If this is the Second Coming of Coding Will There Be Rapture or Rejection?

Peter R. Albion

University of Southern Queensland, Toowoomba

Coding made the national headlines in 2015 and appears to have garnered sufficient support from politicians to be seen as an essential component of education for all. Educators with longer memories will recognise that this is not the first but the second coming of coding for all as a focus of education in technologies. The first coming ended in rejection because too few teachers really understood the potential. The immediate response to the second coming appears to be rapture but it could easily end in rejection. How do we develop a sane response to the current impetus for coding and sustain it? The answer lies in preparing teachers with sufficient knowledge of coding and computational thinking for it to be authentically useful in their own lives. Only then will they appreciate its value for learners in their classrooms.

Introduction

Coding made it into the national headlines in 2015 when Federal Opposition Leader, Bill Shorten, asked then Prime Minister, Tony Abbott, whether he would "support coding being taught in every primary and secondary school." The then Prime Minister initially derided the idea with a comment about kids going to work as coders at age 11 but later confirmed that the Government was already supporting the concept in the national curriculum (Bagshaw, 2015). A few months later, the Queensland Government launched Advancing Education, described as an action plan for education in Queensland and featuring coding as a key component marked by the hashtag –

#codingcounts (DET, 2015a). The website noted the highlights of Advancing Education as fast-tracking of the Digital

Technologies subject from the new *Australian Curriculum: Technologies* (ACARA, 2015) beginning in 2016, creation of a coding academy, and incubation of future entrepreneurs. Robotics was proposed as a key component and professional development was to be provided for teachers. These moves are part of a wider embrace of STEM (Science, Technology, Engineering, and Mathematics) education as critical to our national future.

Although 'coding' has appeared more often in political discourse, the *Australian Curriculum: Technologies* (ACARA, 2015) uses 'program' or 'programming' as frequently as it does 'code' or 'coding' when referring to the techniques used to control digital technologies. Among those who work with digital technologies, 'programming' is understood to refer to the activities involved in specifying the logic involved in the solutions to problems whereas 'coding' refers to the process of expressing that logic using a language appropriate to the task and the technologies in use. A recent European report about programming and coding in school curricula (Balanskat & Engelhardt, 2015) acknowledged that coding can be viewed as a subtask of programming in which an algorithm is expressed in a target programming language but opted to use the terms interchangeably in the report. The same report also acknowledged the importance of 'computational thinking' (Wing, 2006) which is one of the key ideas in the *Australian Curriculum: Technologies* and a starting point for being "more effective in an increasingly computed society" (Booch, 2014, p. 11). Computational thinking, programming and coding are related ideas and all are important for the process of solving problems and expressing the solutions in digital technologies. Perhaps because of its role in expressing ideas in a language similarly to general literacy, coding has become most visible in public discourse about education.

Various opinions have been expressed about coding in Australian schools in the past year or two. In close proximity to a call by the Prime Minister for improved teaching of STEM subjects, especially coding, outgoing Chief Scientist of Australia, Ian Chubb, suggested that primary teachers graduating from teacher education programs are not sufficiently well prepared to teach coding (Dodd, 2015). As remedies he suggested "attracting higher quality students into primary teaching courses, boosting science, technology and maths courses in teaching training, improving professional development and putting specialist STEM teachers into schools to mentor others." Effective as these might be in the medium to long term if they could be funded and managed, they are not short term solutions.

Recent University of Southern Queensland graduate, Elke Schneider, has blogged about her experiences with teaching digital technologies. She was enthusiastic about the recognition of the importance of digital technologies but expressed misgivings about how the push for teaching coding could be implemented, noting that many teachers were concerned about learning to code themselves before attempting to teach it. She questioned whether pushing coding into schools without adequately preparing teachers might turn kids off rather than on (Schneider, 2015).

On the World Bank site, Michael Trucano described initiatives in Europe and elsewhere to introduce coding into classrooms and cited the aforementioned European report (Balanskat & Engelhardt, 2015) highlighting some of the related pedagogical challenges as well as the challenges of professional development for teachers. He listed arguments commonly made for coding, including future employment and development of logic and problem- solving skills. He concluded by suggesting that we should be thinking at least as much about coding to learn as learning to code, arguing for the place of coding across the curriculum where it can be applied in support of varied topics (Trocano, 2015).

