

The introduction of Computer Science to NZ High Schools — an analysis of student work

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

Peter Andreae
School of Engineering and
Computer Science
Victoria University of
Wellington
Wellington, New Zealand
peter.andreae@ecs.vuw.ac.nz

Heidi Newton
School of Engineering and
Computer Science
Victoria University of
Wellington
Wellington, New Zealand
heidi.newton@ecs.vuw.ac.nz

Anthony Robins
Computer Science
Department
University of Otago
Dunedin, New Zealand
anthony@cs.otago.ac.nz

ABSTRACT

In 2011 New Zealand introduced computer science as a topic that students could take as part of their studies in the last three years of high school. The change was initiated in late 2008, so the new material was introduced with barely two years of preparation and minimal teacher training. Despite this tight timeline, many schools adopted the new topic, and many students successfully completed assessment in it in 2011. The format of the assessment was required to be a report. In this paper we look carefully at the work that students submitted by examining publicly available information (statistics, markers' comments and exemplars), and performing a detailed analysis of a sample of 151 student papers. We describe the nature of the assessment (which is report-based with very flexible criteria for how students can demonstrate their understanding), and examine the kind of work that students submitted to meet the criteria, drawing out good practices that enabled students to do well. A recurring theme is the importance of students being able to use personal authentic examples so that the examiner can hear the "student's voice" in their report work, which provides evidence that the student has *understood* the topic rather than paraphrased descriptions. The analysis also reveals the value of prompting students effectively to get them engaged properly with the concepts, and identifies successful ways to achieve this in the three areas of the analysed standard (algorithms, programming languages and usability).

Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education]:
Computer Science education

General Terms

Measurement, Experimentation

Keywords

Computer Science education, report-based assessment, high school, algorithms, programming languages, usability

1. INTRODUCTION

Over the period 2011 to 2013, new computer science standards are being introduced to New Zealand (NZ) schools that will give students the opportunity to explore topics such as algorithms, human-computer interaction, cryptography, AI, graphics, and other areas of computer science that previously were normally encountered for the first time in university courses. This change addresses a problem that is common to many countries, where computing in schools had become focussed on learning to use the computer as a tool rather than supporting students interested in developing new software and systems. For example, a January 2012 report from the UK Royal Society points out that "many pupils are not inspired by what they are taught and gain nothing beyond basic digital literacy skills such as how to use a word-processor or a database" [8]. In the US, similar complaints were being made; for example, Margolis and Goode describe having to address the issue that "computer science exists on the margins of many public school core requirements" [10], and a 2009 report on the state of computing in US high schools (surveyed in 2005 and 2007) points out that "many schools and countries, policy-makers and administrators are failing to provide students with access to the key academic discipline of computer science", and concludes that "the results of these studies appear to indicate a continuing and troubling decline in student interest in computer science courses in high schools" [9].

These echo a 2008 report from the New Zealand Computer Society that noted: “We suspect that there is a huge number of potential computing professionals who have already opted out of the discipline during secondary school, either because of the lack of relevant achievement standards, or because of the unpalatable offering of what they are told *is* relevant for a future computing career” [11]. A report from a group of NZ teachers also raised serious concerns: “Computing is perceived to be second rate, is uncoordinated, under resourced, unsupported, lacking in a professional body, and in dire straits” [7]. Computer science has long existed as a high school subject in Israel and South Korea [8], but the above quotes illustrate how in the early 21st century serious computing courses were essentially on the decline in the US, UK and New Zealand. These concerns triggered a rapid sequence of events beginning in 2008 that resulted in the introduction of computer science as a subject in NZ schools in 2011 [3, 4].

Part of the purpose of the new topics being introduced to schools is to provide students with some grounding in CS concepts, but the primary goal is giving them exposure to the topic, providing the opportunity for them to find out if it is something they might be passionate about, without the confusion of it being presented as low-level digital literacy skills. There are several related initiatives internationally that have been started in recent years to provide formal computer science courses in high schools, including: (a) the “Georgia computes!” program “to increase interest in computing at the pre-teen level, improve quality of computing education at the high school level, draw students into the undergraduate level, and make apparent to students the opportunities for graduate study in computing” [6] (announced in 2006); (b) the “Exploring Computer Science” program in the Los Angeles Unified School District, which aims to “make computer science knowledge more available to and engaging for a broader segment of our student population” [10] (started in 2008); (c) the US “Computer Science: Principles” pilots to introduce CS as an Advanced Placement course [2] (started in 2010); (d) the new German compulsory standards introduced in some regions from 2008 [5]; and (e) the 2012 report by the Royal Society in the UK advocating sweeping changes to the ICT curriculum [8] that has resulted in planned changes. The more general CSTA K-12 standards reflect a strong interest in changing how computing is taught in schools¹.

In the NZ high school context, computer science has been introduced through three new “achievement standards” that are typically taken in the last three years of high school. The focus of this paper is the first standard, which was first taught in January 2011 (the start of the school year in NZ), and is described in Section 2. The paper looks in detail at the actual work done by students under the new standard. A sample of 151 student submissions were analysed in detail to identify both good practice and issues such as student misconceptions. From this we can learn better ways to deliver computer science topics to students in this age group.

An important consideration for interpreting this analysis is the context and the fast timeline with which the new ma-

terial was introduced. The body of knowledge defining the area was published in August 2009, just 16 months before the standard was first taught. Between this point and the beginning of 2011 the standard had to be written, teachers prepared, and courses designed. This meant that the preparation wasn’t as thorough as it could be, but it was seen to be better to have a few schools adopt the new standards quickly (they are not compulsory) than to delay for a year. Effectively the first year was a pilot, although the scale of uptake was remarkably high: in 2011 students from 49 schools registered for the new computer science standard [4]. Adjusted by population, this would be equivalent to 3,500 high schools in the US, so the level of uptake would be a good start on the 10,000 high school target of the CS10K project, and was achieved with just two years lead time.

The formal training provided to teachers was minimal; some had access to sessions at teacher events, and one-day workshops were run around the country, but there was no formal systematic training program and engagement was largely driven by individual teachers’ motivation. A key role was played by the newly-formed New Zealand Association of Computing and Digital Technology Teachers (NZACDITT) in sharing information and initiating events, and a network of contacts was set up in the country’s seven universities that teach computer science. A lot of the leadership (including leaders of the NZACDITT) were in Christchurch, and these people were impacted by the September 2010 and February 2011 Canterbury earthquakes, which occurred just at the point that the standards were about to be taught, and had a further impact on providing support for teachers making the changes! An important component of teacher support in the New Zealand system is exemplars of student work that show what might be expected, and the computer science exemplar didn’t become available until June 2011, well into the year in which the new standard was being offered.

