Modern digital literacy: the computer science curriculum goes mainstream

Joy Gasson, Patricia Haden
College of Enterprise and Development
Otago Polytechnic

As computers and computing become increasingly ubiquitous, it is necessary to re-examine their role in education. We must broaden our traditional view of Computer Science and Information Technology education to include the skills required to function in a world where computers and computing are embedded in our academic disciplines and our daily activities. In doing so, we have the opportunity to inject computing effectively into all curricula, not just CS and IT, and to explore new pedagogical methods. The resulting educational landscape can be broad and inclusive, and help all learners to achieve their potential in the Digital Age.

Keywords: technology education, inclusive education, computational thinking

Introduction

As little as 30 years ago, computers were specialist devices maintained by highly trained technicians and used solely by professional computer programmers. Today, computers are mainstream: Infants play with iPads and grandparents make video Skype calls to their grandchildren. We shop online; we use spreadsheets to keep track of our finances. We have computers in our living rooms, and in our pockets. When 83% of homes in Australia have the internet (Australian Bureau of Statistics, 2014), when 82% of children in Finland have mobile phones (Kupiainen, Suoninen & Nikunen, 2011), the role of computers and computing in society has obviously changed. In response to this change, it is necessary to re-examine our approach to computing education.

In mainstream computer science education, recognition of the increasing ubiquity of computing in the lives of non-specialists has resulted in the rise of the Computational Thinking (CT) movement. CT was originally formulated by Wing (2006), who suggested that modern computers should be viewed as problem solving tools. She maintained that the cognitive approach dictated by the capacities of the computer led to a particular style of problem solving, which she termed computational thinking. The CT approach to problem solving leverages the things that computers are especially suited for -- processing of large quantities of data, problem decomposition (as complex logic is broken down into the simple operations computers are able to perform), abstraction (as modern programming languages are used to describe logical modules and components) and automation (allowing the computer to handler repetitive deterministic tasks). When electricity became widespread, it changed the way people approached the problem of turning milk into butter. So does the ubiquity of digital computation change the way people approach whatever problem they have to solve. Wing suggested that as problem-solving with computers has become the norm for an increasingly large number of disciplines and activities, computational thinking needs to be viewed as a core cognitive skill, not as the province of experts and specialists.

CT proponents have extended this argument to propose that computational thinking be considered an integral educational outcome, equivalent to literacy and numeracy (Cutts, Esper & Simon, 2011). They suggest that computational thinking can and should be taught to all students from the earliest years of school. There is currently a vigorous and well-funded programme in the United States to formally introduce CT into the K-12 (ages 5 to 18) curriculum (Snyder, Barnes, Garcia, Paul & Simon, 2012) - a far cry from the distant and abstruse "computer science" of only a few decades ago.

However, the fact that "everyone uses computers" does not imply that everyone uses (or will use) computers in exactly the same way. Analogously, mathematics has long been considered a core educational competency, introduced very early in school and, during those early years, taught in approximately the same way to all students. However, by late secondary school, students' maths education paths diverge, depending on aptitude, inclination and anticipated future need (e.g. see Bayfield High School, 2014). Some students will complete only the minimum required number of years of maths study, some will study maths at a basic, foundational level, and some will pursue advanced maths through high school and on into the undergraduate and graduate tertiary years. The same is true of computers and computing, and an effective programme of computing education must accommodate this.

A model for computing education

We propose a model of computing education based around four distinct groups: the Theoretician, the Practitioner, the Power User and the End User. (We discuss the four groups in detail below.) While computational thinking skills may be taught in a uniform way to all very young children, the presence of these different categories of computer user has an impact on the design of computing curricula in the later secondary and tertiary years.