Elsewhere, the Chief Scientist has noted that, instead of 'future-proofing', humanity would be better served by 'future-priming', through developing the capabilities needed to benefit from technological change (Chubb, 2015). Those comments were made in the context of launching a report that argued that all Australians, not just those destined for careers in STEM specialisations, require sufficient knowledge of STEM to engage in related debates and decision making (Williamson, Raghnaill, Douglas, & Sanchez, 2015). As presented in the *Australian Curriculum: Technologies*, which includes computational thinking among its key ideas and explicitly mentions coding in its content descriptions, and in the Queensland Government initiatives, STEM education inevitably includes coding and teachers will be required to learn and teach it.

There is no doubt that STEM education and coding as an element of the new *Australian Curriculum: Technologies* (ACARA, 2015) are hot topics in Australia and more widely. As has been noted above, there are varying opinions about why coding should be taught and learned in schools. There are also doubts about the capability of teachers to teach it, at least without extensive professional development. Hence there are good reasons to think carefully about the issues. This paper attempts to recall some of the past history of coding in schools and to consider what lessons might be learned from that history.

A short history of coding in Queensland schools

Those few teachers who have been around computing in schools since the 1980s will recognise that this surge of interest in coding for all in classrooms is at least its second coming. It may be useful to recall some of that history for the benefit of those too young to remember it.

Coding appeared in Queensland secondary school classrooms when computing was first included as an option for study in the mathematics curricula of the mid-1970s but, for the first several years at least, few schools had direct access to computers so the process of learning to code was slow. If schools elected to teach programming, the common approach was for students to code their simple programs on mark sense or punch cards and send those to a computer centre to be executed and the printed output returned. For rural schools that required the cards to be sent by bus or post to Brisbane State High School for execution. The cycle from recording code on cards until the printed output was returned required 3 days to a week. If there was an error in the code, then correcting it and obtaining new output required another cycle of several days.

Such a long cycle was untenable for teaching because it allowed limited scope for development of more complex programs through practice and left long periods in classes to be filled with more theoretical activities that were less motivating to many students. It was not long before schools began to adopt the early programmable calculators that were becoming available around that time. Despite limitations for data entry and storage and the absence of standard programming languages, it was possible to develop programs using the standard algorithmic structures of sequence, selection and iteration. The opportunity to rapidly plan, code and test solutions within the space of a single lesson in a classroom rather than waiting days for output was more motivational for all and allowed development of more complex programs within the space of a semester.

The next major advance was the appearance of the first personal computers in schools in the late 1970s. The Apple][, Commodore PET, Tandy TRS80, and Ohio Scientific were among the first computers to find their way into schools. Because there was very little commercial software available in those first years, the major application for microcomputers in schools was for programming in the mathematics curriculum. For the first few years of the 1980s the number of microcomputers in Queensland schools was few enough that the Department of Education was able to produce a brief annual report that detailed the microcomputers in schools across Queensland and their uses.

The first coming of coding for all

Although the Logo computing language was created in 1967 and was used experimentally in schools from that time, its popularisation followed the emergence of personal computers in the late 1970s (Logo Foundation, 2015). A version was created for the Apple][computer at that time. Following publication of *Mindstorms: Children, Computers, and Powerful Ideas* (Papert, 1980), Logo became more widely known. By the mid-1980s, as computers became more widely available in Queensland schools, there were professional development activities across the state to introduce teachers to programming with Logo with the expectation that it would be widely adopted in schools at all levels.

Papert (1980) described scenarios in which computers, using behaviourist principles embodied in programmed instruction, would be used to teach children. He regarded that approach as undesirable and noted that one of the best ways to clarify our understanding of something is to explain it to somebody else. Hence, he argued that it would be preferable for children to teach the computer rather than be taught by it. That is, the process of analysing some process and expressing it in instructions for a computer would be an excellent way of clarifying and demonstrating understanding.

The utility of teaching as a path to learning is evident to anybody who has ever tried to explain something of even moderate complexity to somebody with only a limited grasp of the necessary mental building blocks. Teachers, parents, sports coaches, and students working together in study groups are all familiar with how trying to explain something to somebody else works to clarify understanding for the explainer at least as much as the explainee. A computer is the ultimate patient learner and the process of deconstructing some process and expressing it in the limited vocabulary of a programming language is an excellent way to clarify understanding.

