICT teaching in schools is already excellent” and he acknowledged that the Ofsted report makes a particular point of highlighting that some primary schools do an excellent job teaching basic computer programming, which students find inspirational and educationally valuable. Ofsted also reported that many primary schoolchildren by Year-6 are proficient in digital literacy. Nevertheless, there is clearly a consensus that some sort of changes needs to be made to what and/or how children and young people learn about computing and computer science at school.

What are the challenges?
The decision that a change is needed to the school curriculum combined with the ever-increasing rapidity of technological developments raises many questions and challenges. These include:

• Are we clear about what is wrong with computer science and ICT in schools now?
• What is the rationale for what learners at different stages need to be taught?
• What do we want learners to be able to achieve as a result of studying computer science and/or ICT at school? Do we want to prepare students to take up a career in a computer related industry or to be able to use their technology in an intelligent manner and adapt it to meet their needs?
• What is happening in schools currently, what works and what might be used by others? The capturing of this practice-based knowledge is not straightforward, but it is key to understanding what needs to be taught in schools and how this is different from what needs to be taught in Universities. How do we capture this knowledge and integrate it into curriculum changes?
• There is good programming activity happening already in some schools, are we therefore sure that changing the curriculum is really what is needed?
• Where do ICT and computer science fit in the structure of the school curriculum and how do we recognise the interdisciplinary nature of these subjects? Media, design, science?
• The possible content of a school computing curriculum is huge and identifying the key components is not straightforward: How do we address the initial and continuing training demands if a new curriculum is introduced?

How can we tackle these challenges?
The understanding and development of computational thinking is fundamental to how we equip young people to make the most of what technology has to offer, if we make this a major focus of all computing education we may better equip learners to be able to use this understanding effectively to learn how to apply particular tools.

There is a growing body of excellent resources available, many of which are free. However, each needs to be recognised within the context of the groups who create the resources. For example, the computing at schools website (see http://www.computingatschool.org.uk/) has an excellent
range of resources and presents key definitions that differentiate Computer Science form Information technology for example; and proposes key constituents of what learners need to understand in their curriculum document. However, this is written very much from a university computer science perspective and highlights the fact that we need to be clear about the nature of the rationale for the development of the curriculum for different age ranges of learners. For example, higher education computing and industry input may be more relevant to the design of content and curriculum for older learners.

3. What do we know about computer programming as a tool to develop and enhance computational thinking?

A useful starting point for this question is to consider what we know about how people learn to program a computer. We know that part of the process of programming is the translating of a mental plan into one that is compatible with the computer. One area of research that has something useful to say about how people make this translation and learn to construct working computer programmes is the psychology of programming, which is about understanding the thinking that is needed in order to construct a computer program.

Research from the psychology of programming illustrates some of the challenges that may impact upon how a person learns a computer programming language. These are:

- the language’s concepts and syntax (e.g. Prolog versus Java),
- the programming environment (e.g. visual editors versus text editors),
- the example domain being used (e.g. scientific versus social media),
- the ability to hold in their mind the details and abstractions about the language environment and the problem being solved and the potential dynamic fluidity of change.

All this understanding is needed when designing and writing the programme code and when errors occur and ‘bugs’ need to be tracked down.

A commonly held misconception is that programming is associated with coding and not with more abstract, design-oriented and intellectual challenging activities. Essentially a computer program is a model that represents part of the creation of an idea hatched in the mind. These processes, arising from human experience and thought, are huge in number, intricate in detail, and at any time only partially understood. Even though the programs are carefully handcrafted discrete collections of symbols they continually evolve as our understanding of the model, as the computational thought, deepens, grows and generalises. Computational thinking can revolutionise the way we think and the way we express what we think. So while a framework for expressing certain thinking deals with notions of ‘what is’, a framework for expressing computational thinking provides us with notions of ‘how to’.