The Theoretician

When simple, common objects from cars, to wrist watches to baby toys all contain computers, it is easy to lose sight of the fact that the scientific underpinning of computing is neither simple nor commonplace. The modern computer rests on a theoretical foundation that is largely mathematical and extremely complex. While the vast majority of modern computer users need have no understanding of automata theory, combinatorics, or the proof of P = NP, major technological advances still depend on progress in the formal mathematical systems of computation, and just as a proportion of students will wish to pursue a Ph.D. in theoretical mathematics or physics, some students will wish to pursue a Ph.D. in theoretical computer science. For these students, advanced mathematical training is essential. Historically, mathematics has been considered an essential part of *all* formal computing education. Recently, Thompson and his colleagues (Thompson, Bell, Andreae & Robins, 2013) performed a survey of New Zealand high school teachers in order to assess their readiness to teach computing and digital technologies. Along with questions about application use and programming experience, the survey contained items specifically exploring mastery of the mathematics on which theoretical computer science is based, because, Thompson argues, "[c]omputer science topics inevitably involve some mathematics" (pg. 248). While we acknowledge the inestimable value of advanced mathematics for those wishing to become Theoreticians, we argue that is not an appropriate part of the computing curriculum for all types of user.

The Practitioner

In the Practitioner category we place those users who perform computer software or hardware development or who maintain computer systems, as a profession. These are the computer programmers, the systems administrators and the software engineers. Such users must have a detailed and encyclopaedic knowledge of the machines and languages with which they work. They must have extensive experience with programming languages, integrated development environments, communication protocols and other elements of applied computer science. Their understanding of problem-solving via computational thinking will be very advanced. Nonetheless, they do not necessarily need to have great familiarity with the theoretical mathematics of computer science. Professional programmers need to know the correct algorithm to use to solve a particular problem, they do not need to be able to generate a proof of its performance complexity in the case of infinite processors and infinite memory. The Practitioner is the intended output of traditional Information Technology and Information Science tertiary degrees.

The Power User

The Power User is an expert in some discipline *other than computer science* who nonetheless must understand and use complex computing tools specific to his or her discipline. The proliferation of such tools has been rapid and invasive, and in many cases, has qualitatively changed the way that professional work in the discipline is undertaken. As described by the Center for Computational Thinking at Carnegie Mellon University (2014) "it is nearly impossible to do scholarly research in any scientific or engineering discipline without an ability to think computationally." We contend that it is this group who has been most profoundly affected by recent advances in computing technology and for whom facility with computational thinking is most essential.

The Power User can be found everywhere. In the physical and biological sciences, computer modelling software has enabled tremendous advances in understanding of complex system such as weather phenomena (e.g. Davis, Wang, Chen & Chen, 2008), wildlife habitats (e.g. Nielsen, McDermid, Stenhouse & Boyce, 2010) and biological systems (e.g. Pérez-Jiménez & Romero-Campero, 2006). In medicine and health care, artificial intelligence algorithms are used to search vast databases for patterns that can support diagnosis and treatment (e.g. Mahar, Imam, Tickle & Chen, 2013). In archaeology, 3D-modeling software is used to visualise lost civilisations (Allen, 2008). Even in the Arts, digital tools are used to produce new varieties of media, interactivity and expression (e.g. Barnett, 2009).

These software tools are designed, developed and maintained by computer professionals -- by The Practitioner.

Thus the Power User does not need to be a computer programmer or a network engineer, and does not need all of the technical training that the Practitioner requires. However, the Power User's tools are very complex. Learning to use them effectively can require extensive training. For example, Wilson and Tulu (2009) have recently argued for the introduction of an *entire tertiary qualification* in the field of Health Informatics, to cover the special requirements of medical knowledge and technical expertise necessary to work with the commercial software commonly used in modern medical institutions.

The End User

End Users are those who employ commercial software and hardware intended for non-specialist use. The child playing a computer game, the business person composing an email, and the student researching a topic on Google are all End Users. The Theoretician, the Practitioner and the Power User are almost certainly End Users as well, as they interact with computers in their non-professional capacities. The fact that we are all End Users is at the core of the Computational Thinking movement's philosophy, and End User skills are now considered as universally relevant as reading, writing and (basic) arithmetic. Lincoln University (in southern New Zealand) offers a series of undergraduate courses in End User Computing (McLennan, Churcher & Clemes, 1998). The focus in these courses is to teach students how to use common commercial software products such as spreadsheets, word processors and simple database engines to solve "real world problems" (Lincoln University, 2014). They report that even among their computer science graduates, the End User Computing courses are identified as among the most valuable in the curriculum. An interesting recent trend in software development are products designed for "End User Programming" (Cypher, Dontcheva, Lau & Nichols, 2010). These tools wrap the complexities of traditional computer programming in intuitive interfaces, allowing non-programmers to achieve high levels of control over software functionality. A common example is the WordPress blogging and web development system (WordPress, 2014), which allows general users to design and build interactive web sites with a simple point-and-click interface.