Papert's own educational views, developed during his earlier work with Piaget, favoured a constructivist approach in which children would learn from experience. He pointed to the way in which children learn language and other common knowledge through being immersed in the world and recounted his own experience of learning about ratio through an early fascination with systems of gears. Based on his observation and experience Papert argued that, just as children learn English or another language by living in a world where that language is spoken, the best way for children to learn mathematics would be to be in 'mathland'. He developed the idea of microworlds, simplified but realistic representations of some system, for learning through experience and extended the ideas of constructivism into constructionism in which learning is demonstrated by constructing and sharing some artefact (Papert, 1980).

The Logo computer programming language was developed specifically for education. It was designed to be easily accessible at a basic level so that young children could quickly learn the rudiments but rich enough to support more advanced programming. It was used successfully with very young children and to teach university computer science courses (Logo Foundation, 2015). Early in the history of Logo, Papert's group devised and popularized the 'turtle' which was originally a robotic device that could be driven around on the floor under computer control and was capable of raising and lowering a pen to trace its path. Instructing the turtle to move so as to trace out particular patterns helped make simple programming concepts concrete and provided a microworld in which learning geometrical concepts of distance and angle was a natural consequence of trying to program the turtle to achieve some tangible movement goal. The mechanical turtle was expensive so most implementations of Logo in classrooms were restricted to using the same instructions to drive a 'turtle' around a computer screen. The 'screen turtle' was the most common experience of Logo for most teachers and children.

Logo continued to be developed and extended in various versions though the 1980s and 1990s. There were multiple versions of the language with graphic and other extensions, including interfaces to support control of Lego and other systems. Nevertheless, within a decade or so Logo had all but disappeared from most school classrooms. There were a few schools, including some of the early adopters of laptop programs, that made versions of Logo a feature for a time but, as computers became more common in business and homes, the focus of computing in schools mostly shifted to standard office applications and the World Wide Web.

Implementation of Logo in school classrooms attracted research interest that demonstrated benefits for learning, especially in mathematics (Clements, 1987; Kurland & Pea, 1985; Hoyles & Sutherland, 1987). There seems to be no definitive explanation for the disappearance of Logo from most classrooms despite the apparent benefits for learning. Very likely the reason was that typical classroom applications seldom progressed beyond drawing simple geometric shapes on the screen or producing short animations. The likely cause of that may well have been that few teachers had any real concept of what else might be possible with the tools at their disposal. The first coming of coding for all in the general school classroom concluded almost as quickly as it had begun.

The second coming of coding for all

Programming continued and expanded in schools over the past couple of decades but that activity was very largely in secondary schools and, with few exceptions, involved a specialised subject taken by a minority of students. Beyond the initial burst in the 1980s, there has been no widespread adoption of programming or coding as part of general education in either secondary or primary schools and no apparent impetus for such adoption, until recently.

Over the past five years or so interest in computer science and related topics in schools has resurfaced around the world. As described above, the terminology varies – programming, coding, computational thinking – but the same broad ideas are in play. The shift has not been sudden; there were hints of it in Friedman's (2006) discussion about what is needed in education for the flattened world. He argued that the increasingly interdisciplinary nature of problems and their solutions would require that workers in a variety of fields have at least some knowledge of coding but it is doubtful many have heeded that call. More significant drivers for renewed interest in coding have included corporations concerned about where they will find the talent to spearhead their next advances in computing and governments concerned about the future of their economies. It is those factors that are at work in Australia and driving the responses from governments recounted in the introduction.

The Digital Technologies subject in the new *Australian Curriculum: Technologies* (ACARA, 2015) has highlighted computational thinking as a key idea and presented some elements of programming and coding as core to a general education. The review of the national curriculum (Donnelly & Wiltshire, 2014) expressed scepticism about the proposals for digital technologies, apparently on the basis of suggesting that other countries are not doing it and Australia ought not lead the charge. Contrary to those claims by Donnelly and Wiltshire, there have been visible moves in the UK and elsewhere in recent years to embed coding and related material in school curricula (Balanskat & Engelhardt, 2015). The exchange in Federal parliament cited above suggests that our politicians recognise that it is an idea whose time has come but there continues to be public debate about the possible place of coding in schools.

