TEACHING & LEARNING
RESEARCH INITIATIVE
NĀU I WHATU TE KĀKAHU, HE TĀNIKO TAKU

Supporting teachers and learners of programming by understanding feedback on syntax, semantics, and style

Diana Kirk, Andrew Luxton-Reilly, and Ewan Tempero

May 2020

Introduction

New Zealand has a shortage of skilled workers in information technology (IT). Skills relating to computer programming (e.g., software engineer, applications programmer, software tester, web developer) all appear on the 2019 Long Term Skill Shortage List published by Immigration New Zealand (Immigration New Zealand, 2019). In 2020, Immigration New Zealand reports a continued lack of information and communications technology (ICT) workers in New Zealand (Immigration New Zealand, n.d.). However, New Zealand is competing in a global market facing similar demands. In many parts of the world, demand for computing graduates outstrips supply. Consequently, New Zealand cannot rely on filling its requirements from other countries. We need to develop skilled professionals domestically.

In the education sector, the importance of digital technologies in New Zealand has been recognised by the Ministry of Education with the introduction of digital technology into the schools technology curriculum (Ministry of Education, 2018a). Reasoning about problems and applying problem-solving strategies derived from computing is now considered to be an essential skill in modern society. One strand in the technology curriculum contains the subcategories programming, algorithms, and data structures. These areas directly relate to computer programming and are thus crucial areas for addressing the reported skills shortages in New Zealand.

This means that more teachers will be required to teach the more fundamental concepts in these areas. The need for knowledge relating to how to teach programming, algorithm development, and computational thinking to a high standard is of primary importance for teachers in New Zealand schools, particularly for teachers at senior secondary level. However, despite widespread recognition of the importance of educating programmers, many secondary school teachers in New Zealand have no programming experience and no formal qualification in computer science. In 2013, Thompson and Bell found a lack of ready-to-use material designed specifically for the New Zealand curriculum (Thompson & Bell, 2013). The authors also found that many teachers lack confidence in teaching the National Certificates of Educational Achievement (NCEA) programming curriculum and assessing the programming standards (Thompson & Bell, 2013). A companion study identified the clear need for professional development of, and support for, teaching of programming in New Zealand schools (Thompson et al., 2013). Teachers participating in a 2-day professional development workshop on teaching programming struggled to master technical aspects of computer programming, and expressed concerns about teaching programming at more advanced levels to senior students (Haden et al., 2016). Faulkner et al. note the lack of rigorous research on programming in high schools that may be used to guide teachers in delivering and assessing curricula and standards (Faulkner et al., 2014).

In this project, we hoped to better understand and improve teacher–student interaction when learning programming at senior secondary level by exploring the following themes:

- the nature and quality of feedback delivered by existing teachers of programming
- differences in feedback that are due to teacher subject expertise
- teacher understanding of what students are struggling with, and what information teachers seek, to provide better feedback
- use of secondary sources of feedback, such as automated tools
- how to improve the quality and timeliness of feedback experienced by students.

The NCEA programming standards require teachers to make judgements about whether a student's program runs (*syntax*) and functions correctly (*semantics*) but also expect that the student has produced well-structured code (*style*). The latter is of key importance for reducing the costs to *maintain* a program. It is widely accepted that the biggest cost to a software organisation comes after a software application has been first delivered to a client. The reasons are that applications are extended to include new functionality, ported to other environments, or that errors are found in the application and must be fixed. To create maintainable applications and thus reduce these costs, program code must be well-structured (i.e., written with good *style*).

There are thus three aspects of programming that must be considered:

- Syntax concerns the symbols that make up the written code. If these are incorrect, the program cannot be run.
- Semantics concerns the correct functioning of the program. A program may appear to run successfully, but some parts might work incorrectly (i.e., the semantics are incorrect).
- Style relates to how well structured the code is (i.e., whether it will successfully support lower maintenance costs).

Our goal was to build understanding of how teachers and students use feedback on syntax, semantics, and style to improve programming skills. To achieve this, we collected baseline data and developed and evaluated interventions that would improve the use of feedback.

Approach

Research strategy

To investigate our research questions, we organised the project into three conceptual phases. Phase 1 involved the collection of baseline data to determine how teachers and students are currently using feedback. In Phase 2, we developed worksheets to address issues relating to programming feedback on syntax, semantics, and style. In Phase 3, we used the worksheets in teacher workshops and used feedback from teachers to evaluate and improve these as resources for teachers and students.

We approached the research from three directions.

1. Talking to teachers

We wanted to gain a deep insight into the issues experienced by teachers when providing feedback to senior programming students. Our strategy was to speak to small groups of teachers in an informal, face-to-face situation. To achieve this, we ran a series of informal **instructional** and more formal **presentation workshops**. Informal workshops were hosted with the intention of identifying issues and understanding the issues experienced by inexperienced teachers and their viewpoints on the kinds of support that would be most helpful. They were held at locations convenient to the teachers. We applied the insights gained in two ways: first, to create and evolve support **teaching resources** based on, and informed by, the teachers' inputs, and second, to establish suitable questions for an online survey.

