A Review of Computer Science Resources to Support NCEA

Sumant Murugesh
Computer Science and Software Engineering
University of Canterbury
Christchurch
sumant.murugesh@canterbury.ac.nz

Tim Bell
Computer Science and Software Engineering
University of Canterbury
Christchurch
tim.bell@canterbury.ac.nz

Ann McGrath
College of Education
University of Canterbury
Christchurch
ann.mcgrath@canterbury.ac.nz

This quality assured paper appeared at the 1st annual conference of Computing and Information Technology Research and Education New Zealand (CITRENZ2010) incorporating the *23rd Annual Conference of the National Advisory Committee on Computing Qualifications*, Dunedin, New Zealand, July 6-9. Samuel Mann and Michael Verhaart (Eds).

Abstract
The Ministry of Education in New Zealand is in the process of making a major revision to the technology curriculum for NCEA. These changes include the addition of a body of knowledge with distinct knowledge and skills for five strands of Digital Technologies, one of which is ”Programming and Computer Science”. Because this contains material that will be new to schools, the changes present several implementation challenges, especially professional development for teachers in the subject area, the development of teaching materials, and also advice for students to choose the most appropriate career path. This paper will give a comprehensive review of the quantity and quality of existing resources that are available for teachers to use as course content and for their own professional development, identify strengths and weaknesses in the resources, and gaps that need to be filled. The gaps represent opportunities for tertiary organisations and industry to support schools as they make the transition to the new structure.

Keywords
Computer Science, Programming, Information Technology Education, Curriculum, Pedagogies, Teaching, Learning, Kinaesthetic Activities

Background

A secondary school Computer Science curriculum can have a significant influence on student career paths, both for laying the groundwork for further study, but more importantly, for exposing students to the essence of the discipline itself.

In recent years few New Zealand schools have taught Computer Science as a discipline – at best there have been modules on programming at some schools, and more often computing education has been focused on general-purpose applications and skills. To compound this problem, courses that teach "computing as a tool" have given students the impression that Computer Science might be an extension of these topics because of the lack of corresponding courses between secondary and tertiary education. The new technology curriculum (released in 2007) provided a generic framework for teaching technology, but computing teachers found it very difficult to map their subject onto the generic topics, and many schools resisted the integration with the technology curriculum, which resulted in a reliance on unit standards. Consequently courses offered fell outside the official NZ Curriculum, and developed a second-rate reputation (Bell, Andreae, & Lambert, 2010).

In 2008, two reports were released that highlighted the poor state of computing in schools (Carrell, Gough-Jones, & Fahy, 2008), (Grimsey & Phillipps, 2008). As a result the Ministry of Education called together a "Digital Technologies Experts Panel" (DTEP) representing industry, tertiary and high schools. By mid 2009 it had produced a body of knowledge, and recommendations that resulted in Digital Technologies becoming a distinct area within the Technology Curriculum. Alongside this development cognisance was taken of teachers' professional development needs as they grapple with more change. Amongst other initiatives, in September 2009 university Computer Science departments began developing a guide to resources to support implementation in time for 2011.

This paper will give a comprehensive review of the quantity and quality of existing resources that were identified in the process of collecting teaching resources, identify strengths and weaknesses, and highlight gaps so that that these can be filled.

Implementing CS in Secondary Schools

The Digital Technologies Expert Panel recommended that digital technologies be delivered as five "strands": Electronics, Programming & Computer Science, Digital Information, Digital Media, and Digital Infrastructure. Achievement standards are being written for these strands to assess specific knowledge and skills not address by the generic technology achievement standards; at the time of writing, the level 1 (for year 11 students) standards have been drafted, and levels 2 and 3 are in progress. As well as the implementation of these new achievement standards, teaching and learning guidelines are being created specifically for Digital Technologies to give clear guidance on how the standards could be taught, based on the body of knowledge associated with these strands. While these will provide specific guidance on issues like which features of a programming language might be taught, it doesn't provide the actual resources for teaching the topic. This is where a guide to resources, the focus of this paper, plays an important role. While teachers are free to use whatever teaching methods are appropriate for their environment, most will appreciate strong

guidance and ready-to-use resources, particularly given the large amount of new material that they will need to learn themselves, and then prepare to teach.