There have been voices raised in support of an introduction to coding as part of a general education. Sterling (2015) is one who argues that the centrality of coding to modern technology makes it important for all to understand the possibilities even if we have no expectation that all students will have careers in coding any more than we expect that all will become artists as a result of studying The Arts in school. His position is actually less in favour of children using technology in classrooms than of their developing some understanding of the principles of computational thinking upon which the technologies depend. Indeed, he expressed outright opposition to compulsory laptop programs in schools, arguing that widespread access to mobile phones provides students with the necessary exposure to technology. Trying to rebottle the genie of (laptop) computers (or tablets and smartphones) in the classroom is probably a lost cause but understanding something of how things work and the potential for new applications is an important addition to being capable users of existing applications.

At the other end of the spectrum there are arguments that a little bit of coding in classrooms may be a dangerous thing (Merkel & McNamara, 2015). Their argument is that simplistic approaches to coding misrepresent the nature of information technology by encouraging ad hoc tinkering. The reality is that information technology depends upon the systematic approaches of software engineering implemented through team work. Their assessment of the proposed digital technologies subject is that the anticipated outcomes would challenge some students in university programs. The implication is that the curriculum is a futile venture because it demands too much of learners and teachers.

Interestingly, with the arguable exception of Kevin Donnelly, none of the voices cited above could claim much familiarity with school education, let alone the challenges of teaching computing to primary school children. Stuckey (2015) has voiced an opinion from an educational perspective in a blog post and admitted, in response to a comment, to having taught Logo in the mathematics curriculum when she was teaching in schools. She was not convinced that coding is the new literacy, preferring science for that role with coding included under its umbrella. She recalled that she learned to code in BASIC, a common activity in the early days of personal computing when there was little or no software other than what one could code, mostly by copying code sourced from magazines. She also noted that she no longer used that knowledge and knew few or no people who have any use for coding in their daily interactions with computers. She suggested that coding may be best learned outside of class time though the logic and computational thinking that underpin it may be more vital in general education. There are many things taught and learned in schools that have limited everyday utility for most people – trigonometry, calculus, atomic structure, geography of foreign lands, European history, to name a few. Nevertheless, we probably do not need to add to the list and should take care that new additions to the curriculum seem relevant. Certainly we want teachers to see the value of the curriculum even if it is not immediately apparent to the children in their classes.

The *Australian Curriculum: Technologies* (ACARA, 2015) specifies the creation of visual programs and refers to the use of a visual programming language. The most widely known language of that kind and the one most likely to be used in addressing those aspects of the curriculum is Scratch (scratch.mit.edu). Ironically Scratch has been developed out of the same laboratory that Papert established and led at MIT. It lends itself to the creation of animations and games that children will likely find motivating and the use of drag and drop pieces to construct simple programs makes it even easier to get started with Scratch than with Logo and guards against syntax errors although it remains possible to commit logic errors. The Scratch website supports an active community of users and provides for sharing of projects that can be copied and modified by other users or dissected in order to learn how particular effects were achieved. The website claims to have more than 12 million projects shared by users.

How far Scratch, or any of the similar languages that have appeared, can be pushed toward developing 'useful' programs is not entirely clear. Some variants are extensible and the RALfie project (ralfie.org) has developed some extensions to one variant, Snap! (snap.berkeley.edu), that enable remote control of Lego and other systems (Kist et al. 2016). It seems likely that most teachers, and learners, will find Scratch and its variants similar to Logo in what it can do and for that reason it is at risk of meeting a similar fate. The most obvious projects involve simple animations and games that may demonstrate knowledge of programming and other subject matter but have no obvious utility in the real world. Enthusiasts may persevere but the already crowded curriculum will squeeze out all but the most necessary and minimal efforts at coding in many classrooms.

Neither rapture nor rejection is a sane response

How should educators respond to this second coming of coding in the schools? Is this the long awaited rapture of computing in the classroom or will it end in another round of rejection? How should we be responding to this opportunity?

Neither unbridled rapture nor outright rejection would be a sane response to the second coming of coding for all. A sane response will be carefully considered and measured rather than driven by panic in the face of technological change.

A rapturous response would accept without question the claim that "coding is the new literacy" (DET, 2015b, p. 5) as though there is a single coding language that will serve all purposes when a more significant new literacy is the critical literacy to assess inflated claims. It would also signal acceptance that hopes for future employment rest upon ability to code when there are many people already working in industries built upon coding who do not code and no likelihood that will change. Coding is important but it is not the single magic bullet that will save our national economy and set us all upon the path to prosperity.