Despite the constraints on preparation time from introducing the new standards so quickly, and adoption of the standards being optional, the number of students attempting the computer science standard in its first year of introduction was still substantial (1,429 students in 49 schools registered for the new CS standard [4], and 654 were submitted by the deadline). To give a sense of scale, there are currently (in 2012) about 485 schools teaching at high school level in NZ, although many of the schools are specialist or smaller rural schools. In NZ there are about 62,500 students at “Year 11” (the third to last year of high school, typically around 15–16 years old), which is the year that most students would attempt the new computer science standard. The first year was essentially a pilot, but nevertheless had sufficient uptake that it represents widespread national engagement.

In this paper we analyse the student submissions in detail based on three sources: officially released statistics and commentary about the grading, the published exemplars of student work, and a detailed analysis of a sample of 151 student submissions. Section 2 explains the new computer science standard and how it is assessed; Section 3 reviews the publicly available information about the submissions, Section 4 is a detailed analysis of the sample of student submissions, and Section 5 reports some lessons learned based on the observations.

¹<http://csta.acm.org/Curriculum/sub/K12Standards.html>

2. THE AS91074 STANDARD

During the final three years of high school in NZ students are focused on the “National Certificate of Educational Achievement” (NCEA), which involves completing a number of “standards”. Details of how computing appears in these standards are given in previous work [3, 4], and in this paper we focus on one particular standard which was introduced to schools in 2011, called “AS91074: Demonstrate understanding of basic concepts from computer science”². The standard is set and regulated by the New Zealand Qualifications Authority (NZQA).

The AS91074 standard is worth 3 credits, which translates into approximately 30 hours of student work, so it can be used as a relatively small component of a larger course, typically combined with other standards on programming, web design, digital media, information systems, electronics and so on. It is a “level 1” standard, which means it is generally used in the third to last year of high school; levels 2 and 3 apply to the final two years respectively.

The standard covers three main topics from computer science, which can be summarised as:

algorithms: the difference between algorithms, programs, and informal instructions; the kinds of steps that are used to make an algorithm; and comparing the “cost” of different algorithms for the same problem

programming languages: the kinds of programming languages that exist (including high and low level languages) and the roles of compilers and interpreters

usability: evaluating and comparing user interfaces in terms of usability

These topics provide a broad exposure to ideas from computer science, ranging from the mathematical analysis of algorithms to the human-factors aspects of usability evaluation. They are broad rather than deep — students don’t have to implement algorithms, but they can run programs that are provided; they don’t write compilers, but they need to document how they use them; and they don’t design interfaces, but instead learn to look at existing interfaces critically.

AS91074 is the first in a series of computer science standards (the level 2 and 3 standards are being phased in during 2012 and 2013) that together cover a wide range of topics from the ACM curricula. The general area of “Programming and Computer Science” includes two other standards at level 1, AS91075 (which is essentially about designing computer programs) and AS91076 (implementing programs), with further standards at levels 2 and 3. These are not analysed in detail in this paper, but come up occasionally because most students doing the CS standard are also attempting the programming ones.

²It is also known as “Digital Technologies 1.44”, which is the number it had when in draft. AS91074 is available publicly from <http://www.nzqa.govt.nz/ncea/assessment/search.do?query=Digital+Technologies&view=all>.

AS91074 is an “external” standard, which means that it is assessed at the national level; the majority of standards (including the programming ones) are “internal” and assessed at the school but moderated (i.e. audited) nationally. For various reasons, AS91074 is assessed using a student report, which means that the student submits a report on what is essentially project work. The report is limited to 14 pages.

The “AS” in the title stands for “achievement standard”, which means that it can be awarded with four levels of achievement: Not achieved, Achieved, Merit and Excellence, abbreviated as N/A/M/E respectively. Thus unlike many other grading systems, “A” means a minimal pass, and “E” is the best result; “N” means that the student gets no credit.

The standard itself is very brief — just over 2 pages, almost half of which is boilerplate information such as review dates and standard text. The main content of the standard is the criteria for the A, M, and E levels of achievement, which are given in just a few “bullet points” — 4, 5, and 4 points for A, M, and E respectively. The bullet points for M and E generally describe the higher standard of work required for those levels rather than additional content. For example, one of the Achieved bullet points is:

- describing an algorithm for a task, showing understanding of the kinds of steps that can be in an algorithm, and determining the cost of an algorithm for a problem of a particular size.

The equivalent point at Merit is:

- showing understanding of the way steps in an algorithm for a task can be combined in sequential, conditional, and iterative structures and determining the cost of an iterative algorithm for a problem of size n .

And the one at Excellence is:

- determining and comparing the costs of two different iterative algorithms for the same problem of size n .

Each of the three criteria require that the student shows that they can determine the “cost” of an algorithm (typically the running time or number of steps/comparisons); if they simply report the time from a single experiment then they have reached the Achieved level; if they compare different values of n (i.e. show the shape of the cost function) that would reach the Merit level; and if they *compare* two algorithms (typically a graph with two curves) then that would be Excellence. Of course, the markers will be looking for more than just numbers: the student needs to present the data well and discuss the trends that it shows.

The brevity of the standard gives great flexibility to teachers and students. In the algorithms example above, they can explore whatever algorithms they choose as long as the chosen algorithms can be used to demonstrate that the student has met the criteria. Of course, this flexibility also puts a lot of responsibility on teachers and students to choose good

examples, and in practice teachers share ideas and subject experts publish advice to guide these choices.

Marking of the student submission is done “holistically”, which means that if a student does very well in most areas but has problems in just one point, they won’t get a low grade just because they didn’t meet one basic criterion. For example, in general a student will have met all 4 of the Achieved bullet point criteria to be given the Achieved grade. It may be that their work was weak in one of the criteria, but if they had done particularly well in another (perhaps at Merit or Excellence level) then they would be given achieved overall.

The use of a report for assessment is challenging. To “demonstrate understanding of basic concepts from computer science”, a student report cannot just paraphrase definitions or on-line articles about the topic. For example, a report that describes selection sort and states that its cost is $O(n^2)$ does not constitute a demonstration of understanding, since this could be simply parroting text from a teacher or a paraphrase of one of many descriptions available online. Similarly, just listing the properties of a good user interface could be paraphrased from a textbook, with no understanding. Instead, students need to report on a personalised activity or investigations in order to demonstrate that they have acquired their own understanding of the concepts.