All four members of our user taxonomy -- the Theoretician, the Practitioner, the Power User and the End User -- interact with digital artefacts to perform necessary tasks, and these tasks are all valuable. The challenge for educators is to design curricula at all educational levels to facilitate the different knowledge and skills required by each group. In particular, we maintain that the Power User especially is underserved by traditional computing curricula, whose essential structures were established before computing became so deeply embedded in non-computing disciplines, before Power Users became as common as they now are. This shortcoming has an unfortunate consequence: it makes computing education inaccessible, or unattractive, to many students who are not suited to the Theoretician or Practitioner role, but whose interests and abilities would make them ideal Power Users. This, in turn, contributes to the current chronic shortage of qualified workers in many of the disciplines which now rely on the Power User (see, for example, CareersNZ, 2014). To address this problem, we need to contemplate radical modification of both what and how we teach. Adjusting to the changing role of computing in the world requires a redesign of curriculum and pedagogy at both tertiary and secondary levels.

Tertiary curriculum and pedagogy

The traditional undergraduate and graduate degrees in Computer Science and Information Technology are designed to serve the Theoretician and the Practitioner. (Individual tertiary programmes may be more heavily inclined to one group or the other.) For example, the Association for Computing Machinery (the primary scholarly and professional body for Computer Science) identifies the following Core Knowledge Areas in its most recent recommended tertiary curriculum (ACM Computer Science Curricula 2013; pg. 37):

Table 1: Core knowledge areas. ACM recommended curriculum 2013

Algorithms and Complexity	Networking and Communication
Architecture and Organization	Operating Systems
Computational Science	Platform based Development
Discrete Structures	Parallel and Distributed Computing
Graphics and Visualization	Programming Languages
Human Computer Interaction	Software Development Fundamentals
Information Assurance and Security	Software Engineering
Information Management	Systems Fundamentals
Intelligent Systems	Social Issues and Professional Practice

These comprise a broad and robust coverage of information and skills necessary for theoretical computer

scientists, software developers and systems engineers, i.e. specialist computing professionals. However, they would be both daunting and to some extent irrelevant for the lawyer, the doctor, the archaeologist or the artist who wishes to use the power of modern computer technology to solve important problems in his or her field.

In contrast, the University of Waikato (in northern New Zealand) offers a Bachelor of Science in Applied Computing, which it describes as "concentrat[ing] on using existing tools and software libraries to build systems in the database, internet, game and multimedia areas" (University of Waikato, 2014). The degree requirements allow for multiple courses each year taken *outside* of the Computer Science department. Thus a student following this programme can acquire essential specialist knowledge in law, or medicine, or art, along with advanced technical computing skills for mastery of complex software -- they can become a Power User.

In many institutions, it may be impractical to introduce an entire degree programme in Applied Computing or in a computing-intensive non-CS discipline such as Health Informatics. However, many tertiary institutions might easily support collaborative efforts between departments in response to the increasing embedding of computing in other academic areas. For example, a college with a medical school or nursing programme could take advantage of the technical skills of their CS or IT departments to offer classes devoted to training future health practitioners in the end-use of complex software.

Several recent instances of this type of collaborative approach can be found. Perkovic and his colleagues (Perkovic, Settle, Hwang, et al. 2011) describe the introduction of general computation and computational thinking into both the specialist and common curricula at DePaul University. The common curriculum is a set of papers taken by all students, regardless of their area of major study, with the goal of developing a broad knowledge base and strengthening general academic skills. The authors describe the recent addition to the common curriculum of a paper titled "Mathematical and Technology Literacy" which focuses on learning to solve problems with the support of end-user software. In addition, computational thinking has been explicitly incorporated into a total of 19 different papers at DePaul including not only CS and IT, but also Geology, History and Environmental Science.