Rejection might be based on the optimistic view that we are producing, and will continue to produce, sufficient people with coding skills to ensure continued development in vital industries so there is no need for all to learn coding. Alternatively, it might be based on a pessimistic view that too few teachers are prepared to teach coding to all so there is no point in trying to do something that is likely to be difficult and probably has little value.

A sane response will fall between those extremes. It will recognise that society needs skilled coders but that there are other roles, even in the development of software, that require instead capabilities in visual design, process analysis or other fields that contribute to software development. It will also recognise that some understanding of coding, and more particularly of the computational thinking that underpins it, will be invaluable for all as a basis for understanding the risks and benefits of committing our lives to code in autonomous vehicles or elsewhere and for recognising problems that may, or may not, be susceptible to coded solutions. Coding as literacy in that sense of reading and responding to the world more effectively does have value for all.

Understanding the value of coding in everyday life is essential if we are to seriously promote coding for all as a core element of the school curriculum. Unless the relevance and value of young Australians learning something about coding is clear to teachers, parents, politicians, and the learners themselves it is unlikely to persist in the curriculum beyond the first flush of enthusiasm. What then is the value of coding for the average person?

The value of learning to code

I have some empathy with some of the views expressed by Stuckey (2015) who wrote “Many years ago I learned BASIC, which is a simple coding language. I never use it now but I still appreciate what learning it taught me about the logic and computational thinking that goes into programming.” In the early 1970s I learned some coding through self-education on time share computing systems and taught some in mathematics classes in the later 1970s. I was an early programming hobbyist, buying my first Apple][computer in 1980, and progressed to writing programs to assist with various school administration tasks. Eventually I acquired a Graduate Diploma in Applied Computing and taught Information Processing and Technology in secondary schools before moving to the university. Like Stuckey (2015), I have some knowledge of programming and I do occasionally write code to complete work or personal tasks but most days I am not coding.

However, I am less confident about agreeing with Stuckey (2015) when she wrote “Coding really isn't an everyday practice in the way reading and maths are...it's probably more like understanding a car engine...if you basically understand how a car works you can drive and maintain it efficiently and effectively.” The residue of my coding experience does have benefits in my application of computers in day-to-day life. A large part of that residue is probably the logic and computational thinking that Stuckey wrote about. Having some knowledge of programming helps me to see and follow the logic of unfamiliar applications. It also raises my expectations about all aspects of design in digital technologies and makes me more critical of applications and websites than some of my non-coding colleagues who have no reason to expect that systems should be more usable than they are. My capacity to analyse things in ways that lead me to construct spreadsheets or other tools to manage routine tasks is also enhanced and I am more likely than my colleagues to use tools like auto-text expansions and keyboard macros to automate repetitive tasks. On balance I think that having learned coding has developed my capacity for logical and computational thinking in ways that enhance my day-to-day use of digital technologies and that draw upon that knowledge of coding in meaningful ways.

For example, in recent days my teaching activity required me to assign content descriptions from the *Australian Curriculum: Technologies* to 180 students in my Semester 1 class and advise them of their assignments. I scraped the content descriptions from the ACARA website, pasted them in a text editor, and used search and replace to get them into a tab-delimited list that I could paste in Excel. I scraped student details from the participants list in Moodle and pasted that in another sheet in the same Excel file. There I wrote formulas to separate first and last names and then pasted content description codes against the names to register the assignment of content descriptions. I imported the student list into Filemaker Pro and then imported the list of codes and content descriptions and made the relational links. I was then able to create a simple script to send personalised email messages advising students of their assignments with codes and content descriptions. The limited coding required was facilitated by the Filemaker scripting system that is based on selecting from a list of available commands and setting relevant parameters, similar in some ways to a visual programming language in that there is no need to write code from scratch. Although there was no real coding required, thinking through the process and creating the necessary formulas and logic for the script drew upon my computational thinking skills developed through learning to program and code. Our institutional systems offer no means to achieve that outcome, other than by tedious manual processes, and I doubt that many of my colleagues could accomplish the same result. The power of computational thinking and some knowledge of coding is evident to me and I can see that even where somebody might not do work themselves understanding the potential offers power that is not otherwise available.