There are many ways to personalise the reports to meet this expectation. For example, timing an implementation of selection sort on their own computer using their own input data reflects a personal experience with the issue of quadratic behaviour as n increases. Given the number of possible permutations of even a small sequence of values, it is easy for students to example traces data that will be unique to themselves. For the programming language topic, a demonstration of a program written in some language, and compiling and running it can be done use the local environment in the student’s own account, using a program they have written themselves (perhaps for the programming standard, or even a simple modification of a “Hello World” program that uses their name). For the HCI topic, a student can apply the list of principles to the interface of a device that they personally own, such as an alarm clock or an mp3 player.

3. STUDENT SUBMISSIONS

There is considerable public information available about the student work for the AS91074 standard, which we have collated and analysed in this section. The three areas we look at are the student grade statistics from 2011, the published exemplars, and the markers’ comments on the work.

3.1 Student grade statistics

The New Zealand Qualifications Authority publishes statistics on student performance in each Achievement Standard³, and in Table 1 we show the number of candidates for the three Programming and Computer Science standards, along with the success rates of students (Not achieved, Achieved, Merit and Excellence).

³<http://www.nzqa.govt.nz/studying-in-new-zealand/secondary-school-and-ncea/secondary-school-statistics/>

In 2011, the first year that the AS91074 achievement standard was available, 654 student reports were submitted. The programming standards were significantly more popular, with nearly five times as many students taking those standards. This isn’t surprising because the topic is better understood, and is one that many teachers are familiar with and would have been more confident to offer it.

To put the participation rates in perspective, there were about 62,500 Year 11 students in school in NZ in 2011⁴, so participation in the new standards was relatively high considering that at least 75% of schools didn’t offer them [4].

The student grade distribution for AS91074 is typical of achievement standards, with relatively few students reaching the Excellence level, but the majority getting credit for the standard (Achieved or better). The programming standards have a higher proportions of Excellence grades — in fact, the results show similar numbers of students getting Excellence and Merit, and reflects the bimodal distribution of performance that has commonly been observed for programming courses [13]. We also note that there is a difference in grades between male and female students; fewer females attained the Excellence grades, with the difference being particularly noticeable for the programming standards.

It would appear that offering these standards early in high school has attracted a higher proportion of female students than advanced courses typically do. The proportion of female students taking the computer science standard (AS91074) was 27%, and the related programming standards (AS91075 and AS91076) had 30% and 36% respectively. While low, this fraction, which is for students taking the standards in their third to last year before university, is significantly higher than the proportions who enrol in university computer science courses — universities in NZ typically have about 10 to 20% female students in first-year computer science classes — so there is some hope that offering computer science earlier in the education system may reduce the gender imbalance.

3.2 Public exemplars

For privacy reasons our main analysis of student work in Section 4 is limited to broad statistics and observations about the sample reports. However, there are some public samples of student work: after the 2011 reports were marked, a set of “exemplar” reports was selected (two or three at each level of A/M/E) for publication⁵. The reader is encouraged to view these to get an idea of the kind of work students are submitting, and we will use some examples from these in our discussions. The exemplars include annotations from the markers to show how the criteria have been met.

The exemplars are provided to guide teachers on the standard of work that is expected, but because they are public, it has to be assumed that students will have access to them. This has both benefits and problems; modelling good practice is clearly helpful pedagogically, but it also provides the

⁴<http://www.nzqa.govt.nz/assets/About-us/Publications/ncea-annualreport-2011.pdf> (page 13)

⁵These are available from <http://www.nzqa.govt.nz/assets/qualifications-and-standards/qualifications/ncea/NCEA-subject-resources/Technology/91074-exp-2011.zip>.

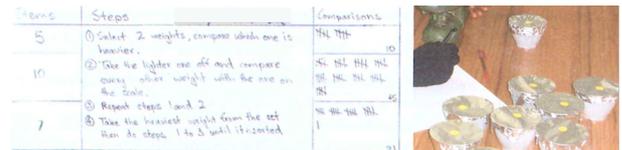
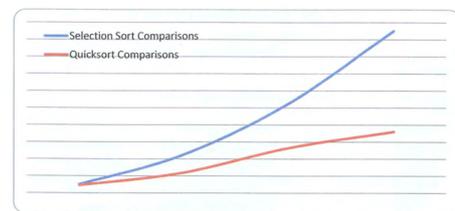
Table 1: Student success rates in Programming and Computer Science standards 2011

	Candidates	Gender	N/A	A	M	E
AS91074 Demonstrate understanding of basic concepts from computer science	654		32.7%	37.2%	18.2%	11.9%
Male	475	72.6%	33.3%	36.0%	18.3%	12.4%
Female	179	27.4%	31.3%	40.2%	17.9%	10.6%
AS91075 Construct an algorithmic structure for a basic task	2131		34.3%	31.5%	16.1%	18.1%
Male	1493	70.1%	34.3%	31.3%	15.0%	19.4%
Female	638	29.9%	34.3%	31.8%	18.7%	15.2%
AS91076 Construct a basic computer program for a specified task	3245		31.8%	33.2%	18.0%	17.1%
Male	2082	64.2%	29.9%	33.4%	17.6%	19.2%
Female	1163	35.8%	35.3%	32.8%	18.7%	13.3%

temptation for students to simply paraphrase the model answers in attempt to imitate the best practice, rather than generate their own authentic report from scratch.

The first Excellence exemplar covers the criteria for the standard, although it isn't perfect. It draws a sound conclusion about binary vs linear search, including analysing the time needed for linear and binary search of 100 items; linear search was estimated as "1-100" comparisons; binary search is estimated at 7, calculated by repeatedly dividing the number in half (i.e calculating $\log_2 100$ without using logarithms). The material about programming languages compares Pascal and Scratch with binary code, which provide a good contrast. The HCI report looks critically at some aspects of two mobile touch-screen devices; it is a little ad-hoc, but does provide a balanced view based on personal experience. This report illustrates the holistic marking and the focus on the criteria: despite being rated an Excellent report, it does contain some confused reasoning; for example the report claims "A binary search uses the yes/no, on/off, 0/1 concept" (perhaps confusing binary search with binary numbers); and tells us that "[Pascal] cannot be used with a mouse, and can only be controlled by the keyboard". These are reminders that students are encountering the terminology and environments for the first time and are still making sense of them.