This paper focuses on a strand which introduces significant changes: Programming and Computer Science. This strand looks at computing as a discipline rather than computing as a tool, and will introduce a range of Computer Science topics that go well beyond computer programming, and their success will depend on having competent and confident teachers to deliver the courses. Relatively few teachers have a background in Computer Science, with many originally having taught typing, commerce or maths. For the changes to NCEA to be successful, NZ teachers need to be provided with a rich source of resources to teach the new topics, and they will require extensive professional development (PD) to be able to make use of them. Given the very short timeframe (less than one year to prepare for the new Year 11 courses), it will be particularly valuable if existing resources for teaching can be identified and rated so that it is easy for teachers to select the ones that will enable to them to implement courses with the minimum of effort. Fortunately, as will become evident from the survey later in this paper, there is indeed a wealth of suitable resources.

Methodology

Towards the end of 2009 relevant online and offline resources were searched to support the new topics in the Digital Technologies body of knowledge. The methodology used was to exhaustively scan a wide range of computing teaching resources, and document any that appeared suitable to support the new topics. This included:

- Direct contact with individuals involved in similar initiatives in the UK, US and Australia
- Going through repositories such as the CSTA Repository (Computer Science Teachers Association, 2006) and CITIDEL (Computing and Information Technology Interactive Digital Educational Library) (Villanova University, Virginia Tech University, 2007)
- Going through collections of Computer Science material written for schools from organisations such as CS Unplugged, MaTHmaniaCS, CS for fun, PLTL, KLA QwikWiki, TeachICT and CS Inside.
- Monitoring extensive discussions on teacher support mailing lists, where teachers recommend resources to each other.
- Performing searches on YouTube using terms like "Computer Science", "Programming" and "Education", and looking for videos that are related to good ones that have already been identified.
- Looking at textbooks in Programming and Computer Science written for beginners' use.
- Examining online encyclopaedias and dictionaries, especially Wikipedia, and following external links from these.
- Looking for resources already developed for teaching towards pre-university entrance examinations overseas such as AP (Advanced Placement) in the USA and AQA (Assessment and Qualifications Alliance) in the UK.
- Exploring resources published to encourage female students to learn CS.
- Looking through CS departmental websites of universities for teaching materials and graduate student projects that were shared online.
- Performing internet searches for pages that would be relevant.

This resulted in a vast collection of videos, visualisations, lesson plans, textbooks, podcasts, wikis, programming tutorials and various other kinds of teaching resources (Murugesh, 2010), a draft version of which is available online[1]. Almost all of the resources are either available free of charge, or at a relatively low cost. These resources were then compared with standard curricula, including the Association of Computing Machinery (ACM) K-12 Model curriculum (ACM, 2003) and the new Digital Technologies Achievement Standards and Body of Knowledge (MOE, DTEP, 2009). Setting aside material aimed at tertiary students, we identified around 30 major sources for high quality resources that can be readily used by secondary school teachers.

Results

In this section we will highlight some of the material found in the survey, with the aim of giving examples of the kinds of resources that are ready for NCEA use, those that will need to be simplified, and finally, areas that do not have any resources. The weakly resourced areas will need to be addressed, and contributions to complete the shared pool of resources could be made by tertiary institutions, experienced teachers, or writers contracted specifically for this purpose. We begin by examining some of the resources that are at a suitable level and cover a broad range of topics.

General Resources

Teachers are able find concise explanations of Computer Science terminology using online dictionaries and encyclopaedias. Wikipedia (wikipedia.org) is a major source, and despite its lack of a formal quality assurance process, for computing terms it can provide a good starting point for definitions, and external links to useful material. Other useful dictionaries identified were FOLDOC (foldoc.org), Dictionary of Algorithms and Data Structures (nist.gov/dads/), Computer Science Research Library (www.cs.mu.oz.au/~csyeo/csrl/), and TECHtionary (techtionary.com). Another general resource is textbooks in Computer Science that cover a broad range of topics. Of particular relevance are those that have been developed in the UK for training towards qualifications such as the AQA.