I could provide numerous other examples of where some knowledge of coding or computational thinking has enabled actions that would be impossible without such knowledge. Even where it is not possible for reason of secured access or limited skills to effect a solution directly, understanding the potential opens up possibilities for talking with those who have the necessary access and skills. Learning to code and to exercise related computational thinking can be beneficial even when the relevant skills are not applied directly. Such knowledge also provides the background from which to assess existing solutions and to make informed judgements about their suitability for particular purposes and the potential for improvement. That knowledge and related skills are sorely needed in a society where we otherwise have to accept whatever inadequate systems are foisted upon us. As was argued by Wing (2006), computational thinking is a “universally applicable attitude and skill set” (p. 33) that should be part of a current day education for all.

A way forward

The children now in schools are so-called 'digital natives' who, compared to older 'digital immigrants', are supposed to possess superior capabilities for working with digital technologies by virtue of having grown up with them (Prensky, 2001). That characterisation has been thoroughly discredited (Bennett, Maton, & Kervin, 2008); simple observation of younger people working with technologies should be sufficient to refute the idea of the 'digital native' as a generalisation. Many of them are adept with some common technologies and a few have extensive skills but for most of them confidence and facility with social media is a superficial veneer and there is little depth to their knowledge of the technologies they use. That is logical when we realise that the speed of technological development means that nobody is ever really a 'digital native' who grew up with current technologies. The pace of technological changes means that we are all perpetually 'digital immigrants' living by our wits with constantly evolving technologies.

The role of general education is not to prepare students to take up careers that may disappear before they enter the workforce but to develop the capability to live by their wits, manifesting resilience in the face of the only real constant, namely change. The native/immigrant dichotomy with respect to digital technologies has never really been valid (Bennett, Maton, & Kervin, 2008). The analogy of residents and visitors (Jones, 2011; White & Le Cornu, 2011) is probably more apt, better reflecting the differences between those who stay long enough in any context to develop some degree of comfort and those who do not stay long enough to care or really need to know. So far as coding or programming is concerned we need sufficient people with specialised skills to build the digital worlds that we will inhabit but not everybody needs to be a builder. For most of us it will be sufficient to understand the possibilities well enough to have a sensible conversation with the architect or builder or perhaps to have sufficient skill to be 'digital renovators' (Jones, 2011), able to adapt what we find to better suit our needs. Mere coding, the ability to express a given solution in a language that can be interpreted by a computer, will not be sufficient to achieve that. The underlying skills of computational thinking (Wing, 2006) will be much more useful as the basis for analysing problems and developing solutions that may ultimately require coding for their implementation. It is there that we need to focus and the current discourse about coding may be part of raising awareness of that broader goal.

If we interpret the *Australian Curriculum: Technologies* (ACARA, 2015) appropriately it is not about turning every child into a programmer or software engineer. It is about developing computational, design, and systems thinking as means toward creating preferred futures. For that to happen tools like Scratch can provide a useful starting point but we will need to move beyond that to engage students in designing and developing solutions to real problems that require computational thinking. Teachers who have limited experience of engaging in 'digital renovation' to adapt their own digital environments to be more suitable may struggle to imagine and implement suitably authentic learning activities (Lankshear, Snyder, & Green, 2000). Our real challenge may be in assisting teacher colleagues to develop the necessary attitudes and aptitudes to take charge of the digital technologies in their lives and prepare their students to do likewise. If we can do that we may not achieve rapture but may at least maintain sufficient enthusiasm for the second coming of coding to avoid rejection.

References

ACARA. (2015). *Australian Curriculum: Technologies*. Retrieved from http://www.australiancurriculum.edu.au/technologies/

Bagshaw, E. (2015, 29 May). Tony Abbott ridicules his own party in school coding gaffe, *The Sydney Morning Herald*. Retrieved from http://www.smh.com.au/national/education/tony-abbott-ridicules-his-own-party-in-school-coding- gaffe-20150528-ghbdal.html

Balanskat, A., & Engelhardt, K. (2015). *Computing our future: Computer programming and coding*. Brussels: European Schoolnet.

Bennett, S., Maton, K., & Kervin, L. (2008). The 'digital natives' debate: A critical review of the evidence. *British Journal of Educational Technology, 39*(5), 775-786. doi: 10.1111/j.1467-8535.2007.00793.x

Booch, G. (2014). To code or not to code, that is the question. *IEEE Software, 31*(5), 9-11. doi: 10.1109/MS.2014.128 Chubb, I. (2015). 'Technology and Australia's Future' report launch, from http://www.chiefscientist.gov.au/2015/09/speech-

technology-and-australias-future-report-launch/Clements, D. H. (1987). Longitudinal study of the effects of Logo programming on cognitive abilities and achievement.