The second Excellence example is longer and more carefully presented. It compares two sorting algorithms using the a balance scale activity from CS Unplugged (csunplugged.org/sorting-algorithms), and includes notes and photos of the activities (Figure 1) that provide good evidence that the student has engaged in the process. The number of comparisons used for $n = 5, 10, 15$ and 20 are given for a single manual simulation of selection sort and quicksort using a balance scale. The report provides a graph without axes (Figure 2), which shows the difference between the two algorithms, and a good conclusion is drawn, that the "lower cost [of quicksort...] will become more noticeable as the number of items compared increases." This is an articulation of the difference between n^2 and $n \log n$ time; it does have a slight imprecision in that the word "compared" should be "sorted," but it is reasonable evidence that the student understands the relationship, which is the goal of the standard. The HCI work in this exemplar mentions a visit to a university Usability Laboratory, and this would have been an excellent opportunity to see HCI evaluation happening in an authentic context and no doubt contributed to the

**Figure 1: Notes and photo of experiment using balance scale activity from an Excellence exemplar****Figure 2: Graph from an Excellence exemplar**

student's appreciation of the topic.

At the other end of the scale, the Achieved exemplars show work from students who have engaged with the topic, but haven't made in-depth observations. For example, the first Achieved exemplar discusses bubble sort, and gives a trace of the algorithm being applied to 8 names (Figure 3). At the end of this example the writer gives the number of comparisons and swaps for that example, but makes no comment comparing this with different values of n , or with different algorithms. The second Achieved exemplar uses a different approach, showing a screenshot from an on-line visualisation of bubble sort (Figure 4), and using the statistics from the visualisation to determine the cost of the algorithm. In fact, the exemplar goes on to compare bubble sort and selection sort for different values on n , but there is no explanation of how these were measured, and more significantly, the other sections on programming languages and usability are weak, and hence the overall grade is Achieved.

Exemplars are also available for work meeting the Merit criteria. These show more advanced understanding and discussion than the Achieved exemplars, but don't have the completeness of the Excellence ones.

Doing well in the report will be easier for students who are good at writing, and because there are also elements of math

I have eight names that need to be sorted into alphabetical order: Hannah, Angela, Ella, Bob, Flora, Campbell, Darrell and George.

- 1) You will start off comparing the first two names which are Hannah and Angela. Angela is first in alphabetical order so Hannah and Angela swap places.
- 2) Next you compare Hannah and Ella. Ella is first in alphabetical order so Ella and Hannah swap places.
- 3) Next you compare Hannah and Bob. Bob is first in alphabetical order so Bob and Hannah swap places.

Figure 3: A trace of bubble sort from an Achieved exemplar

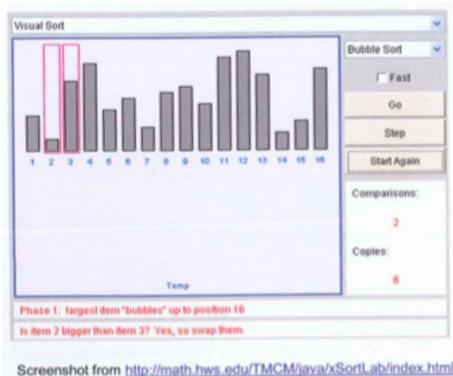


Figure 4: Using a visualisation to demonstrate an algorithm from an Achieved exemplar

and scientific experimentation, students with strengths in science and math will also find it easier to do well. We consider this to be a positive, since writing, math and general science skills are valuable for computer scientists, but this emphasis represents a significant change from the computing classes before 2011 which were generally focussed more on *using* computers and tended not to attract students with good science and math skills.

It's also important to note that these students are working on topics that have traditionally been taught to university students, and one has to avoid the trap of expecting them to perform at the same level as more experienced students. Inevitably the average level of experience in writing, science and math will be lower, and it is important to calibrate expectations appropriately.

3.3 Markers' assessment report

After the grading of the reports was complete (January 2012), the material was returned to the students and a report from the markers was released⁶.

The markers' report began by commenting on the quality of the presentation. For example, the report notes that "small screen shots, text too small to read, and graphs with unlabeled axes did not contribute to a demonstration of understanding." Learning to conform to guidelines is good practice for future CS practitioners, and being in the habit of presenting data clearly is valuable! This is a benefit of requiring a report from students rather than examinations, and helps

⁶<http://www.nzqa.govt.nz/nqfdocs/ncea-resource/reports/2011/level1/technology.pdf>

to emphasise the importance of communication skills. It also reflects the value of students being aware of good experimental method (particularly for reporting on algorithm performance). Those who had a weaker science background might not be familiar with the process of collecting, reporting and discussing results in a precise manner.

Another issue raised by markers was the personalisation of the work — the markers said that "candidates who clearly demonstrated understanding of basic concepts from computer science wrote in their own voice, providing evidence from their own work and experience to support any factual or referenced material." The markers also pointed out that reports that were simply paraphrasing other sources didn't make it clear that the student had understood the topic. The public availability of exemplars will make it even more important for the student's "voice" to be heard i.e. to make sure the report is from personal experience on examples that are likely to be unique to the student.

Regarding the human-computer interaction work, the markers observed the common problem of confusing "functionality of devices with usability." Students very quickly focused on the physical form of an interface (buttons and menus), or the functionality of the software, without identifying the human factors that could make it difficult for a user to find their way around the interface.

The markers' report concludes with a summary of what was required at each level, which is essentially the criteria of the standard, although it includes typical problems found in "Not Achieved" work, which (apart from not meeting the criteria) were a lack of detail in descriptions, paraphrasing without understanding, and not addressing all the requirements when using a template.

The marking process is challenging because the standards are very open-ended, and work can be submitted based on a wide range of contexts. The standards are also new and there is no past history to draw on. We note that markers need to be well briefed, and provided with detailed guidelines. In particular they need to be familiar with relevant material from popular sources on the web (starting with but not limited to Wikipedia) so as to recognise material that has been used without understanding. This is particularly important in some sub-areas (like programming languages).

4. ANALYSIS OF STUDENT PAPERS

To understand the student work better we manually analysed in detail 151 sample reports from the 654 AS91074 submissions in 2011, noting the choice of examples that each student used and how they presented their understanding of the topic. The sample was selected by NZQA, who are responsible for grading the student work. All of the sampled papers were marked by one marker; the marking process has checks in place to ensure consistency between markers so this should not introduce a significant bias. The selection process could not be rigorously randomised, but because student submissions are essentially shuffled to prevent one marker getting all the reports for one school, the sample is reasonably broad. Table 2 shows the grades of the sampled reports compared with those of all submissions.