The basic concepts that appear in the NZ body of knowledge are often taught at first year undergraduate level at universities, so using the first few lectures from such courses can be relevant for school use. A wide array of lecture notes, video lectures and podcasts are available from various universities. However in certain cases, some modification and tailoring will be required to make these appropriate for school use.

Another general set of resources that is freely available are those that provide kinaesthetic activities aimed at school-aged children. The CS Unplugged (csunplugged.org) and CS for Fun (cs4fn.org) websites provide extensive free material aimed at teaching CS concepts to school students; related resources are also at MATHmaniaCS (mathmaniacs.org) and The Computing Science Inside project (csi.dcs.gla.ac.uk). Offline activities such as these are sufficient to cover the requirements of the standard, although they can also be integrated with programming exercises if desired. For instance, the Greenroom (Kölling, 2010) and Scratch Unplugged (Ben-Ari, 2010) have programming activities that are based on CS Unplugged kinaesthetic activities. In addition, teaching

[1] http://www.cosc.canterbury.ac.nz/tim.bell/dt/

programming concepts using robotics in schools is becoming more and more popular. Entire curriculums are being developed to get girls involved in robotics (IRCS, 2001). However, robotics kits can be expensive for schools to purchase.

In order to properly map the specific resources to NCEA level, each of the three main learning objectives within the Programming and Computer Science strand will be discussed separately. At level 1 (usually year 11) students would be working with a relatively introductory version of these topics, but by level 3 (year 13) they would be looking at broader and deeper coverage.

Learning Objective 1: "Demonstrate an understanding of concepts across Computer Science and Software Engineering"

At level 1 this is covered by a simple understanding of the concept of an algorithm, the role of programming languages (including compilation and translation), and a simple introduction to interface evaluation[2].

Visualisation of algorithms is a popular medium for teaching the concept of algorithms. The better visualisations let users change parameters like problem size, magnification, and the initial condition of data sets (Martin, 2007). A problem with many visualisations is that they are too confusing for students.

Another way to teach algorithm is using role-play activities, or data structures built out of students. The kinaesthetic activities mentioned earlier demonstrate many algorithms effectively. For example, csunplugged.org has a "battleships" searching activity that contrasts linear search, binary search and hashing, and with follow-up activities it is easy for students to appreciate the massive difference in performance between algorithms. An activity from the KLA collection that represents pointers by having students place their hands on another student's shoulder provides a powerful "inside" view of how data structures work.

The concept of programming languages can be demonstrated by getting experience with compilers and interpreters, to appreciate what their roles are. Many free systems are available to support language compilation and interpretation, although students will probably need to be supplied with complete sample programs since these may be in languages that they haven't yet learned to program in.

Simple interface evaluation is covered in one of the CS Unplugged activities (based on non-computing devices such as doors and stoves), although a new exercise involving gadgets such as microwave ovens and mobile phones might be more relevant for helping students to become critical of real interfaces. Constructing such an exercise wouldn't be difficult, but this is one important gap identified by our evaluation.

At level 2 the body of knowledge concepts include complexity, tractability and computability; compression, error detection and correction, and encryption; the idea of formal grammars; and the software development lifecycle. Several resources are available that cover intractable problems, especially in graph theory, such as map colouring exercises, online

[2] Interface evaluation is actually published with the second objective, but in the Achievement Standards it has been grouped with the CS Concepts.

games and kinaesthetic activities. An area that is lacking materials is the area of formal grammars, as the resources for this topic tend to be aimed at CS graduates and go into greater depths too quickly. However, there are resources on automata theory aimed at young students, which can be used to convey the idea that programming languages are specified precisely (although technically this is only the lexical part of the language). Also at this level, there is a reasonable coverage of materials in software development methodologies in terms of tutorials, case studies, and exercises for giving a basic understanding of the process and lifecycle.

HCI evaluation at level 2 requires a slightly more formal approach. There is a limited amount of resources, such as articles on heuristic evaluations and theoretical material, so this area will benefit from activities and tasks to illustrate the use of usability heuristics in the evaluation of interfaces that is aimed at high school students and the contexts they live in.