In order to understand program behaviour it is necessary for the programmer to have a model of the computer that will execute the program: the ‘notional machine’ Taylor, J. and du Boulay, B. (1987). It is a very creative process, like the creativity in musical thinking that requires the musician to hold in their mind a model of how a piece of music will sound: the musician is able to ‘move the notes round’ and hear other possibilities within their thoughts.

Writing the code that makes a computer behave in a particular way is a creative pursuit. Watching young children make sense of the world teaches us an important lesson: that people learn best when they are making things, and sharing what they’ve made with each other. Making something produces something to talk about, reflect upon, and ultimately learn with. And it presupposes that one has something with which to build – blocks, or paints, or musical instruments. In the past, most things were more or less unbuildable – how do you produce a model of the universe, play around with the forces of gravity, or model the outbreak of the First World War with pencil and paper, unless you’re a mathematician But now, with computers, literally anything is possible. Computers open doors which used to be closed to everyone except the very few: the mathematicians,
scientists and musicians who could build things in their heads. As Seymour Papert said, in describing his theory of ‘constructionism’ some 20 years ago, the special thing about building is that it constructs a ‘public entity, whether it’s a sand castle on the beach or a theory of the universe’. Noss (2012).

Reflecting on what you have constructed is a key part of learning. Until now, this lesson didn’t easily translate into learning more generally. But now, with computers, ideas that could only live in the minds of people can have a life on the screen – bringing them alive, and, most importantly, giving people the chance to construct mental representations of dynamic systems alongside virtual ones. Noss (2012).

What are the challenges?
Good decisions about using programming as a tool to develop and enhance computational thinking require clear understanding of the barriers to taking ‘the thinking’ into the form of a program. It is not easy to keep in your mind ‘the notional machine’ and that is only part of what must be kept in the programmer’s head. It is even harder to teach the novice programmer, the student and the child because there is so many different abstract layers that must be held in place at any one time. Key questions that need to be answered are:

- What kind of ‘computational thinking’ should we teach and want our children to learn?
- How can we develop good computational thinking and exploring for children?
- How do we make clear decisions about which tools to use when there are so many tools to choose from?
- Which language should we teach in the primary school and which in the secondary school, and why are these the languages of choice?

How can we Tackle the challenge?
There are many tools that have been developed with the intention of supporting learning through the expression of creativity a list of such tools is included in the appendix to this report. However, we need to be clear about the aim and the use of each of these tools and we need to understand the potential impact of using a particular tool.

Computational thinking can be developed through many different approaches, for example exploring computational thinking through using Scratch. In exploring the world this way we examine different aspects of the tool e.g. the application, the objects being constructed, the script being used to drive the objects. Scratch permits a child to explore the development of a movie by putting together different ‘content objects’. To be able to construct the movie the child needs to have an idea of what that would look like, what objects they might use and then how to get those objects to move and do things. The moving and the doing things requires the child to write a script for each character. The kind of ‘actions’ that a character is determined by the type of script that is written. The script is the program.

4. What is the immersive experience of creative learning?
Flow is the mental state of operation in which the person is fully immersed in what he or she is doing by a feeling of energized focus, full involvement, and success in the process of the activity. Proposed by Csíkszentmihályi and Csikszentmihalyi (1990) the positive psychology concept has been widely referenced across a variety of fields. Many people encounter the experience of flow and many computer scientists, computational thinkers and those who play video games will be all too aware of losing themselves in a task, when they become truly immersed. An important point about the flow experience is that after each instance, a person is more than the person they were before. Each piece of knowledge absorbed, each new refinement of a skill, enlarges the self and makes it more highly ordered, forming, in their words, ‘an increasingly extraordinary individual’(page (24) Csíkszentmihályi and Csikszentmihalyi (1990). Flow is an experience “so gratifying that people are willing to do it for its own sake, with little concern for what they will get out
'What the Research Says’ Briefing 2
27th April, 2012

of it, even when it is difficult or dangerous” (page 71) Csíkszentmihályi and Csikszentmihalyi (1990).