Table 2: Student success rates in AS91074 2011 (all submissions and sample analysed)

Source	Number	N/A	A	M	E
All submissions	654	32.7%	37.2%	18.2%	11.9%
Sample analysed	151	36.4%	35.1%	18.5%	9.9%

Table 3: Number of pages in student submissions for AS91074 in 2011 (sample of 151 submissions)

	All	Not Achv	Achieved	Merit	Excellence
Submitted pages					
Average	6.5	4.6	6.4	8.4	10.9
Min	1	1	2	4	8
Max	16	13	14	13	16
Student-written content					
Average	5.4	3.4	5.2	7.3	9.8
Min	0.5	0.5	1.5	3.5	6.5
Max	16	12.5	11.5	12	16

We discuss the student work here first in terms of the length of the document, then under the three main topics of the standard: algorithms, programming languages and usability. We conclude by discussing general issues with the writing style, and the effect of teacher advice on the students' work. As would have been the case for grading the student work, some of our analysis of the reports involved judgement calls on what a student's intention was, including dealing with inaccurate descriptions and confusing text, so the figures reported below are approximate, but indicative of the kind of work that students submitted.

4.1 Report page length

The reports are limited to 14 pages (and markers will not mark any additional pages). An important question is whether this limit constrains the students. The number of pages submitted by students in the sample of 151 analysed documents is shown in Table 3. The first set of figures shows the number of pages the students submitted, and the second set shows our estimate of actual student-written content, which ignores the teacher-generated rubrics (which is the most common "padding" in the submissions), blank areas, tables of contents, and reference lists (which were rare).

Only 4 of the 151 sampled submissions used the full 14 pages (in fact, one of those 4 used more than 14 pages, but the extra pages would have been ignored by the markers). Most students did not seem to be unduly constrained by the limit, and 91% of the submissions used 11 pages or fewer.

The average number of pages grows with the quality of the work, which isn't surprising since more information is needed to cover all the criteria for Excellence. However, having a long document doesn't necessarily predict a good grade; one 13-page submission did not even reach the Achieved level because most of the space was taken up with a lot of detail of algorithm execution without discussing the bigger picture such as the performance of the algorithm. One student was awarded Excellence for just 6.5 pages of writing. In this case the student didn't use any images, but had clear explanations and examples which showed understanding of the topic; for example, binary and linear search were compared for large values of n , providing a convincing contrast.

Two of the longest Excellence submissions used a lot of images, including several screen shots of multiple interfaces. The screen shots were relevant, although not essential, and the work could have been presented in slightly less space. Another 14-page Excellence submission was primarily text, and contained considerably more detail and covered a wider range of concepts than would be necessary to meet the Excellence criteria.

Some of the longer submissions had extensive traces of algorithms that went over multiple pages. Often these were presented inefficiently — for example, one trace of a sorting program listed every comparison made, one per line, which was very difficult to read, and took four pages to describe sorting eleven numbers. Other students managed to convey similar traces in a fraction of the space by using better layout which more clearly showed the big picture of what was happening as well as making each step clear. Since layout isn't one of the criteria, both approaches should be accepted, but teachers advising students on submission lengths would need to take into account how compact a student's representations are.

Based on our observations, the limit of 14 pages seems suitable as an absolute maximum, but students could be advised that about 10 pages of their own writing is usually sufficient to cover the Excellence criteria, and longer documents would be required only if they rely heavily on images in their descriptions. It is important that students focus on the criteria of the standard and cover all the requirements.

Not all students addressed all three areas. Two students did not attempt the algorithms topic; seven made submissions that didn't touch on programming languages, and nine didn't tackle usability. It would appear to indicate that these few students simply ran out of time to complete all the criteria, but submitted their work anyway (none of them received a passing grade since there were criteria that were clearly not met.) Most of these gave the impression of a hasty attempt, but one student did a good job of two of the topics, and would have done well if they had submitted something on usability.

4.2 Algorithms

For the algorithms topic, the main criteria were to differentiate an algorithm from a program and informal instructions, and to discuss the "cost" of an algorithm. Almost any example could be used to discuss the first criterion, but measuring the cost required an algorithm that had interesting behavior based on its input size n . This criterion requires a good definition for the cost; generally it would be the running time of a program or the number of steps taken by the algorithm. In some cases memory requirements or disk accesses might be considered as a cost, although this wasn't used in any of the reports analysed here. Unfortunately some students misunderstood the cost to be the *length* of the program or algorithm, which painted them into a corner since this couldn't be measured based on the size of the input, n .

About 63% of the students used sorting algorithms as their example, and 19% used searching. Many students met the Achieved criteria or better for the algorithms part using these examples. About 9% of the students appeared to use

either their own program or another algorithm as an example, and these were generally not very convincing for meeting the criteria (most of them received “Not achieved” for the standard). An example used by one student that was (just) suitable for meeting the basic criteria was the Towers of Hanoi solution, although it wasn’t feasible to compare two different algorithms with different costs with this example, and hence couldn’t be used for the Excellence criterion. At least five students attempted to discuss the cost of an algorithm without using an example at all — none of their submissions met the criteria.

The most popular sorting algorithms were bubble sort and quicksort (each was used in about 31% of the submissions). The high use of bubble sort is likely to reflect its popularity with teachers more than its pedagogical value [1]. Quicksort provides a strongly contrasting example to compare with quadratic time sorting algorithms. The other popular sorting algorithms were selection sort (used in about 22% of submissions) and insertion sort (20%).

To meet the Excellence criteria, students needed to choose two algorithms to compare, and this was attempted by about 36% of the students. About 26% of them compared the performance of two different sorting algorithms, with about 16% comparing an $O(n \log n)$ algorithm with an $O(n^2)$ one, giving the opportunity to observe a strong contrast in performance. About 10% compared two $O(n^2)$ sorting algorithms (mainly two of bubble sort, selection sort and insertion sort), which didn’t illustrate the contrasting performance differences that can occur between different algorithms for the same problem, although there were constant factor differences observed. About 11% of the students used two different searching algorithms for the comparison — all of these compared linear search with binary search, which provide a strong contrast in cost. One Excellence student also used hashing (based on the CS Unplugged searching activity), which led to a worthwhile discussion.

The values of n that were chosen for comparing algorithms were often too small to make good observations. It wasn’t unusual for student to report performance for searching or sorting up to just 10 or 20 items, but it is at 100 or even 1000 items that some algorithms start to show compelling performance differences. Very few students explicitly observed that the relationship between two algorithms was non-linear, although a few of the students who evaluated searching algorithms did recognise the significant difference between their measurements which reflected $O(n)$ compared with $O(\log n)$ time — these were mainly students who scored Excellence overall. Note that students don’t have to implement binary search (which is notoriously hard to get right [12]), but can run a provided program that performs the search and counts comparisons, or use a visualisation (several are available online) that reports the time or number of comparisons made. Similarly, they don’t have to do a formal analysis of the cost, but could either plot data points and talk about trends, or reason about it based on repeatedly halving n .