Presentation workshops generally involved a formal presentation on aspects of code quality, followed by practical exercises for teachers. However, project team members actively encouraged informal discussion and feedback from teachers during the practical sessions. Experienced teachers in the project team acted in an advisory capacity during preparation, to ensure that the terms used by the project team were consistent with the language used by teachers, rather than "academic speak".

For all workshops, the team used professional teaching networks to assist in the recruitment of participants.

Face-to-face conversations in the early phases were aimed at identifying key issues that teachers face when providing feedback on syntax, semantics, and style, and also gaining an understanding of how feedback related to, and was supported by, NCEA resources. Conversations during later phases involved discussion on how teachers perceived, and were supported by, the resources provided. In all cases, the face-to-face, collaborative nature of the discussions and workshops helped the project team understand issues from the teachers' perspectives.

2. Surveying teachers

In addition to gaining the deep understanding that is best achieved by face-to-face discussions, we were interested in identifying any patterns in teacher understanding and approach. For example, do teachers with more programming experience have a different approach to providing feedback than those who are new to programming? Does class size impact strategies?

To address this aspect of our investigations, and to provide us with some baseline data, we created and implemented a **survey**. The earlier discussions with teachers provided advice on how to target survey questions. The questionnaire was piloted with partner schools and then implemented as a web-based survey.

3. Secondary research

The third aspect of our approach was to explore the current status of topics that directly relate to teacher feedback when teaching computing to senior secondary students. The aims were to support our primary research by consolidating our understanding of what support is currently available for teachers and how the NCEA approach differs from other educational approaches along with possible consequences. Outputs from this research targeted double-blind, peer-reviewed **academic publications**.

How we met the research objectives

We found that the above complementary approaches enabled us to achieve a satisfactory mix of providing support, creating resources, and identifying key areas for future research.

For each research objective, we summarise below how the objective was addressed in our research. In the first column, we list the research elements that provided the data to address the objective. In the second column, we briefly overview how the element was applied and summarise how the data were analysed. The research elements are described in greater detail later, along with a summary of the main findings.

TABLE 1. **Summary of how the research objective was addressed**

What is the nature and quality of feedback delivered by existing teachers of programming?	
Workshops	A series of instructional workshops was run throughout. These comprised a series of sessions with small numbers of teachers who were unfamiliar with programming concepts and techniques. They included discussions on how teachers approached giving feedback to students and issues experienced. Two formal presentation workshops were held in Auckland in 2018 and 2019 as part of CS4HS (Computer Science for High Schools). These had 32 and 54 participants respectively and included teachers with a wide range of programming experience. Teachers were asked to informally discuss the various kinds of feedback given to students and to share experiences of success and issues when providing feedback. Insights and learnings served as inputs to the creation and evolution of teaching resources, to inform an online survey and to motivate a study into how different teaching institutions view software quality.
Survey	We implemented an online survey to elicit teachers' experiences and viewpoints on providing feedback to students. The survey contained both closed- and open-ended questions. There were 45 responses from secondary school teachers. We applied statistical analysis on the quantitative data from closed-ended questions and reflexive thematic analysis on the qualitative data from the open-ended questions (Clarke & Braun, 2006).
Study	We investigated the attitudes and approaches taken towards software quality by three systems for education prevalent in New Zealand. These were the New Zealand Ministry of Education's technology curriculum for senior secondary school students (*The New Zealand Curriculum* (*NZC*)), the Cambridge Assessment International Education (CAIE) for Computer Science, and the ACM Guidelines for Computer Science, the de facto standard for universities.

Are there differences in feedback due to teacher subject expertise?	
Survey	The survey contained questions relating to three aspects of teacher programming expertise. These were length of time teaching programming, programming background, and the number of programming languages practised. These were analysed for correlations with the data concerning aspects of feedback.

What do teachers understand about what students are struggling with, and what information do they seek, to provide better feedback?	
Workshops	This was discussed during workshops and used to both inform survey questions and to focus a study exploring available resources for use in the classroom.
Survey	The survey contained both closed- and open-ended questions relating to teachers' understanding of student issues and the kinds of support they would like.
Study	We investigated the resources currently available to support teachers of technology in New Zealand.

How do teachers use secondary sources of feedback, such as automated tools?	
Workshops	This was discussed during workshops and used to inform survey questions.
Survey	The survey contained both closed- and open-ended questions relating to teachers' use of automated tools.

How can we help teachers improve the quality and timeliness of feedback experienced by students?	
Workshops	This was discussed during workshops and used to inform survey questions.
Survey	The survey contained both closed- and open-ended questions relating to the quality and timeliness of feedback given by teachers.