Both computational thinking and computer science require a creative space in the mind to think through the problems and to explore solutions. The creative thinking for the programmer brings together the internal model (the notional machine) and the external model (that being created by the programmer as a representation of that internal model) seamlessly while in this mental state of flow. This focus that we can see as being deeply engrossed in the activity appears to be an essential part of the creative problem solving space. Education has always seen as a critical part of learning that a child is engaged with a task and is focussed. The question becomes what is happening for the learner when in this mental state of flow.

In an interview with Aaron Sloman, although he does not refer to flow, he refers to representational redesription proposed by Karmiloff-Smith as a possible process by which humans can build increasingly abstract and more widely applicable theories from these learned behaviours.

Aaron states “If you teach kids to do things and they notice what they are doing and they see some of the patterns and they can then start articulating what they have discovered by doing and in some case Karmiloff-Smith representational re-description starts to happen. They don't just see a pattern between what I did now and what I did previously' they start re-organising a whole field’.

This would seem quite powerful if in the process of computational thinking that the mind not only is solving the problem to hand but also potentially processing the thinking space in such a way that an abstraction is possible and the new ideas are incorporated and created.

What are the Challenges?
The learner’s development from a ‘Flow’ perspective is an immersive experience due to the sense of lost in self, self-forgetfulness. But it is the immersive experience that is problematic. The idea of Flow may itself help us understand the learning process in computational thinking and computer science. However, the hidden world between the computer and the child, where computational thinking is taking place is difficult to examine and explore. This is not a comfortable aspect in modern society that measures learning by visible product.

Evidence\(^2\) shows that people who think with models consistently out perform those who don’t. And, moreover people who think with lots of models out perform people who use only one. To develop these models requires the learners to have time to explore and experiment and test out taking risks and being engaged with the process. There is no shortcut and it is challenging, demanding new ways and models about thinking about learning, but it is a critical step to ensure learners are empowered to make critical decisions about how they engage with this world and why.

How can we tackle the challenge?
We need time to comprehend what we are learning. Computational thinking is no different but perhaps it demands a creativity of thought that we had not in the past been able to explore. Now we are trying to make this explicit but we don’t have a frame of reference. The thinking “space” entered and committed to by a child, adolescent or adult enables the experiencing self, the inner self rather than being self-conscious. To achieve this requires exploring through computational models of play, exploration and solutions that help in developing ways of learning and exploring the world of mind in relation to how the world works with, for example ubiquitous computing.

We need to attend to the affective experience of learning as well as the cognitive and ensure that learners want to learn before we teach them how to program a computer.

\(^2\) https://www.coursera.org/course/modelthinking
Michael Young proposes that we need to reflect on the work of Zuboff’s (1984) investigation into the impact of technology in the workplace. A similar approach to understanding the teachers and learners in the classroom and how to think about computational thinking away from the computer and how to understand computational thinking as part of using the computer: both the knowledge as a discipline and what the learner needs to understand in order to make sense and to start learning about thinking computationally and computer science.

“A vision came together for me that morning...I realized that the people I had been interviewing were on the edge of a historical transformation of immense proportions, as important as that which had been experienced by the eighteenth and nineteenth century workers....I saw that a world of sensibilities and expectations was being irretrievably displaced by a new world, one I did not yet understand’ (pg xiii) (Zuboff, (1984))

5. What the Experts say
In this section we report key messages from our conversations with a range of experts

Professor Aaron Sloman is a philosopher and researcher on artificial intelligence and cognitive science. In a recent interview with Aaron Sloman he discussed his views and thinking about computational thinking and computer science in schools.

When exploring why computational thinking is important Aaron states “Computational thinking is one of the required approaches to understanding the universe, which was not available to the deepest thinkers in most of the history of science, mathematics, engineering and philosophy.”

When exploring how we might understand what is happening when a learner is thinking computationally and why this is so important Aaron draws on the work of Karmiloff-Smith representational re-description to explain the idea of abstraction and learning that is taking place.