In general, students took a simplistic approach to analysing their algorithms, and they need to be made aware of the idea that algorithmic complexity can be non-linear, which is a crucial factor surrounding the scalability of algorithms

(for example, a quadratic sorting algorithm given 10 times as much input takes about 100 times as long to run). Some weren’t able to observe the difference because the algorithms they chose had the same complexity, and thus only showed a small constant difference; others didn’t notice significant differences because they only used small values of n . Only three of the submissions analysed used a graph to show the algorithm performance; others used tables, or simply presented results in an uncollated fashion. The lack of graphs would inevitably make it hard to see trends, although some students were able to describe them from tables of figures. Since the significant differences are likely to be unexpected, students need to be guided to look for them, for example, by encouraging them to experiment with very large values of n , and to chart the results through a spreadsheet, which makes it easier to deal with a larger number of observations.

The choice of algorithms can clearly make a difference to the richness of the student report. The most effective choices were the pairs of algorithms with significantly different asymptotic complexities (binary search *vs* linear search, and quicksort *vs* insertion/selection/bubble sort). Both the binary/linear search and quicksort/quadratic sort pairs have costs that differ by a factor of $n/\log n$, which provides a obvious contrast for large values of n .

Students don’t need to implement the algorithms to meet the criteria, and in fact the techniques needed (e.g. arrays/lists) are beyond the requirements of the programming standards at this level. For students who have more programming experience, the simpler algorithms (linear search and quadratic sorts) might serve as good programming exercises but the difficulty is that a student must get the programming exercise completely correct in order to be able to use them for valid experiments (for example, one student gave results that showed Bubble sort to be 73 times faster than Insertion sort, which almost certainly reflects an issue with the implementation given that Bubble sort is usually slower than Insertion Sort). While quicksort is harder to implement, a list-based version of it (where the list is duplicated at each partition) is relatively easy to implement, and would still demonstrate the performance improvement required. Binary search is also easy to understand but difficult to get exactly right [12]. Interestingly, some students explained in their reports about why binary search is so hard to implement; it is good to see them sensitised to issues surrounding program correctness. In general we would recommend that students experiment with programs they are provided with (or visualisations), but interested students could be encouraged to implement their own version as an extra exercise to deepen their understanding, but not use those versions to measure timing.

The use of bubble sort tended to be a distraction, as it is very closely related to other quadratic sorting algorithms, involves more steps for students to trace, and has a tradition in algorithm pedagogy that paradoxically has made one of the worst sorting algorithms one of the most well known [1]. As a contrasting algorithm to quicksort, selection sort is useful because its performance is easily analysed using high school math, and insertion sort is useful because it has a contrasting best/average/worst case.

One complication with the algorithms topics is that a closely related standard (AS91075) on designing a computer program used the term “algorithmic structure” to describe the design of the program. This appears to have been confused with the use of “algorithm” in the AS91074 standard; some online postings from teachers indicate that they regarded them as describing the same thing, and at least one student stated that an algorithm is a program plan, which is essentially correct, but confusing in this context. Thus some submissions on algorithm “cost” are based on a student’s own design of a program, which typically doesn’t have any interesting behaviour to analyse, and in fact may be $O(1)$ since the program does a very routine task. For example, several programs simply read in a value, did a calculation and printed the result. The use of the word “algorithm” has since been removed from the AS91075 standard to avoid this confusion.

4.3 Programming languages

The criteria for programming languages centre around the role of programming languages, high and low level languages, and how high level languages are translated to low level ones (i.e. compiling and interpreting).

Most of the students described high and low level languages and compilers abstractly, but at most a third attempted some sort of concrete demonstration of a compiler being used. A demonstration is much more convincing of student understanding because the student has personally had the experience of converting a program to a low level language and running it. Even better than demonstrating the process on a given program, students could have used their own program (for example, one done for AS91076) as the example, showing how it is compiled and/or interpreted. Because the standard discusses both interpreted and compiled programs, it’s unlikely that students would be familiar with two different languages and thus only one of the examples could reasonably be personalised, although some languages have both compiled and interpreted versions (e.g. Basic) and these could be contrasted. Another approach would be for students to learn the bare minimum of a language e.g. write a “Hello world” level program but substituting their own name. None of the submissions seemed to have taken this approach, and it would be interesting to evaluate, since it forces students through the process of getting the program to run. Using the student’s own program was risky for the algorithms topic, but for the programming languages topic it provides good personalisation and shows an awareness of the process, and it isn’t sensitive to the quality of the program (as long as it runs or compiles). Despite this, we observed few examples in our sample where students had obviously used their own program.

In the student work analysed, the main languages mentioned as examples were Visual Basic/Basic (in about 18% of reports), Scratch (13%), Pascal (5%), Flash/Actionscript (4%), C (4%) and Alice (3%). Other languages used included C++, Java, JavaScript, Picaxe, and Python. About a half the reports didn’t use an example in a specific language, and these generally didn’t reflect a lot of understanding since the concepts were explained in very abstract terms.

Projects that compared a compiled and interpreted language

mainly used Visual Basic, C, Pascal and Java as examples of compiled languages, and Scratch, VBA in Excel, Alice, JavaScript and Python as examples of interpreted languages. The situation isn’t always simple, as modern languages can blur the distinction of compiling or interpreting (e.g. Java compiles to byte code, which is then interpreted). One student mentioned how Scratch can run as a Java applet itself, so it is interpreted twice *and* compiled! If students can fully understand these complexities then they will have a good grasp on the issues intended, but it will be important for teachers to provide guidance on the languages to explore.

Another example used in several reports was Visual Basic (compiled) compared with Basic (VBA) macros in Excel (interpreted). The contrast is effective because it is essentially the same language and focuses the student on the different way it is run rather than the difference in language; the examples we observed using this contrast all appeared to meet the criteria for Excellence in the programming languages requirements. We would note that JavaScript is a useful language to explore because it is definitely interpreted, readily available, and is not confused by the drag-and-drop aspect of Scratch, whereas it is problematic to use HTML as an example of an interpreted language (as one project did) since it is not at all clear that HTML is usefully viewed as a programming language, even if it is a formal markup language.