Workshops

CS4HS (Computer Science for High Schools)

CS4HS is a yearly workshop-style conference held in Auckland, generally during November. The workshop sponsors include Google, Auckland University of Technology (AUT), and the Museum of Transport and Technology Auckland (MOTAT). Its aims are to support teachers of digital technologies in New Zealand by upskilling high school teachers in various technology-related topics (Auckland University of Technology, 2019).

Our aim for these workshops was two-fold. First, we wanted to elicit feedback from teachers on the resources we had prepared. Resources included a presentation describing some aspect of code style and materials to be used as the basis for practical exercises to be carried out by the teachers. Second, we wanted to explore issues experienced by teachers by encouraging informal discussions during the practical part of the workshop. This provided the face-to-face contact required for a deeper understanding of issues, as described in the previous section.

Our contributions to CS4HS in 2018 and 2019 were as follows:

- The 2018 conference was held at AUT on 21–22 November 2018. The session was attended by 32 secondary school teachers. Members of the research team (Andrew Luxton-Reilly, Ewan Tempero, and Tyne

Crow) opened the session with an overview of software quality, "Improving quality of feedback in student programs". The presenters explained what software quality is and why it is important that programmers understand and appreciate its importance (i.e., they must produce code that will minimise future program maintenance costs). Teachers then split into groups and worked through a set of worksheets with assistance and guidance from the research team. Each worksheet presented a specific aspect of code quality, explaining its role and relevance for NCEA, providing examples, and guiding teachers through a targeted practical exercise. During this part of the process, project team members discussed with teachers how they might evaluate student work and provide effective feedback to students. Presentation slides and resources were made available online for ongoing access.

- The 2019 conference was held at AUT on 14–15 November, 2019. The workshop focused on another aspect of quality (i.e., how to design and create programs that are well structured and modular). The presentation "Activities for developing programs and the logic behind them" was delivered by Tyne Crow and Andrew Luxton-Reilly. The presentation overviewed the importance of creating code that is well structured, robust, and flexible, and how these ideas relate to the criteria specified in the NCEA achievement standards. Teachers then again split into groups and worked on practical exercises that focused on planning and developing the logic required for coding to realise these criteria. Research team members (Andrew Luxton-Reilly, Ewan Tempero, Tyne Crow, and Diana Kirk) supported the activities. Again, team members provided suggestions to teachers about effective ways of providing feedback to students. Presentation slides and practice resources were made available online for ongoing access.

Both workshops attracted a mix of inexperienced and experienced programming teachers. Collaboration and informal discussion were encouraged during the practice sessions, to help us understand issues experienced by individuals. For both workshops, feedback from attendees was extremely positive. The less experienced teachers were pleased to have some concrete examples along with an opportunity to practise under the guidance of the research team. The more experienced teachers were keen to discuss the finer points relating to aspects of quality relevant for NCEA, such as robustness and flexibility, and how the importance of the topic can be explained to students.

A common thread from the workshops related to an uncertainty in how to interpret the NCEA criteria in a concrete way. This affects the *nature and quality of feedback delivered*, as the lack of clarity in the NCEA expectations left teachers unsure about the level at which the various programming concepts and techniques should be introduced. A second thread concerned *what students are struggling with*, specifically the difficulty students have with the aspects of code relating to code quality. It was felt by some that the introduction of quality aspects should be left until the higher levels. However, others believed it was important to focus on quality early on. A final thread concerned the significant variability in student capabilities and motivation. This variability, along with the extensive variation in teacher experience, caused us to realise that if we are to *help teachers improve the quality and timeliness of feedback*, we must first understand this variability in greater depth.

Workshop slides and materials have been made available as resources for teachers. The resources developed for the 2018 conference were used as the basis of a series of instructional workshops in the Auckland area (see next section).

Workshop resources

A prototype resource was developed based on the worksheets developed for the 2018 CS4HS conference (see above). The resource is a series of worksheets, each of which introduces a single aspect of code quality. Each quality topic "leads on" from the previous topic, and includes an explanation of what the topic is, why it is important, and some guided practical exercises. A separate checklist on syntactic, semantic, and style elements allows the teacher to "test" his or her knowledge. The aims were to deepen teachers' knowledge and understanding of code quality, thus helping them become more confident when providing feedback to students.

Rather than simply providing a set of worked examples, a key aspect of the worksheet resources is that each worksheet builds on the previous one in a way that we believe supports learning about code quality in an intuitive way. We hoped this approach would result in a resource that was suitable for both teaching teachers

and for teachers to use in the classroom as teaching resources. Feedback from teachers indicates that this targeted approach was successful.