The learning “starts off empirical, play with things space and structure orderings and so on and then something can happen that transforms your understanding so that it becomes generative. So that things previously that you could only discover empirically by trying them out when you have had that transformation you can work them out and prove them. You don’t have to discover them empirically. … [] So someone might, without having to discover empirically work out that the area of an arbitrary triangle is half base times height using that deep generative understanding, which is a form of computational thinking that mathematicians have been doing for a long time and young children are doing while not knowing they are doing it.

And the problem is how to help teachers identify when it’s happening and when transitions are not happening and could happen and how they can facilitate them but for that you have to have a deep theory of about what those transitions are. And I mention Karmiloff-Smith because I haven’t met anyone else who has got a deep theory about what they are and even hers is not yet a deep implementable theory”

So what about the teachers, what needs to happen for them?

“Reaching an understanding of why do teachers need a kind of computational thinking and that maybe their learners don’t during the time they are teaching them but they might later. The kind of computational thinking that says what has to change in the child’s mind for them to absorb this material and what are the different stages that kids can be at and what kinds of things can help them move further. And there may well be amongst the kids at schools those that want to get the generative understanding sooner and others who don’t. They are quite happy to go on acquiring more and more expertise. And find ways on building on those differences might be a very important part of teaching.”
An important part of learning is making mistakes and having some strategies to know how to re-think the problem.

Aaron draws from the work by Sussman the “The Virtuous Nature of Bugs” to explain the importance that errors, bugs and mistakes have in the role of learning. “[ ] this was why when I later started to teach programming […] we built into our teaching the importance of our students getting things wrong.. including us giving them hints saying what will happen if you try so and so. What you call teach files that provide a framework and a problem and let you play with it and we would give them hints that would lead them to get into errors and if they don’t do that they don’t learn that some things work and some things don’t and you can start noticing patterns about the ones that don’t work. So you can save a lot of time not trying things because you recognise that this might be a pattern and this sort of thing is not going to work. But also when things go wrong and you don’t know why you have to develop strategies for finding out why and then debugging and so on. I’m sure good maths teachers having being doing that for centuries.”

And how might successful learning of computational thinking be achieved?

“If we work out how best to stretch the minds of the great thinkers of the future, then we’ll be in a much better position to devise good educational opportunities for the others, in a principled way, by considering what can be left out, or simplified, or presented more slowly, for whom and why.”

Adrian Mee, who works of for the Institute of Education and teaches ICT to postgraduate teachers explains about the computational thinking and computer science in the schools.

“What do we mean by computational thinking? If we are talking about logical thinking yes it does. It has always been part of the curriculum. The problem is if you read a lot of these news articles and the people who maybe experts in computing but they are not curriculum experts. There is a huge amount of confusion between the national curriculum, the program of study, what’s taught in schools, the exam syllabus – all of those things seem to be conflated into the school curriculum.”

What about computer science and the curriculum? Adrian has an interesting challenge for us all to think about “What we are talking about now is the role of computer science in the curriculum and should it be compulsory. This is a crucial issue … there is a thousand and one experts in their field all turning up and saying every child should learn this well my challenge to that is get a ruler and a pencil, piece of A4 paper, draw 5 rows and 5 columns and you will end up with 25 boxes. Anything that is essential for what every child needs to know - try and fit it in those 25 boxes. That’s the essence of the problem. And nobody ever thinks about the grid.”

How do we go about defining the curriculum? There are lots of activities on the internet and new schools and ideas popping up.

“Just a bunch of interested individuals trying to push their particular hobby is no way to define a curriculum. Should they code, should they bake, should they learn to play squash – the answers is probably yes they should learn to do all those things but I am back to my 25 periods per week [5 by 5] grid. And at the moment – what we have got in key stage 3 is 4% of the curriculum. We have got 1 period per week – of the curriculum. And the website I have just looked at they are offering 12 lesson plans – possibly under the assumption that 12 lesson plans equals a curriculum – which it doesn’t – which is what we have had with the CAS group as well – the idea that a bag of worksheets and a website somehow constitutes a curriculum

The economic argument doesn’t have much sway with Adrian. He reminds us of the role of the curriculum.