Some students presented their work on programming language as posters. This approach has potential since the student is writing for a particular audience (probably other students), and thus might be expected to focus the explanations and examples more. It also seemed to discourage paraphrasing, perhaps because they were thinking of it as communicating to peers rather than trying to write something that a teacher would recognise as correct.

4.4 Usability

The criteria in the standard emphasise factors of a user interface that contribute to its usability. Unfortunately many reports seemed to confuse usability with functionality, or focussed on the physical layout of the interface. For example, a student might comment that a cellphone can take photos (the functionality), but miss the point that the typical task that a user has (such as taking a photo and transferring it to an online photo album) might involve pressing mysterious key sequences, dealing with incompatible devices, and using a time-consuming procedure that is difficult to remember (usability). The better work focussed on typical tasks done by the user, and any problems encountered completing those tasks.

Using a personally owned device and talking about the student’s own experience gives a clearer demonstration of understanding than writing about interfaces abstractly. About 61% of the reports provided a suitably personalised report. Those that didn’t discuss a personal device mainly gave general usability principles that appeared to be paraphrased from standard sources. In these cases, where illustrations were given they tended to be standard examples of usability issues, and once again it is hard to hear the “student voice” when this approach is used.

The students who reported on experience with their own dig-

ital system used a variety of interfaces as examples: about 29% of the reports evaluated some sort of mobile phone (simple phones, smart phones and iPhone related devices) or some feature of one (such as sending a text message or taking a photo). The main other commonly chosen devices were an iPod/Touch device, used in about 7% of the reports, and a video game controller (5%). Other interfaces appeared less frequently, including a calculator, CD player, DVD player, DVD recorder, email software, games, web browsers, web sites, an mp3 player and a tape player. Even a toaster was used to provide a reasonable evaluation.

Ten students reported on an interface that they had written themselves. These generally weren't good as examples: few of them appeared to meet the criteria, and the remainder were focussed on describing their interface without discussing it from the user's point of view. It would appear that these students had tried to combine this standard with work done for the design and programming standards. For user interfaces this makes it difficult for the student to provide an objective and critical evaluation.

An important step in usability assessment is to identify the tasks that the device is to be used for, which helps to take the focus away from features and layouts, and put it onto the sequence of operations that a user would perform to achieve their goals. It is not unusual for a device to be frustrating to use because in practical situations the functionality of the device can be difficult or slow to access, and this is revealed if a task is evaluated instead of just a feature of the device. Only about 36% of the submitted reports described the task that a user might do.

Another useful approach that can pick up usability issues is observing another person using the interface. Only 9 students did this, and all these projects clearly met the criteria; a third-person observation would apparently make it easier to report on usability, although the sample is too small to draw firm conclusions. Some of the students who observed another person took notes of every step the person took. This approach was particularly convincing when the student described their observations, and it set the student up well to meet the Merit criterion for usability, and Excellence if they compared another device.

In general it seemed that the students had little teaching on HCI and usability evaluation. Although not required at this level, introducing usability heuristics (e.g. from useit.com) provides a tool for students to analyse systems. Another approach is the *cognitive walkthrough*, which is described for high school students at www.cs4fn.org/usability/cogwalkthrough.php. This encourages students to observe someone else using the system, which makes the evaluator more aware of the steps being made.

4.5 General writing issues

A key phrase in the title and wording of the standard is for students to “demonstrate understanding” in order to meet the criteria. It is tempting to respond to criteria such as “explaining the need for programs to translate between high and low level languages” with abstract textual explanations that are paraphrased from books or online sources (and in fact the instructions for the report explicitly say that para-

phrasing is acceptable). Although the general instructions *allow* paraphrasing, a more compelling way to show understanding would be for a student to provide a worked example using a context unique to themselves (for example, analysing the interface of their own mobile phone, or running experiments on the speed of an algorithm on their own computer). In many cases we found that “showing an understanding” was interpreted as simply explaining the concept. In this case the process could end up grading Wikipedia's knowledge, and any slips in the paraphrasing process only served to show that the student *didn't* understand the concept!

A common problem in the student reports is giving poor explanations of what they had done. For example, in explaining sorting, one student wrote that B is bigger than A, and A is bigger than C, but we weren't told what A, B and C were — quite possibly they were sorting weights or labeled envelopes, but this wasn't mentioned at all. Another problem was assuming that the marker would know things that the teacher did (perhaps because students are used to internal assessment); for example, referring to the “method that was used in class” rather than describing it.

Many reports showed a lack of scientifically convincing reporting, such as drawing conclusions from just one speed measurement from each of two algorithms, or having results spread around the document and not explicitly comparing them. It was also not unusual for the student to assume that the reader would go to the trouble of finding patterns in the results for themselves; for example, one student had a table of execution times that showed a clear difference between selection sort, bubble sort and quicksort, but they never articulated that they had noticed the difference, and thus got credit for measuring the algorithms, but not for showing an understanding of the different costs.

The work on the programming languages topic overwhelmingly used paraphrased quotes. This is largely because it is a lot harder to personalise (there are only so many compilers and machine languages that students are likely to encounter, and the role of interpreters and compilers is fairly well defined). A similar situation arises with the requirement to discuss the relationship between algorithms and programs; there are only a few ways to describe this accurately. In these cases the criteria aren't ideal for a report and it would be better if they prompted examples rather than factual information.

4.6 Guidance for students

The nature of the standards is such that teachers can direct student work in a variety of ways to enable them to demonstrate their understanding. Many of the projects reproduced questions and rubrics from teachers, and the nature of these instructions had an influence on how easy it would be for students to produce work that met the criteria.

Several sets of the sampled reports had identical questions heading each section, which either would have come from the same teacher, or a group of teachers who had shared teaching ideas.

Because the quality of student work depended a lot on the tasks that were set for them, we have attempted to identify

groups of submissions that seem to have been set the same tasks. Because teacher peer support means that teaching material is shared, this does not necessarily mean that the similar papers have the same teachers or are even at the same school, although inevitably those who are in the same class probably were given the same assignment.

For example, in one set of reports which had the same questions, several students compared Insertion sort with Selection sort, which will have a constant-factor difference. Many of the students correctly concluded that Insertion sort is twice as fast as Selection sort, but missed the opportunity to explore the possibility of a difference that was more than a constant factor.

Some of the questions were presumably intended to stimulate discussion, but seemed to be treated by students as an exam, resulting in very short answers. For example, a series of questions about choices of programming languages in one group resulted in very brief answers of just a few words. Although the questions on programming languages may not have had the intended effect, the questions on usability for the same group resulted in generally good responses from the students. For this the rubric prompted the students to choose someone who isn't a digital native (e.g. parent or grandparent) and observe them using a device for about an hour.