Instructional workshops

The resource was used as the basis of a series of workshops in the Auckland area run by Margot Phillipps. These workshops were piloted in 2018 and run throughout 2019. The aims were to introduce inexperienced teachers to programming techniques, with a focus on quality-related techniques, to understand issues experienced in the classroom, and to trial and refine our teaching resources. Each series of workshops involved approximately 12 sessions with the same participants, often a single teacher. During the pilot, the prototype resource was refined according to teacher feedback.

The informal discussions during workshops addressed several of the study objectives (see Table 1 in previous section). An observation from the workshops was that the participants varied significantly in motivation and level of engagement and this prompted the use of different approaches to support learning. For example, it was found that, for highly motivated and engaged participants, showing worked examples that were similar to the problem to be solved was an effective strategy.

Survey—understanding teacher feedback on programming style

A survey was conducted to understand teachers' viewpoints on syntax, semantics, and style and how they provided feedback to students to help them improve their code. The survey addressed our interest in identifying any patterns in teacher understanding and approach and provided us with baseline data (see previous section). Survey questions were informed by earlier workshop discussions involving programming teachers. The questionnaire was piloted with partner schools before being distributed nationally as a web-based survey.

The survey had a series of multiple choice and open-ended questions. For teacher demographics, we asked how long the teacher had taught programming, how many years they had actively practised programming, their programming background (e.g., practical experience outside the classroom and any computing qualifications held), and the languages and tools with which they were familiar. For classroom demographics, we asked about class sizes and levels, or mix of levels, taught.

For each of these categories, we asked about the teacher's perspectives on how successful they felt they were in providing feedback and how useful they believed their feedback was. We also asked in what way they would like to improve their feedback and what kind of support they would find most helpful. For these questions, we provided multiple choice options and also included the opportunity to comment and expand on choices made. There were approximately 45 responses containing usable data.

We applied chi-squared tests to analyse the quantitative data from closed-ended questions. For contingency tables with low cell values, we amalgamated some table cells to form a 2 x 2 matrix and analysed using Fisher's Exact Test. We applied reflexive thematic analysis to analyse the qualitative data from the open-ended questions (Clarke & Braun, 2006).

Respondents represented a large spread of background and experience. More than half had been teaching programming for more than 4 years and three had not yet commenced teaching at the time of completing the survey. Fourteen of the respondents had industry experience relating to programming. Of those who did not report having industry experience, eight reported to have written a program of more than 1,000 lines, 15 had only programmed a little for reasons other than teaching, and eight had only used programming in relation to their teaching practice. Python was the most common language used, but many respondents reported being familiar with other languages. Class sizes varied, with 21–30 students being the most common.

Themes—feedback by inexperienced teachers

Quantitative analysis indicated that inexperienced teachers spent more time delivering feedback on syntax than on semantics and style. There was a correlation (95% confidence) between respondents with less than 4 years' programming experience spending proportionally more time giving feedback on syntax. Survey questions relating to supporting tools and resources included those to help teachers improve their own expertise, those to help teachers provide formative feedback to students, and those to help with marking student submissions. All categories of feedback were considered useful, with no major trend in favouring resources for any category of feedback.

Themes—feedback on style

One theme that emerged during qualitative analysis was that teachers spent relatively less time giving feedback relating to style. Similar time is spent on feedback for syntax and semantics, with a mode in the 31%–40% range, but significantly less is dedicated to style, with a mode of 11%–20%. Only seven respondents spent more than 20% of time feeding back on style. Teachers found giving feedback on style generally problematic. Many students produced running programs that were badly written with poor structure and the students were unwilling or unable to understand why this was not acceptable. Students tended to ask for more help with syntactic and semantic issues relating to functionality and that feedback on style had to be more intentionally given to students:

> *Style often gets lost here because lots of programs will run with bad style errors. Students ask about this less, so I need to be diligent in observing what they are doing.*
>
> *Students don't all want style feedback. They just get it anyway.*
>
> *Style is the hardest to instruct students on and has no effect on the running of the program. Students don't understand how important it is to style for easier maintenance and readability.*

Themes—programming environment

Teachers used a broad variety of programming environments and languages, and many reported that the choices made influenced their effectiveness in providing different aspects of feedback:

> *The Brackets code editor has good prompting which has hugely cut down on the syntax errors.*
>
> *In Python we use Pycharm and follow the style guide in terms of spacing etc.*

Themes—time

Qualitative analysis indicated that the majority of teachers felt there was not enough time to provide the kind of feedback they would like to provide. Less experienced teachers lacked the time to upskill to an appropriate level. This was linked to class size, time available for the course, and students working on independent projects:

> *Large classes and lack of class time sometimes means that I am unable to give timely feedback to all students who would like it.*
>
> *With 30 students all working on their own project the debugging becomes the biggest issue…*

Findings

Analysis of the quantitative and qualitative data caused us to understand that the variation in classroom context was much more extensive than we had anticipated. Teaching experience varied from "just started" to very experienced. Teachers' programming background ranged between "none—just learning as I go" to "several years' experience in the IT industry along with a formal qualification". There were variations in class size and homogeneity of levels taught (some were teaching two NCEA levels at the same time). Approaches to teaching ranged from a focus on empowering students to self-learn to providing students with explicit instruction and guidance. When asked about the kinds of resources wanted, some preferred tools and resources that would support personal learning while others wanted tools and resources that students could use independently.