Similarly, Hambrusch and her colleagues (Hambrusch, Hoffman, Korb, et al. 2009) from Purdue University describe a course aimed at developing computational skills in the sciences via a "problem-driven approach focusing on scientific discovery through computational methods" (pg. 183). This single paper is taken by all science majors and is taught collaboratively by staff from the computer science, physics, biology, chemistry and statistics faculties. In each area, problems that require a computational approach are identified, and students learn the computational techniques and tools required in the context of the associated specific scientific discipline.

Although the role of Power User may be easiest to envision in the sciences, many educators are exploring ways to leverage CT in the humanities and arts. Barr and Stephenson (2011) present a framework for introducing computational thinking into all academic disciplines. They identify a number of core CT competencies including data collection, data analysis, data representation and automation (see below for a more general discussion of the elements of curriculum development and design) which they maintain can be found in all subjects, not just hard sciences. In social studies they suggest the collection and analysis of population data; in the language arts, the linguistic analysis of sentences.

Even those careers traditionally considered "vocational" such as automotive repair and building construction, increasingly require their practitioners to be comfortable with complex computing tools. Students following these career paths don't need automata theory or computer programming classes, but to remain competitive in modern industry, they need computer competence that goes beyond that supported by traditional curricula.

Computing's increasing ubiquity may also dictate change in the content of existing courses in the CS and IT curricula. For example, advanced maths is currently considered a requirement for any student wishing to enter computer science or information technology, and is typically included in the first year of CS and IT degrees. In this context "advanced maths" often means exclusively calculus. However, for the eventual Power User (and for some Practitioners), calculus is neither necessary nor particularly useful. For example, Gasson (2005) describes a radical restructuring of the formal mathematical component of a Bachelor of Information Technology degree from traditional calculus to discrete numerical methods set specifically in a computing context. After teaching typical calculus for several years, Gasson noted that "Even the simpler concepts that were developed in the Calculus paper were not used in our degree programme or in the majority of IT positions in which our students are ultimately employed." To address this conflict, she restructured the paper from its existing content (e.g. teaching differentiation using examples about the trajectories of missiles; teaching integration using examples of

water flow through canals) to something both more accessible and more relevant to information technology students. The current maths paper in the degree covers binary logic, theory of functions, recursion, induction and other mathematical concepts which occur commonly in software and systems development. Further, each topic is specifically contextualised to computer science. For example encryption algorithms (the techniques used to protect computer passwords sent across the internet) rest heavily on primes and factorisation. Random number generation (essential to making computer games unpredictable and exciting) uses the linear congruential method. Students who might be bored by prime factors and linear congruence in isolation are invariably interested in passwords and computer games. Placing theoretical material in a concrete and relevant context provides both clarity and motivation. Since the restructuring of her degree's mathematics component, Gasson has seen both student interest and student performance improve.

Courses such as those described above aim to produce the Power User -- an area specialist with strong computing and computational thinking skills. As noted earlier, a current problem across many industries is a shortage of Power Users, and this shortage is expected to continue or worsen as more disciplines come to rely on computational approaches and solutions (Settle, Franke, Hansen et al., 2012). These courses have the potential to immediately address this shortage, as graduates would be fully prepared to enter their chosen field without the additional training that is currently often required (Wilson & Tulu, 2010). They might also alleviate one of the problems that plagues nearly all traditional Computer Science degree programmes -- high student attrition. Failure rates worldwide in first year computer science courses (especially programming) are alarming; reported failure rates often exceed 60% (Robins, 2010). Considerable research effort continues to be devoted to improving pedagogical approaches, but the reality remains that large numbers of students who begin a computing course are unable to finish it. We may choose to accept that such failure indicates that a student is not suited to becoming a Theoretician or a Practitioner. It does not, however, demonstrate that the student could not, given a different educational structure, become a very effective Power User. Modifying our existing curricula in response to the current computing landscape is an opportunity to help transform failure into success.