“There is an assumption that the school curriculum is there to serve the economy. Well nobody seems to go back to the aims, purposes of the national curriculum. Because if you read them it
'What the Research Says’ Briefing 2
27th April, 2012

doesn’t mention about training people for the world of work. It’s mentioned as it should be but that curriculum is meant to be a joint enterprise – it is meant to serve the needs of the individual and it is meant to serve the needs of the community as well – it’s both an individual issue and it’s a joint national enterprise.”

Under what rational are people thinking to make the change asks Adrian Mee.
“What it is not is a vehicle for people to push their particular views. Which it has always been – for two centuries any kind of curriculum has been a confluence of political ideologies, industrialists wanting pre-trained labour – I mean that has always been there. I think we need to realise that is a factor of what we are talking about... Computer science in schools and the rest of it. So this idea of why you are doing it is an issue. For instance there is a shortage of programmers in the games industry therefore 8 million children should learn to be games programmers? Well you know there is a shortage of top-level chefs. There’s a massive shortage of plumbers...

What do we need to do now?
“We need to take the lid off the networks because it is the networks that is the technology they are dealing with - not the PC – it is what is connected to what – it’s that critical judgement about what should be happening – that’s not a massive step forward for a curriculum.”

“[For] the curriculum what we need is a way of engaging with kids with technologies they use all the time and to develop their capacity to talk about the meaning of the technologies, about values – how it works in the broad sense.[..] What I’m heading towards here is that philosophical background of computing – which to be fair that was what Alan Turing was interested in – He wasn’t a bits and bites man – computational logic –but some of the ethical issues that drove him for obvious reasons – so it is something that has being on-going and that is a hard thing to do – to develop that kind of critical thinking skill about technology in schools. ‘cos schools are all about being told stuff.”

Professor Peter McOwan of Computer Science in the School of Electronic Engineering and Computer Science at Queen Mary shares his insight and deep thinking about computational thinking. He draws from his interdisciplinary experiences and research interests from visual perception, mathematical models for visual processing, in particular motion, cognitive science and biologically inspired hardware.

Peter explains why computational thinking is a very important way of thinking that we all need. “It is important because it is a way that you write computer programs and I think computer programs are very important. They underpin every part of work life and play in the world today. So having someone who can think computationally and translate that into computer programs is very important. But I think that if you are not writing a computer program then it is a very useful kind of abstraction to a series of generic problems and it is a way of understanding the world and something about understanding the mechanisms that make the world work and being able to take difficult problems and being able to break them down into kind of simpler kind of components.”

Computational thinking and computer science go hand in hand as a critical both a way of thinking and understanding the world we live in. As Peter so succinctly and clearly explains about why we cannot ignore computer science.

“It underpins the entire world around us. Everything you do, say, speaking on the phone, the machines that go ping in the hospital, everything there’s computer sciences, where you are taking information and you are processing it in someway and somebody has written the program to do that and therefore they have looked at a problem of information processing and come up with a way of abstracting that to blocks and tools to computational thinking”

The invisible becomes visible is very powerful and augmented reality has brought us new ways of thinking and inspecting the world around us. Peter explains the next stage.
'What the Research Says’ Briefing 2
27\textsuperscript{th} April, 2012

“We've got to the stage now where human evolution isn't a fight for a survival hand to mouth but the next stage of evolution are going to be technologically supported where we create new human existence. [] What are the new paradigms? What are the new ways we will take all of this information – and do useful and exciting things with it? … Augmented reality is another classic example of that where you create world overlays. Often the data that is being collected both from the visible which is what you can see plus the invisible – which is all the other data – that is soaking the environment.”

Creativity, lateral thinking and communicating deep and powerful ideas are the result of computational thinking.