Some classes implemented project-based learning and encouraged the students to select a project of their choice, while others provided pre-defined projects, either created by the teacher or sourced from the NCEA support materials. We illustrate the variation in teaching experience with some quotes from the survey:

> *It takes time to teach correct syntax.*
>
> *Once they get the basics, and also how to look up syntax for themselves, most students can figure out a lot of syntax problems on their own.*
>
> *Semantics can be the hardest to grasp because it is how to think as a coder.*
>
> *What is Python semantics?*
>
> *Experiences play a big role here. I can pick up semantic errors faster than I used to be able to.*
>
> *I don't know enough yet to be able to help my students with style.*
>
> *Mostly I have used the programming language and written many programs before trying to teach the programming language.*
>
> *Style is more about teaching good practice. It depends on how much a student cares about it.*

A second finding was that, although teachers tended to state they were confident when providing feedback, the open questions told a different story. There was a notable lack of confidence among teachers with little programming experience outside the classroom. Many desperately wanted help in the way of explicit examples, guided materials, and access to university-level mentors:

> *Some of the students far outstrip my experience and knowledge. I often have the class guru whom we all turn to for the answers.*
>
> *What I am not confident [about] is whether my feedback follows best practice in the industry since*
>
> *I do not have any programming experiences in the industry.*
>
> *I do not understand programming enough to provide useful feedback.*
>
> *I rely on students working it out for themselves.*

These themes were not unexpected. However, the characteristics of contextual variation were more varied than expected. Such variation is not found in the tertiary education environment and so this represents a significant departure from what is known. Future research must take this into account. Furthermore, the issue of lack of confidence among teachers was noted early on (Bell et al., 2014) and this does not appear to have diminished over the years.

The outcomes of the study are now ready as a research output for submission to a suitable CS education target.

Study—technology support

As the introduction of Digital Technologies into the New Zealand secondary school curriculum is relatively recent, a study was conducted to explore the available resources and find out what support was available for teachers. A major consideration is the degree of flexibility in the curriculum that allows teachers to develop unique implementations of the curriculum. This means that different kinds of resource may be required to support the variety of implementations.

The need to provide quality support has been commonly outlined as one of the challenges facing countries implementing computer science in school curricula. Key issues relate to the insufficient number of teachers with appropriate knowledge and few computer science graduates choosing teaching (Webb et al., 2017). An early survey showed that only 45% of teachers of digital technologies at NCEA senior levels had even basic programming skills and only 64% felt confident about teaching computer science (Bell et al., 2014). The study found that there was a large range of resources available. These included the official resources made available by the Ministry of Education and a disparate variety of materials developed by various individuals, organisations, and groups.

Official resources

The primary resource available as an official Ministry of Education resource is *Te Kete Ipurangi* (*TKI*) (Ministry of Education, 2018b). Following the links within the Technology learning area leads to a section containing specific exemplars and snapshots for the different progress outcomes, along with information about professional support, case studies of individual schools, recommended links, and online webinars (Ministry of Education, 2018a). The exemplars are a fundamental part of the curriculum implementation. A second official resource is found on the New Zealand Qualifications Authority (NZQA) website,[1] where teachers can find the assessment standards along with success criteria, explanatory notes, clarifications, annotated exemplars, and links to key information on *TKI*. The Ministry of Education has also made available a Digital Fluency Package, aimed at helping teachers to personally upskill in technology. The focus is on helping teachers understand the "big picture" and introduces the fundamental ideas and motivations for the curriculum.

Professional support networks

In addition to the official resources, the study found several professional support networks for technology teachers. The main association for digital technology teachers is Digital Technology Teachers Aotearoa (DTTA).[2] Teachers may share ideas, resources, and issues through a discussion group, website, and national meet-ups and DTTA helps teachers run events. Technology Education New Zealand (TENZ)[3] provides support across all technology areas. The support is particularly useful for teachers who teach in other technology areas but are expected to teach one or two digital technology classes. Digital Future Aotearoa is a volunteer-led organisation that organises coding clubs for teachers and students. Their website[4] provides support for finding and hosting coding events.

Other tools and resources

One main resource for technology teachers is CS Unplugged,[5] developed as an outreach programme by Professor Tim Bell and the University of Canterbury. It is a widely used collection of non-digital exercises aimed at helping teachers become familiar with the computer science (CS) fundamentals by introducing core CS concepts at an abstract level. Closely related to this is the *Computer Science Field Guide*,[6] developed at the University of Canterbury as an open source, interactive site with a role similar to a student textbook. The ICT Grad School[7] and the Unitec Mind Lab[8] both offer postgraduate courses for technology educators. Code Avengers[9] is a New Zealand-based learning platform for programming.