At level 3 the body of knowledge advocates students having a broad view of a range of topics in CS and Software Engineering, with a view to making them aware of the issues that exist in various topics, even though they won't know many of the actual techniques and tools in those topics. Teachers may choose the areas they are comfortable with. We have identified well resourced areas as Computer Security, Databases, Visual Computing, HCI, Intelligent Systems, Networks, Ethics, Information Retrieval, Data Communication and Web Computing. Those areas that have little suitable coverage are Operating Systems, Distributed Computing and Information Management. We have identified the weakest areas as Formal Methods, Discrete Structures and Computer Architecture, meaning that we could only find a few resources suitable for creating an understanding of these concepts for senior high school students.

Learning Objective 2: "Be able to understand, select and design data types, data structures, algorithms, and program structures for a program to meet specified requirements"

In general, this objective is focussed on the design of programs, rather than writing and debugging them. Much of this is best learned from experience, and apart from specific exercises, the best value from on-line resources is to help train the teacher in this area so that they can guide students through it. The objective also includes some material on data structures and data representation.

Level 1 of this objective is focused on the *design* of simple program structures, including the understanding of data types and control structures. There are lectures in generic data structures that are free for educational use, as well as some that are specific to programming languages like Python and Java. Various privately made YouTube videos offer easy to follow tutorials on implementing simple to complex data structures such as lists and arrays, trees and also functions and classes in Python and Java. Introducing programming logic to students can be done in some interesting ways using kinaesthetic activities such as the Programming Languages task from CS Unplugged. There are also games and activities that teach computational and logical thinking such as the ones from CS for Fun, and several Kinaesthetic Learning Activities (KLA's) to teach concepts in functional programming, control flow, data structures and functions, amongst other things. Also, a

range of tutorials and applets exists that clearly explains various searching and sorting algorithms and the use of various data structures.

Level 2 includes advanced representations of data (such as arrays, lists or user-defined types), and using programs with methods. There are videos, role play activities, and online tutorials that teach the concepts of arrays, lists, trees and other data structures, although some are linked to a particular programming language. Exercises that use methods and functions are usually available as extensions of lesson plans.

At level 3 the scope of the objective is to understand the properties and limitations of data types, programming using arrays or lists, methods with parameters and return values, and perform file handling. There are many resources on number systems (binary, hexadecimal, floating point numbers etc.), including kinaesthetic activities, textbook chapters, online number convertors, and videos. Programming exercises that use searching and sorting techniques incorporating arrays and other data structures are also available.

Learning Objective 3: "Be able to read, understand, write, and debug software programs using an appropriate programming language, tools, and software development process"
This objective is focussed on learning to write and debug programs. One of the first issues is working out which language should be taught. The standards are not prescriptive about the language, but just the features it must have. Teachers will need guidance on choosing a suitable language. In many cases it may be driven by local expertise or by the expectations of a tertiary institution so that students can transition smoothly to post-secondary study.

At all levels of the programming and software development objective, the Internet is a rich source for resources. Programming languages and IDE's commonly used at a beginner level are Scratch and Alice. They both use a drag-and-drop approach that encourages exploration by students as they learn programming principles, and both have extensive teaching resources available from their websites (scratch.mit.edu and alice.org). Greenfoot is an intermediate programming environment that introduces Java programming in a very graphical environment, avoiding students having to learn the finer points of Java syntax right at the start. Extensive exercises suitable for secondary school students are available from the teacher-only "Greenroom" on the Greenfoot website (Kölling, 2010).

When students graduate from the drag-and-drop environments to more general-purpose programming languages, the current popular choices in New Zealand include Python, Visual Basic, Java, C, C++ and C#. These popular languages are supported with a growing number of free on-line resources and teaching materials that are very helpful at this level.

Video tutorials are a great way for learning and revising topics in programming. These can supplement the teachers' professional development activities. Universities often offer free video lectures on YouTube, which has a growing number of tutorials.

Another great way to teach programming is using PLTL (Peer Led Team Learning), where a group of students

work together on an exercise cooperatively in small groups led by a peer leader, who is usually someone who knows the field and has done the exercise before (Huss-Lederman, 2007).