Secondary school curriculum and pedagogy

We have argued that by expanding our view of tertiary computer science education to embrace the Power User, we can produce a more inclusive educational program, suited to a potentially large number of students who are interested in problem solving with computers, but who are not interested in some aspects of formal computer science and information technology (e.g. mathematics or advanced programming). This opportunity is even more marked at the secondary school level. If we view computing through the lens of computational thinking, we can identify a number of potential classroom topics and activities that could appeal to students who would avoid traditional computer science classes. For example, recent developments in the construction of electronic components and microprocessors make it possible to take computers out of their traditional solid square boxes and attach them to fabric creations like clothing and soft toys. The set of techniques involved are collectively called "eTextiles". Using electrically conductive thread, one builds an electronic circuit simply by sewing. Components like touch sensors and coloured LEDs provide input and output for interactivity. A small microprocessor with a simplified instruction set acts as the "brain". Buechley and her colleagues (Buechley, Eisenberg, Catchen & Crockett, 2008; Buechley & Eisenberg, 2009) have shown that students as young as 11 can successfully work with these materials to create their own digital artefacts. Buechley observes that eTextiles classes are more likely to attract female students than are more traditional programming papers, an example of the opportunities for a more inclusive curriculum. It should be noted that although students may be drawn to this work by the opportunity to construct an electronic hand puppet or a flashing tote bag, they will, in the process, be learning electronics, programming and computational problem solving skills.

Another of the success stories of the new computer science is the introduction of robotics in primary and secondary schools. Recent developments in consumer robotics, especially Lego Mindstorms (Lego Group, 2014), have made it possible to introduce both construction and programming of simple robots into the school classroom. Students can construct mobile robots using the Lego block techniques that many will have been engaging in as play for many years. The MindStorm system includes a multiprocessor unit that can be programmed using Lego's proprietary "drag and drop" programming language or, with some software extensions, dialects of industrial programming languages such as Java and C++ (Pedersen, Nørbjerg & Scholz 2009). There is a growing corpus of publically available teaching materials specifically for MindStorm robotics in primary and secondary education (e.g. USC, 2014). There is an international contest, known as RoboCup (www.robocup2014.org), which allows school children to compete in events such as Robot Soccer, Robot Sumo Wrestling and Robot Theatre. RoboCup events are extremely popular and attract a broad range of students including those (e.g. girls) who are often underrepresented in traditional computer science education.

Through the construction and programming of simple robots, young students are learning the very core elements of computational thinking -- problem decomposition, abstraction, data processing -- and also some of the more formal elements of computer science such as programming language syntax and formal algorithmic construction. What is of particular interest, however, is ***the way*** that this learning takes place. It is our observation that much of current robotics education in schools does not happen in the regular classroom. Rather, it happens in after-school clubs and holiday programmes. Within this context, didactic instruction given by a lecturer is replaced with independent exploratory learning, driven by the student. After a recent successful RoboCup event, the coach of the Waitati (Southern New Zealand) primary school team explained that "The kids do everything. My contributions were filling out the entry form and driving the van." (Clark, 2014). Traditional science teaching requires lectures, textbooks, worksheets, examinations, etc. This approach has centuries of demonstrated success behind it, and has proven value. However, the inherently open-ended flexible nature of modern computing technology provides an opportunity to employ an equally open-ended and flexible teaching approach. Such techniques maximise the opportunity for constructivist learning and student engagement, both of which have been demonstrated to contribute substantially to educational success (for an in-depth discussion see, for example, Sjøberg, 2007).

However, it must be noted that simply turning students loose with digital tools is unlikely to guarantee effective learning. For example, an increasingly popular method for introducing computer programming into the classroom -- both in secondary schools and introductory tertiary computer science programmes -- is through Visual Programming systems (e.g. Scratch. MIT Media Lab, 2014). These systems allow users to build computer animations using pre-constructed program elements. Users can thus express their algorithmic intentions without having to manage the syntactic complexity that makes computer programming especially difficult for the novice. Meerbaum-Salant et al. (2011) have noted, however, that incorrect use of Visual Programming can cause more problems than it solves. Encouraging students to treat a Visual Programming system as a toy and "have a play" can lead to bad habits and a distorted impression of real computer programming, which can interfere with later transfer to an industrial programming language. As developments in digital technology provide new opportunities for computational thinking and computer science education, we must consider carefully both what to teach and *how* to teach it. We suggest that with digital technologies, a flexible, yet supported and scaffolded approach is needed (see below for further discussion).