Findings

The study found that there is a wide range of resources available, but these are fragmented and difficult to navigate. Official resources are not standardised and tend to focus on support for building teachers' understanding of the curriculum and for helping them plan a compliant implementation, rather than helping teachers upskill in technology. This does create a significant amount of work for teachers although potentially more beneficial in the long run than providing ready-made resource and content. However, it can also be very challenging for teachers who are new to programming and are trying to come to terms with both content and how to integrate this into the wider programme.

1 https://www.nzqa.govt.nz/
2 http://nzacditt.org.nz/
3 https://tenz.org.nz
4 https://www.digitalfutureaotearoa.nz/
5 https://www.csunplugged.org/en/
6 https://csfieldguide.org.nz/en/
7 https://signal.ac.nz/
8 https://www.unitec.ac.nz/about-us/contact-us/schools/mindlab
9 https://www.codeavengers.com

Study—approaches to code quality

Our goal was to understand the differences in the importance given to code quality between universities and NCEA. *Code quality* is important because most of the lifetime cost of a software product is incurred after it is first delivered (i.e., during the maintenance phase), and the quality of the code significantly impacts these costs (Schach, 1999; Sommerville, 2011). This suggests that "code quality" should be a key competency of graduates of computing qualifications. The study was motivated by the fact that aspects of code quality (e.g., adhering to naming conventions) are explicitly referenced in the NCEA standards, and this means that teachers must spend considerable time providing quality-related feedback to students who choose programming as a topic. We wanted to understand how existing qualifications address students' understanding of, and ability to create, code of good quality and to know if, and how, the feedback expectations and mechanisms differed between systems. If the NCEA framework is lacking in some major aspects, students wishing to continue to tertiary education will not be suitably prepared and will thus be at a disadvantage. We wanted to know if this was the case.

In this study, we compared three approaches to computing education. We investigated the end-to-end processes based on the ACM Guidelines for Computer Science (ACM, 2013), the New Zealand Curriculum for Technology (Ministry of Education, 2018a), and the Cambridge Assessment International Education for Computer Science (Cambridge Assessment International Education (CAIE), 2017). The ACM is a professional society based in the US. It was established with the aim of inspiring dialogue among computing educators, researchers, and professionals about challenges in the field of computing. ACM has created, along with IEEE (another US-based organisation), a set of standards and guidelines for computing curricula. The ACM/IEEE Guidelines are generally accepted as a de facto standard for universities, *NZC* forms the basis of education in New Zealand schools, and the CAIE is an internationally recognised assessment system. For each of three systems, we investigated the underlying philosophy behind the system, how the system was implemented, and the mechanisms applied.

The ACM/IEEE Guidelines were created by a task force comprising "computing educators, researchers, and professionals". The Guidelines for Computer Science are structured as a Body of Knowledge, organised into categories and subcategories. They include a small number of references to quality aspects, but these are general and do not mention specific techniques. To see how these were applied by universities, we sourced a set of Learning Outcomes (LOs) for 141 introductory CS courses (Becker & Fitzpatrick, 2019) and found that fewer than 30% of the LOs were specific to software quality. This means that we cannot be confident that students of first-year CS courses are exposed to concepts relating to software quality. In addition, the typical approach to assessment at university is one of *aggregation*—marks and part marks are totalled and a grade applied—and this means that, even if exposed to quality concepts, a student might pass without answering any quality-related questions.

For secondary schools in New Zealand following *NZC*, the underlying philosophy is one of student-centred learning. Rather than "learning a body of knowledge", the aim is to create creative, resilient, confident, life-long learners. The development of the curriculum for technology was supported by advisory groups, including teachers, education leaders, and subject experts, and included consultation with Māori. The curriculum for *digital technology* includes several references to quality topics. Assessment is by means of a set of achievement standards. We found that the standards for computer programming contained some very specific references to quality techniques, and these increased with the achievement level. To pass, a student must pass all elements (i.e., there is no concept of aggregation). We can thus be confident that a student who has achieved a standard at a specific level has some knowledge of the quality concepts expected at that level.

The philosophy underlying the CAIE system is similar to that of the ACM (i.e., it comprises a Body of Knowledge to be learned by the students. In this case, syllabus setting, assessment setting, and marking are carried out by the CAIE. The LOs for the syllabus contain some specific quality techniques. However, when checking examination marking schedules for the past 5 years, we found no evidence for marks being awarded specifically for any of the quality-specific LOs. In addition, as for universities, marking is by aggregation.