Journal of Educational Computing Research, 3(1), 73-94.DET. (2015a). Advancing education: An action plan for education in Queensland. Brisbane: The State of Queensland

(Department of Education and Training) Retrieved from http://advancingeducation.qld.gov.au/.

DET. (2015b). #codingcounts – A discussion paper on coding and robotics in Queensland schools. Brisbane: The State of Queensland (Department of Education and Training) Retrieved from http://advancingeducation.qld.gov.au/SiteCollectionDocuments/Coding-and-robotics-booklet.pdf.

Dodd, T. (2015, 12 December). Are primary teachers ready to give coding lessons? Ian Chubb says no, *The Australian Financial Review*. Retrieved from http://www.afr.com/news/policy/education/are-teachers-ready-to-teach-coding- ian-chubb-doesnt-think-so-20151210-glkibe

Donnelly, K., & Wiltshire, K. (2014). *Review of the Australian Curriculum: Final Report.* Canberra: Commonwealth of Australia. Retrieved from https://docs.education.gov.au/documents/review-australian-curriculum-final-report

Friedman, T. L. (2006). *The World is Flat: The Globalized World in the Twenty-First Century*. London: Penguin.Hoyles, C., & Sutherland, R. (1987). Ways of learning in a computer-based environment: some findings of the LOGO Maths

Project. *Journal of Computer Assisted Learning, 3*(2), 67-80.Jones, D. T. (2011). Residents and visitors, are builders the forgotten category? Retrieved from

https://davidtjones.wordpress.com/2011/07/31/residents-and-visitors-are-builders-the-forgotten-category/

Kist, A., Maiti, A., Maxwell, A., Orwin, L., Ting, W., Albion, P., & Burtenshaw, R. (2016). Hosting and Sharing Your Own Remote Experiments with RALfie – an Open Ended Experiment Design Experience. *International Journal of Online Engineering, 12*(4), 40-42. doi: 10.3991/ijoe.v12i04.5101

Kurland, D. M., & Pea, R. D. (1985). Children's mental models of recursive LOGO programs. *Journal of Educational Computing Research, 1*(2), 235-243.

Lankshear, C., Snyder, I., & Green, B. (2000). *Teachers and Technoliteracy: Managing literacy, technology and learning in schools*. Sydney: Allen and Unwin.

Logo Foundation. (2015). *Logo History*. Retrieved from http://el.media.mit.edu/logo-foundation/what_is_logo/history.html Merkel, R., & McNamara, R. (2015). *A bit of coding in school may be a dangerous thing for the IT industry*. Retrieved from

https://theconversation.com/a-bit-of-coding-in-school-may-be-a-dangerous-thing-for-the-it-industry-42259

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books, Inc.

Prensky, M. (2001). Digital natives, digital immigrants Part 1. *On the Horizon*, 9(5), 1, 3-6.

Schneider, E. (2015). Teaching kids how to code. Retrieved from https://elketeaches.wordpress.com/2015/12/15/teaching- kids-coding/

Sterling, L. (2015). *An education for the 21st century means teaching coding in schools.* Retrieved from https://theconversation.com/an-education-for-the-21st-century-means-teaching-coding-in-schools-42046

Stuckey, B. (2015). *Teaching coding in schools: absolutely necessary or another fad to waste teachers' time?* Retrieved from http://www.aare.edu.au/blog/?p=1076

Trucano, M. (2015). *Learning to code vs. coding to learn*. Retrieved from https://blogs.worldbank.org/edutech/learning- code-vs-coding-learn

White, D. S., & Le Cornu, A. (2011). Visitors and Residents: A new typology for online engagement. *First Monday*, 16(9). doi: 10.5210/fm.v16i9.3171

Williamson, R. C., Raghnaill, M. N., Douglas, K., & Sanchez, D. (2015). *Technology and Australia's Future: New technologies and their role in Australia's security, cultural, democratic, social and economic systems.* Melbourne: Australian Council of Learned Academies.

Wing, J. M. (2006). Computational Thinking. *Communications of the Association for Computing Machinery, 49*(3), 33-35. doi: 10.1145/1118178.1118215