Approaches to embedding computational thinking

In the past decade, the emergence of the Power User and the End User has encouraged educators across academic disciplines to modify their teaching practice to incorporate computational thinking principles. As is to be expected with such a recent educational development, a variety of educational frameworks and methodologies are currently being considered and tested, and there is currently no clear consensus on a single best approach to the embedding of CT. For example, Perkovic et al. (Perkovic, Settle, Hwang et al., 2011) identify a set of seven core principles (based on Denning's larger "Great Principles of Computing"; Denning, 2003): computation, communication, coordination, recollection, automation, evaluation, and design. Educators are advised to construct course materials, learning objectives and assessments around those principles. In contrast, Barr & Stephenson identify specific tasks such as data collection and simulation. They suggest that opportunities to perform these tasks can be found in a wide range of academic subjects. Hambrusch et al. (Hambrusch, Hoffman, Korb, et al. 2009) take yet another approach, concentrating more on identifying the specific tools needed for computational problem solving.

However, one of the most widely used frameworks rests on the identification of just three essential elements of computational thinking: Abstraction, Automation and Analysis. This approach has emerged from a large number of studies exploring the introduction of CT into intermediate and high school programmes in the United States (Lee, Martin, Denner et al., 2011). As defined by Lee et al., abstraction is the process of "generalising from specific instances"; identifying the essential elements of a problem and pruning away the rest. Automation is the work of the computer. Analysis is the act of considering whether one's abstractions are correct and extracting answers from the results of the computer's work. Lee et al. suggest that CT can be embedded into one's educational practice by explicitly incorporating these three principles.

As an example, Lee et al. describe a study module on epidemiology designed for intermediate school children. Students explore the potential spread of an infectious disease throughout their school based on environmental factors such as population size, virulence and physical environment. Students approach this problem by using a computer modelling and simulation tool. They are required to identify the specific features of their environment that would have an impact on spread of the disease (abstraction). They must interact with the modelling software to make it run multiple simulations of the movement and changing health status of the population

(automation). They then assess the accuracy of their approach by comparing their results to actual attendance records at their school during an earlier flu season (analysis).

It must be noted that the "automation" step in the framework of Lee et al. (2011) may, as in this example, involve the use of complex computer software. To accommodate a wide range of students, Lee and her colleagues recommend a structured approach which they call "Use-Modify-Create". With this technique, students are first taught to *Use* existing artefacts. For example, they might run pre-written programs or play computer games. As they gain experience, students are guided toward being able to *Modify* these existing artefacts. This might involve, for example, editing the parameters of an existing piece of code. Finally, students progress to being able to independently *Create* new artefacts. We have successfully employed the Use-Modify-Create approach in developing a robotics module for use in high school digital technology courses (Haden, McKenzie and Parsons, 2014). The module consists of a series of teaching sessions where students with minimal programming or electronics experience learn to build and control simple mobile robots. In the first session, students construct robots using a pre-programmed microprocessor brain so that, if they are able to correctly wire up the wheels and motor controllers, the robot will perform a little dance. In the second session, students are introduced to robot programming through the use of a pre-written library containing high-level commands such as *move(distance)* and *turn(degrees)*. In the final sessions, students are taught how to program the low-level commands necessary to gain complete control over the robot's movement and sensors. This scaffolded approach allows students increasing independence as they gain technical mastery.