The key findings for the study were that there was a significant variation is the perceived importance of introducing beginner programmers to the notions and rationales relating to code quality. For the university and CAIE systems, the focus is on mastering a "knowledge base" (i.e., knowing "how" as opposed to "why"). In both cases, code quality appears to be regarded as not too important in the early stages. However, as course setting is carried out by university lecturers, we found variation in the university LOs, with 11 out of 45 including a quality element. A small number of these specified the rationale underpinning the element. In comparison, *NZC*'s focus on developing rounded individuals appears to encourage a more questioning attitude, with some quality-specific terms appearing in the standards. For example, the *Refined* level requires a program to be "Flexible" and "Robust". There appears to be no mention of "Maintainability" aspects. The CAIE approach is more holistic, in that one body sets the syllabus, assessments, and marking schemes. However, as for universities, marking is by aggregation, and we found no evidence of marks being specifically awarded for questions on quality techniques.

We believe this is a topic that requires further investigation. Some questions to be addressed are:

- Is the minimal attention to code quality in the ACM and CAIE systems the result of deep consideration, or is it an unconsidered consequence of a "Body of Knowledge"-based system?
- Does an expectation to master quality techniques without understanding *why* impact a student's motivation to implement?
- Is there a difference in appreciation of code quality between students taught in a knowledge-based system and those taught in a student-centred system?

Findings and future work

Our goal for this study was to build understanding of how teachers and students use feedback on syntax, semantics, and style to improve programming skills. To meet this goal, we formulated some objectives, each of which addressed a different aspect of feedback. We explored these objectives by means of informal discussions during workshops, an online survey, and literature studies to ascertain the current status of relevant topics and issues. We also created resources for helping teachers upskill in the techniques for creating quality software, and evaluated and refined these during workshops.

We found that the *nature and quality of feedback delivered by teachers* varied significantly. Some teachers were confident in their understanding of the techniques and rationale for software quality while others were struggling with the basic concepts and struggled to share these with students. Student motivation and level of engagement also affected how feedback was given. We exposed *differences in feedback due to teacher subject expertise* in that more experienced teachers tended to view issues with *syntax* and *semantics* as straightforward and were able to focus on program *style*, whereas less experienced teachers spent more time helping students with *syntax* problems. Teachers found that s*tudents were struggling* in different ways, but a common theme was that of semantics (i.e., trying to produce a program that ran as expected). The *information available for supporting teachers* is extensive but mainly focused at the level of the curriculum rather than content, meaning relevant information was difficult to find. For the *teachers' use of automated tools*, we found little evidence of such use. There was strong feedback on the need for explicit support materials to *help teachers improve the quality and timeliness of feedback*. Participants in workshops were extremely positive about the resources we provided, and survey responses indicated a desire for resources for all of upskilling, helping students self-learn, and supporting assessment marking.

A key finding from this research is that many teachers in secondary schools are struggling with teaching text-based programming to senior students. This issue was identified in earlier research (Faulkner et al., 2014; Haden et al., 2016; Thompson & Bell, 2013; Thompson et al., 2013). Workshops and survey responses from our research indicate that this continues to be a problem. In some cases, the main factor is a lack of teacher experience and confidence. However, there were indications that a second factor is the lack of clarity in the meaning of some terms used in the standards. For example, "Robust" and "Flexible" are not consistently understood and applied (i.e., there is an unacknowledged uncertainty within the teaching community).

This represents a finding that requires urgent investigation. There are many resources available to teachers (Crow et al., 2019). However, these have been created in an ad hoc way by different individuals and groups. It is possible that the resources are too difficult to find and use. Our research showed us that the majority of teachers are "time poor" and may simply not have the time to explore resources. Another possibility is that the available resources must be adapted according to the context in which they will be used and, again, teachers have neither the time nor the expertise to do this.

The implication, supported by our research, is that many teachers would like to have very specific guidance on aspects of programming. Attendees at the CS4HS workshops were extremely appreciative of both the information provided during the presentations and the explicit guidance on code quality provided by the worksheet resources. Responses to open survey questions indicated a lack of confidence and a need for targeted, expert guidance. Official, available resources tend to focus on support for building teachers' understanding of the curriculum and for helping them plan a compliant implementation, rather than supporting teachers learning about programming (Crow et al., 2019). The worksheet resources for code quality that resulted from this research represent an example of the kind of resource that might be of use to teachers, both for self-learning and for use as a teaching resource in the classroom.

A second finding is that the diversity of context in the classroom is much larger than we expected. This means that implicit assumptions have been made regarding the programming capability of those teaching and the learning capability of students. University lecturers teaching programming inevitably have some expertise in programming concepts and tools. Although university students vary considerably in levels of capability and motivation, the variation is not as great as might be found in the classroom. The inference is that much of the existing literature that relates to providing teaching resources for programming is not applicable in the classroom situation.