“Computer science is absolutely creative. [...] I have had a BAFTA in my career because of computer science. To be involved in that – that is what creativity is. And it's about being able to suddenly sculpt – create – beautiful things out of digital information. And that can be as beautiful and as creative as painting on a canvas. [...] Lateral thinking and creativity are similar to one another. I had a discussion with an artist who was saying that computers can never be creative but I think you can be quite creative if you approach it from a computational thinking point of view. […]”

He draws from his experience in understanding gravity during a philosophy discussion. Peter explains “And for them their kind of computational model for gravity was very much the idea of the planets with elliptical orbits. Whereas for me it was bendy rubber sheets with ball bearings in it because of the curvature of space, time and general relativity. [...] I can see how to explain the transition between their model of the universe and my model of the universe, rather than seeing them as two separate things there is a continuum link in my brain that says 'how to go from here to there' and that is actually the step by step communication part. That is part of what the computational thinking is about because you are breaking down that communication into the steps.”

The breadth of thinking and understanding by all the experts brings us to the importance of the philosophical concerns of the consequences of all that we do and how computational thinking can by empowering help us to understand both the nature and the impact of our decisions. Here Peter’s thinking and perspective on this brings up very important questions.

“Any technology is morally neutral it is what you decide to do with it that makes it good or bad. A lot of the work I do in going out and talking to the schools and writing articles about computing [what I want to do] is stimulate a debate where people start to say what’s a good idea and what’s not such a good idea and hopefully make sure that the scientists of the future start to think through these kind of ethical dilemmas before they go down into it. [From ] the computational thinking point of view for ethical dilemmas -if such and such a thing happens - then these are the possible consequences to think about. This logical chain reaction of events that could be produced, which if you only see the first couple of steps in the algorithm is a problem – a lot of scientists are only focussed on the first step of the algorithm because that is where the challenge is without realising that unlocks a series of other potential domino effects and they should be at least thinking about those as well.”

6. What next?

This briefing paper has been developed to prompt thinking and debate at an event at the London Knowledge Lab on 27 April, 2012. The debate event will be recorded and will be combined with a more comprehensive review to produce an extended paper, available in June 2012 that will be available through the London Knowledge Lab.
Appendices

Appendix 1: Tools available for learners today

The language resources listed below and the list of tools in appendix 2 are taken directly from the following source http://regulargeek.com/2011/07/20/36-resources-to-help-you-teach-kids-programming/

There are many other programming languages and tools on the web that can also be used and this just provides a quick overview of what is available.

Languages for programming

- **Lisp** and **Scheme** are used heavily in research oriented universities when teaching an introduction to programming and the theory of programming. There used to be a recommended tool called DrScheme which has now morphed into **Racket**.

- **Java** not recommended for children given the extent of the libraries and the difficulties of teaching object oriented programming. However, given the popularity of the language, there are sites targeted to younger developers and tools like **Greenfoot** and **BlueJ** to make learning easier.

- **C/C++** is probably one of the more difficult paths to take. There are limited resources for teaching children, but given the continued popularity of the languages they need to be considered.

- **C#** would be an interesting choice when you include the .NET libraries. Like C++, there are not a lot of resources but Microsoft does have one guide for “Sharp Kids”. With .NET, you could focus on building web applications which could add to the “interestingness” of the education.

- **Visual Basic** is an easy language for people to learn, given its BASIC roots, and adding .NET give it the same benefits as C#. Again, Microsoft has a guide for “Very Bright Kids”.

- **Smalltalk** and its variants like **Squeak** continue to be recommended when people ask about learning programming. The most recommended tutorial is now called **Bots Inc** that is based on the book Squeak: Learn Programming With Robots.

- **BASIC** It is still in use, and Microsoft offers Small Basic for beginning developers.

- **Ruby** has a solid amount of resources for teaching kids, especially when compared to other popular languages. **Ruby for kids** and RailsBridge are good options to review.

- **Python** is another popular language that seems to be recommended to younger developers. The most popular recommendation by far was **LiveWires**.

- **PHP** was not recommended too often, but is included given that it is easy to create a simple website. Keeping kids interested is difficult, and the feedback of quickly building a website could prove beneficial. There is a PHP For Kids tutorial as well.