One group that did particularly well had prompting questions that didn't just mirror the criteria, but started off with a creative situation for the students to get them thinking beyond just checking off the criteria. For algorithms, the students were prompted to get the names of favourite songs from five friends, and then explain searching algorithms using that list. This provided a familiar context (searching for songs on an mp3 player), and a meaningful task (selecting a favourite song). It also ensured unique lists for each student, even though they were doing the same task. The usability section for this group began by asking students to describe the context the device is used in, and the role of the interface. Once again, this prompt doesn't directly address the criteria, but it starts the students thinking of the big picture, and resulted in good quality answers.

Now that the standard has been in use for a year, teachers will be able to share best practices based on how well students were able to respond to the various prompts.

5. CONCLUSION

The analysis above has provided some specific ideas on how to teach and assess the three topics in the new standard. The following general observations are based on the analysis of student work.

Teacher professional development and support is essential: Because the new standard was introduced very quickly, and because computer science is a new subject in New Zealand schools, teachers could not be expected to be well prepared to teach the new material, either in terms of subject knowledge or pedagogical knowledge. Unfortunately this was reflected in the student work, particularly in the guidance of topics they were given for their investigations (e.g. comparing two algorithms that didn't have contrast-

ing performance, or investigating only small values of n). Teachers need to be guided on what the main messages are that students should take away from this experience, and be given recommendations of examples that illustrate these in a compelling way — in this case, contrasting algorithms, contrasting programming languages, and interfaces that are small enough to evaluate but have interesting usability issues.

Students do better when they personalize their work:

When students report on their own experiences, whether it is running an algorithm on their own computer or evaluating a device they have chosen, their work carries a lot more authenticity than abstract discussions of a concept. It also moves their level of understanding up Bloom's taxonomy: descriptions of a concept often come across as a paraphrase that at best reflects remembering facts, while discussions based on a unique example reflect the student applying their knowledge.

Assessment using reports: The AS91074 computer science standard is assessed by a student report submission. For most of the criteria this provides an opportunity for students to report on a rich personal experience, such as experiments with algorithms and evaluation of interface usability with real users. Through project work they get to experience the issues first-hand (such as a slow algorithm or a frustrated user), and get experience communicating in writing about computer science. However, some of the criteria (e.g. "describing... the function of a compiler") invite responses that are just paraphrases of standard definitions, and would benefit from clarification or re-wording (e.g. "demonstrate the function of a compiler using an example") to encourage students to explore the concept in a context rather than simply describe it.

Using provided systems rather than having students build their own:

The best student work was done when they assessed the performance of a *provided* implementation of an algorithm; students who implemented their own programs tended to be distracted by the implementation, were limited in the algorithms they could use, had potentially unreliable performance results, and were biased in their usability assessments. Because the standard is about algorithm performance and not programming, those who focussed on their own program were at risk of missing the big picture, or else might implement algorithms that didn't show a good contrast. Likewise, students who discussed usability based on interfaces they had implemented couldn't see their weaknesses. More advanced students might be expected to implement their own systems and evaluate them, but at the beginning high school level the messages about the key topics are likely to come across more clearly if students start with a working system.

The introduction of computer science to NZ schools has addressed a need that was articulated by industry and teachers, and has happened remarkably quickly. The work done by students shows some pleasing levels of understanding of computer science for some; others have at least had experiences that indicate that they have had a taste of what kind of topics might come up in computer science courses. With very little lead-in time hundreds of students have studied

computer science, and with the lessons learned in the first year, the quantity and quality of students is only likely to increase in the future.

6. ACKNOWLEDGEMENTS

We are grateful to Scott Telfer (NZQA) for assisting with access to key data.

7. REFERENCES

- [1] O. Astrachan. Bubble sort: An archaeological algorithmic analysis. *ACM SIGCSE Bulletin*, 35(1):1–5, 2003.
- [2] O. Astrachan, J. Cuny, C. Stephenson, and C. Wilson. The CS10K project: mobilizing the community to transform high school computing. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, SIGCSE 2011*, pages 85–86, 2011.
- [3] T. Bell, P. Andreae, and L. Lambert. Computer Science in New Zealand High Schools. In T. Clear and J. Hamer, editors, *ACE '10: Proceedings of the 12th conference on Australasian Computing Education*, volume 32 of *Australian Computer Science Communications*, pages 15–22, Brisbane, Australia, Jan. 2010. Australian Computer Society, Inc.
- [4] T. Bell, P. Andreae, and A. Robins. Computer Science in NZ High Schools: The First Year of the New Standards. In L. A. S. King, D. R. Musicant, T. Camp, and P. Tymann, editors, *Proceedings of the 43rd ACM technical symposium on Computer Science Education, Raleigh, NC, USA*, pages 343–348, New York, 2012. ACM.
- [5] T. Brinda, H. Puhlmann, and C. Schulte. Bridging ICT and CS: Educational standards for computer science in lower secondary education. *ACM SIGCSE Bulletin*, 41(3):288–292, 2009.
- [6] A. Bruckman, M. Biggers, B. Ericson, T. McKlin, J. Dimond, B. DiSalvo, M. Hewner, L. Ni, and S. Yardi. “Georgia computes!”: Improving the Computing Education Pipeline. *ACM SIGCSE Bulletin*, 41(1):86–90, Mar. 2009.
- [7] T. Carrell, V. Gough-Jones, and K. Fahy. The future of Computer Science and Digital Technologies in New Zealand secondary schools: Issues of 21st teaching and learning, senior courses and suitable assessments. Technical report, 2008.
- [8] S. Furber, editor. *Shut down or restart? The way forward for computing in UK schools*. The Royal Society, London, 2012.
- [9] J. Gal-Ezer and C. Stephenson. The Current State of Computer Science in U.S. High Schools : A Report from Two National Surveys. *Journal for Computing Teachers*, Spring, 2009.
- [10] J. Goode and J. Margolis. Exploring Computer Science. *ACM Transactions on Computing Education*, 11(2):1–16, July 2011.
- [11] G. Grimsey and M. Phillipps. Evaluation of technology achievement standards for use in New Zealand secondary school computing education. Technical report, New Zealand Computer Society (NZCS), Wellington, 2008.
- [12] R. E. Pattis. Textbook errors in binary searching. *ACM SIGCSE Bulletin*, 20(1):190–194, Feb. 1988.
- [13] A. Robins. Learning edge momentum: A new account of outcomes. *Computer Science Education*, 20:37 – 71, 2010.