Contextual variation is a pressing topic for research. Without understanding the dimensions of variation, and how each affects the teaching of programming, we cannot know when any resources provided will be useful—it will depend upon the context. There can be no "one-size-fits-all" approach to providing resources. A teacher who is new to teaching and has no experience of programming requires entirely different kinds of support than an experienced teacher with a strong programming background. Further, even for those with similar backgrounds, classroom situations and attitudes towards preferred approaches varied and requests for support mirrored this variation.

The implication that any resources we provide must take context into account highlights some crucial areas for future investigation. Such research is of key importance if schools in New Zealand are to provide a programming education to senior students that will prepare them for tertiary education or employment in industry, as is the expectation for other academic subjects taught.

In summary, our research has reinforced the notion that teachers have an urgent need for explicit resources to help them provide effective feedback to senior programming students. The key drivers are teachers' lack of confidence, an inconsistent understanding of some of the terms used in the standards, and a significant diversity in classroom context. Available resources are fragmented and teachers do not have sufficient time to explore and/or adapt these.

The knowledge we have acquired during our research has positioned us well to progress this research by deepening our understanding of the contextual dimensions and developing context-specific resources that will provide explicit guidance for teachers. Questions to be addressed are:

- Can we provide context-specific resources for teachers? This would involve first building a framework that captures the diverse contexts in a helpful way and then designing resources appropriate for specific contexts.
- Which contextual aspects should we focus on (i.e., where is the greatest need)?

Our main finding is that there is no single set of resources that will suffice to support teachers providing feedback. The variation in context requires targeted resources, rather than general advice.

Principal investigators

Associate Professor Andrew Luxton-Reilly
a.luxton-reilly@auckland.ac.nz
Associate Professor Ewan Tempero

Research team

Associate Professor Paul Denny, Dr Nasser Giacaman, Dr Diana Kirk, Tyne Crow

Partners

Pakuranga College (contact Charlie Smith)
AGC Sunderland College (contact Margot Phillipps)

References

ACM. (2013). *Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science.* http://seniorsecondary.tki.org.nz/Technology/Digital-technologies

Auckland University of Technology. (AUT). (2019). *CS4HS: Sessions*. https://www.cs4hs.aut.ac.nz/

Becker, B. A., & Fitzpatrick, T. (2019). *What do CS1 syllabi reveal about our expectations of introductory programming students?* Paper presented at the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), Minneapolis, MN.

Bell, T., Andreae, P., & Robins, A. (2014). A case study of the introduction of computer science in New Zealand schools. *Transactions on Computing Education (TOCE)*, 10.11–10.31.

Cambridge Assessment International Education. (CAIE). (2017). Code of practice. https://www.cambridgeinternational.org/Images/416992-code-of-practice.pdf/

Clarke, V., & Braun, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology, 3*(2), 77–101.

Crow, T., Luxton-Reilly, A., Wunsche, B., & Denny, P. (2019). *Resources and support for the implementation of digital technologies in New Zealand schools*. Paper presented at the Twenty-First Australian Computing Education Conference (ACE'19), Sydney, NSW.

Faulkner, K., Vivian, R., & Faulkner, N. (2014). *The Australian digital technologies curriculum: Challenge and opportunity.* Paper presented at the Sixteenth Australasian Computing Education Conference—Volume 148 (ACE '14), Adelaide, SA.

Haden, P., Gasson, J., Wood, K., & Parsons, D. (2016). *Can you learn to teach programming in two days?.* Paper presented at the Australasian Computer Science Week Multiconference (ACSW '16). https://doi.org/10.1145/2843043.2843063

Immigration New Zealand. (n.d.). *Skilled ICT workers needed in New Zealand*. https://www.newzealandnow.govt.nz/node/416

Immigration New Zealand (2019). Long-term skills shortge list. https://skillshortages.immigration.govt.nz/assets/uploads/long-term-skill-shortage-list.pdf

Ministry of Education. (2018a). *The New Zealand Curriculum*: *Technology*. http://nzcurriculum.tki.org.nz/The-New-Zealand-Curriculum/Technology

Ministry of Education. (2018b). *TKI Te Kete Ipurangi*. https://www.tki.org.nz/

Schach, S. R. (1999). *Classical and object-oriented software engineering* (4th ed.). McGraw-Hill.

Sommerville, I. (2011). *Software engineering* (9th ed.). Pearson.

Thompson, D., & Bell, T. (2013). *Adoption of new computer science high school standards by New Zealand teachers.* Paper presented at the 8th Workshop in Primary and Secondary Computing Education (WiPSE '13), Aarhus, Denmark..

Thompson, D., Bell, T., Andreae, P., & Robins, A. (2013). *The role of teachers in implementing curriculum changes.* Paper presented at the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13), Denver, Colorado.

Webb, M., Davis, N., Bell, T., Katz, J., Reynolds, N., Chambers, D., & Syslo, M. (2017). Computer science in K-12 school curricula of the 21st century: Why, what and when? *Education and Information Technologies*, 445–468.