Conclusion

From smart phones in our pockets to GPS systems in our cars, from medical databases to electronic pay systems at grocery stores, from Angry Birds to the weather predictions on the evening news, computers and computing surround us. As the role of computing grows, so do the number of opportunities for people to engage productively in study and careers that use and apply computing in valuable ways. But this increase in opportunity brings a concomitant need for adaptability in the way we teach computing. It even requires us to change what we see as "computing education". Computing education must embrace and include other disciplines; non-computer science disciplines must recognise that to neglect information technology is to risk becoming irrelevant. This may require a radical restructuring of curricula and pedagogy, which is a daunting prospect. But the return is not only in better student recruitment and retention, or in meeting the needs of the job market. It is also an opportunity to make a more inclusive educational environment, embracing students of all interests and abilities. In world where computing is everywhere, we can teach computing for everyone.

References

Allen, P.K (2008). Integrating new digital technology into the humanities: 3D modelling for archaeology, art history and urban planning, *Journal of Computing Sciences in Colleges,* 23(3), 59-60.

Association for Computing Machinery (2013). *Computer Science Curricula 2013*. Retrieved from www.acm.org/education/CS2013-final-report.pdf

Barnett, A. (2009). The dancing body as a screen: Synchronizing projected motion graphics onto the human form in contemporary dance. *Computers in Entertainment,* 7(1), 5:2 - 5:32.

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

Bayfield High School (2014). *Senior Course Selection Booklet.* Retrieved from http://www.bayfield-high.school.nz/files/1340149186_SeniorCourseSelectionBooklet.pdf

Buechley, L. & Eisenberg, M. (2009). Fabric PCBs, electronic sequins, and socket buttons: techniques for etextile craft. *Personal and Ubiquitous Computing*. 13(2). DOI 10.1007/s00779-007-0181-0

Buechley, L., Eisenberg, M., Catchen, J. & Crockett, A. (2008). The LilyPad Arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education, *CHI '08 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 423-432.

Careers NZ (2014). *Jobs in skill shortage*. Retrieved from http://www.careers.govt.nz/tools/skill-shortage-jobs/result/it-and-telecommunications.

Center for Computational Thinking at Carnegie Mellon University. (2014) Retrieved from http://www.cs.cmu.edu/~CompThink/

Clark, T. (2014) *pers. comm.* 27-6-14.

Cutts, Q., Esper, S. & Simon, B. (2011). Computing as the 4th "R": a general education approach to computing education, *Proceedings of the seventh international workshop on Computing education research*, August 08-09, 2011, Providence, Rhode Island, USA. https://doi.org/10.1145/2016911.2016938

Cypher, A., Dontcheva, M., Lau, T. & Nichols, J. (2010). *No Code Required: Giving Users Tools to Transform the Web.* Morgan Kaufmann Publishers Inc. San Francisco, CA

Davis, C., Wang, W., Chen, S.S. & Chen, Y. (2008). Prediction of Landfalling Hurricanes with the Advanced Hurricane WRF Model. *Monthly Weather Review*, 136, 1990–2005.

Denning, P. (2003). Great principles of computing. *Communications of the ACM*, 46(11), 15–20.

Gasson, J. (2005). Why Calculus isn't Relevant in Tertiary Information Technology Programmes *New Zealand Mathematics Colloquium Contributors' Abstracts*, Massey University, New Zealand 5 -7 December 2005.

Haden, P., MacKenzie, P. & Parsons, D. (2014). Junkbots: A low-cost alternative for robotics in schools. *Submitted for publication to WiPSCE 2014 The 9th Workshop in Primary and Secondary Computing Education*.

Hambrusch, S., Hoffmann C., Korb, J.T., Haugan, M. & Hosking, A.L. (2009). A multidisciplinary approach towards computational thinking for science majors, *Proceedings of the 40th ACM technical symposium on Computer science education*, March 04-07, 2009, Chattanooga, TN, USA.

Household Use of information Technology, Australia . Retrieved from http://www.abs.gov.au/ausstats/abs@.nsf/Lookup/8146.0Chapter12012-13

Kupiainen, R., Suoninen, A. & Nikunen, K. (2011). Online Habits of Finnish Children: Use, Risks and Data Misuse. *Nordicom-Information*, 33(4), 51-57. Retrieved from http://www.nordicom.gu.se/sites/default/files/kapitel-pdf/344_kupiainen_suoninen_nikunen.pdf

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011). Computational Thinking for Youth In Practice. *ACM Inroads*, 2(1), 32-39.