- **HTML** and **JavaScript** are included mainly to give kids some basic programming introduction, and to allow them to build a website. There is an interactive environment called **Waterbear**, and plenty of information at HTMLIsEasy.com.

Other languages that were mentioned were ML, Prolog, Haskell and REBOL. These are not nearly as popular as the other languages, and in some cases may introduce some difficulty in teaching because of the typical lack of knowledge that people have of these languages.
Appendix 2: Learning Environments And Other Things

In addition to the traditional languages above, there are websites and interactive environments that focus on teaching children. Some of these resources are just websites, but others are interactive learning and development applications.

**MSDN Development for Beginners, Kids Corner** is an interesting resource from Microsoft for teaching children. Some of the tutorials mentioned above are from this MSDN resource. Teach Kids to Program is a site focused on teaching kids programming using games.

**Alice** is one of the most highly recommended programming environments for kids. There is a Storytelling version that is geared towards middle school education as well as the base version for high school and college.

**Logo** is one of the oldest teaching languages and it has evolved into many things. There are Windows versions, Mac versions and plenty of tutorials available through some simple Google searching. LCSI Microworlds is one of the more advanced (and commercial) options available.

**Phrogram** (previously known as KPL) is another often recommended environment. It has grown from being child-focused to being a general purpose environment based on .NET.

**Scratch** is a programming language developed by the Lifelong Kindergarten Group at the MIT Media Lab that allows you to create interactive stories, animations and games. This is another popular recommendation with a lot of information and tutorials available.

**Colobot** (Colonize With Bots) sounds like another interesting option. The Wikipedia page explains the education portion of the game, “The main feature of the game which makes it educational is the ability of the player to program his robots using a programming language similar to C++ or Java.”

**Terrarium** is another graphical environment and game, which was developed by the .NET team. It is not focused on children, but it is considered a learning environment for .NET.

**Lego Mindstorms** are robots that have interfaces and projects in various languages. This is a cool concept because you can easily craft a program to control the robot. It definitely keeps people interested, and there is a ton of information available.

**Stagecast Creator** was recommended several times, specifically when targeting younger, preteen children. The biggest reason kids will like it is that they can build games using the environment.

**ToonTalk**, “Making programming child’s play”, is an animated world where kids solve challenging questions. Interestingly, it is one of the few resources translated into several languages.

**GameMaker** is not targeted towards children, but is a game building application that does not require you to write code. Given the non-code basis, it was recommend several times as a good way of learning programming while keeping kids interested.

Interactive fiction **Inform**, the most popular interactive fiction language, is still being used. It is based on natural language, so it could be easier to learn than other code based languages.
'What the Research Says’ Briefing 2
27th April, 2012

References

Annette Karmiloff-Smith http://www.bbk.ac.uk/psychology/our-staff/academic/annette-karmiloff-smith
Livingston, I. and Hope A. (2011) Next Gen: Transforming the UK into the world’s leading talent hub for the video games and visual effects industries. London: Nesta.
Naace “The Importance of Technology”, http://www.naace.co.uk/1778

This paper has been written by
Dr. Patricia Charlton and Professor Rosemary Luckin, The London Knowledge Lab, Institute of Education

It is our intention to motivate discussion and to combine the findings from our research for this paper with the outcomes of the workshop at the London Knowledge Lab on 27th April to produce a report in June 2012. We are grateful to many people for their input to this paper, in particular:
Professor Richard Noss, Director Technology Enhance Learning research programme, The London Knowledge Lab, Institute of Education.
Adrian Mee, Institute of Education.
Aaron Sloman, University of Birmingham
Andrew Manches, The London Knowledge Lab, Institute of Education.
Michael Young, The London Knowledge Lab, Institute of Education.
This briefing was authored with support from the Technology Enhance Learning research programme and the Engineering and Physical Sciences research Council. For further information please contact r.luckin@ioe.ac.uk