An area that really lacks resources here is the software development *process*. There is considerable material about how to do software engineering, but at the school level it is more appropriate for students to understand the motivation for software engineering rather than learn particular methodologies in detail. For example, students could investigate a software disaster (such as the Heathrow T5 or Denver airport baggage handling systems, or the INCIS system in New Zealand), and investigate what went wrong in the process. There is material on the internet relating to these events, but a more focussed lesson plan or guide would be helpful. Also useful would be tutorials that explain and compare the different development methodologies and exercises in test-driven development aimed at school level.

Conclusion

The survey has shown that there is a rich range of resources available that could be put to good use in New Zealand secondary schools. There were clearly a few areas where resources were limited and further development work would be valuable, and where some resources are too complex for use in the school environment. A narrated list of existing resources, at least for the level 1 objective, is being prepared in mid 2010 for teacher feedback and comments. The feedback will enable us to highlight resources that have proved to be useful, and identify the areas that teachers need more help with. New Zealand tertiary institutions that are involved in supporting school teachers will then be able to develop and publish appropriate resources to eliminate those gaps. Computer Science departments throughout NZ are collaborating to provide support for teachers as they work out how to deliver the new material. In addition, one initiative at the University of Canterbury to help with this is a new course in Computer Science Education where graduate students will do projects to produce resources especially aimed at the weaker areas identified by this survey. These projects will then be evaluated by school teachers, and their usefulness documented and eventually shared in the resource pool.

References and Citations

ACM. (2003). A Model Curriculum for K–12 Computer Science, Final Report of the ACM K–12 Task Force Curriculum Committee. Retrieved from http://www.acm.org/education/education/curric_vols/k12final1022.pdf

Bell, T., Andreae, P., & Lambert, L. (2010). *Computer Science in New Zealand High Schools.* Brisbane, Australia: Twelfth Australasian Computing Education Conference (ACE2010).

Ben-Ari, M. (2010). Scratch-unplugged - Project Hosting on Google Code. Retrieved from http://code.google.com/p/scratch-unplugged/

Carrell, T., Gough-Jones, V., & Fahy, K. (2008). *The future of Computer Science and Digital Technologies in New Zealand secondary schools: Issues of 21st teaching and learning, senior courses and suitable assessments.* Retrieved from http://dtg.tki.org.nz/content/download/670/3222/file/Digital%20Technologies%20discussion%20paper.pdf

Computer Science Teachers Association. (2006). Retrieved from A Web Repository of K-12 Computer Science Teaching and Learning Materials: http://csta.villanova.edu/

Grimsey, G., & Phillipps, M. (2008). *Evaluation of Technology Achievement Standards for use in New Zealand Secondary School Computing Education: A critical report.* From New Zealand Computer Society: http://www.nzcs.org.nz/news/uploads/PDFs/200805NCEAReport.pdf

Huss-Lederman, S. (2007). Retrieved from Peer Led Team Learning in Computer Science: http://www.cs.duke.edu/csed/pltl/index.php

IRCS. (2001). *Agents for Change*. (I. f. Science, Producer) Retrieved from Agents for Change: Robotics for Girls, A Robotics Curriculum for the Middle School Years: http://www.ircs.upenn.edu/pennlincs/index.html

Kölling, M. (2010). Retrieved from The Greenroom : http://greenroom.greenfoot.org/door

Martin, D. R. (2007). Retrieved from Sorting Algorithm Animations: http://www.sorting-algorithms.com/

MOE, DTEP. (2009, December). *Technological Context Knowledge and Skills documents.* Retrieved from http://www.techlink.org.nz/curriculum-support/tks/resources/Technological-Context-Knowledge-and-Skills-12-2009.pdf

Murugesh, S. (2010). Delivering Computer Science Concepts at Secondary School Level. *The 8th New Zealand Computer Science Research Student Conference (NZCSRSC)*, (pp. 1-7). Wellington, New Zealand.

Villanova University, Virginia Tech University. (2007). Retrieved from Computing and Information Technology Interactive Digital Educational Library: http://www.citidel.org/