Lego Group (2014) Lego Mindstorms. Retrieved from http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com

Lincoln University. Retrieved from http://www.lincoln.ac.nz/Degrees-Diplomas-and-Certificates/Courses-A---Z/Course-Page/?CourseCode=COMP+307

McLennan, T. Churcher, C. & Clemes, S. (1998). *Should End User Computing be in the Computing Curriculum?* Centre for Computing & Biometrics, Lincoln Univ., Canterbury

Meerbaum-Salant,O., Armoni, M. & Mordechai Ben-Ari, M. (2011) Habits of programming in Scratch. *ITiCSE '11: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education.* 168-172.

MIT Media Lab (2014) Scratch. Retrieved from scratch.mit.edu

Nahar, J., Imam, T., Tickle, K.S. & Chen, Y. (2013). Computational intelligence for heart disease diagnosis: A medical knowledge driven approach. *Expert Systems with Applications*. 40(1), 96–104

Nielsen, S.E., McDermid, G., Stenhouse, G.B. & Boyce, M.S. (2010) *Biological Conservation* 143, 1623–1634.

Pedersen, R.U., Nørbjerg, J., & Scholz, M.P. (2009). Embedded programming education with Lego Mindstorms NXT using Java (lejos), Eclipse (xpairtise) and Python (pymite). *Proceedings of the 2009 Workshop on Embedded Systems Education*, 50-55. https://doi.org/10.1145/1719010.1719019

Perkovic, L., Settle, A., Hwang, S. and Jones, J. (2010). A framework for computational thinking across the curriculum. *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, June 26-30, 2010, Bilkent, Ankara, Turkey. https://doi.org/10.1145/1822090.1822126

Pérez-Jiménez, M. & Romero-Campero, F. (2006). P Systems, a New Computational Modelling Tool for Systems Biology, *Transactions on Computational Systems Biology,* 4220, 176-197

Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education* 20(1), 37-71. https://doi.org/10.1080/08993401003612167

RoboCup. (2014). Retrieved from http://www.robocup2014.org/

Settle, A., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C. & Wildeman, B. (2012). Infusing computational thinking into the middle- and high-school curriculum, *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, July 03-05, 2012, Haifa, Israel.

Sjøberg, S. (2007). Constructivism and learning. In Baker, E.; McGaw, B. & Peterson P (Eds) *International Encyclopaedia of Education* 3rd Edition, Oxford: Elsevier.

Snyder, L., Barnes, T., Garcia, D., Paul, J. & Simon, B (2012) Analysis. *ACM Inroads.* 3(2), 69-71. DOI: 10.1145/2189835.2189857. ISSN: 2153-2184.

Thompson, D., Bell,T., Andreae, P. & Robins, A. (2013). The Role of Teachers in Implementing Curriculum Changes. *SIGCSE'13*, March 6–9, 2013, Denver, Colorado, USA. 245-250.

University of Otago, Department of Computer Science (2014) Retrieved from http://www.cs.otago.ac.nz/cosc341/index.php

University of Southern California (2014) Retrieved from http://www-robotics.usc.edu/interaction/outreach/uno-stagnes/curricular.html

University of Waikato (2014) Bachelor of Applied Computing. Retrieved from http://uwt.waikato.ac.nz/studyhere/BSc_comp.shtml

Wilson, E. & Tulu, B. (2010). The Rise of a Health IT Academic Focus. *Communications of the ACM,* 53(5). https://doi.org/10.1145/1735223.1735259

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, 49, 33–35.
WordPress. (2014). Retrieved from https://wordpress.com/

Contact author: Patricia Haden, Patricia.Haden@op.ac.nz

Please cite as: Gasson, J., & Haden, P. (2014). Modern digital literacy: the computer science curriculum goes mainstream. In B. Hegarty, J. McDonald, & S.-K. Loke (Eds.), *Rhetoric and Reality: Critical perspectives on educational technology. Proceedings ascilite Dunedin 2014* (pp. 49-57).
https://doi.org/10.14742/apubs.2014.1062

Note: All published papers are refereed, having undergone a double-blind peer-review process.