Can You Learn To Teach Programming in Two Days?

ANON 1
1st author's affiliation
1st line of address
2nd line of address
Telephone number
1st E-mail

ANON 2
2nd author's affiliation
1st line of address
2nd line of address
Telephone number
2nd E-mail

ANON 3
3rd author's affiliation
1st line of address
2nd line of address
Telephone number
3rd E-mail

ABSTRACT
Between 2011 and 2013, an updated set of national standards for secondary school computer science education was introduced in New Zealand. This change caused great difficulties for many existing "computing" teachers. After many years of teaching primarily word processing, they were suddenly tasked with teaching programming, even though they were themselves unable to program. In this paper we describe the structure and results of two in-service professional development workshops for these teachers. The workshop structure places emphasis not only on improving a teacher's programming skill, but on exposing him or her to validated pedagogical techniques in programming education. Preliminary results are positive, with most teachers being able to transfer the training into their own classrooms. After the workshops, teachers continue to request support, especially additional classroom-ready materials. We maintain that effective in-service training must include this ongoing support.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]

General Terms
Human Factors, Languages.

Keywords
In-service teacher training, programming education

1. INTRODUCTION
In 1974 "computing" was first introduced into the official secondary school curriculum in New Zealand. At this time, it was classified as a vocational subject, in the same category as manufacturing-oriented courses like sewing and woodwork [5]. Computing classes were usually titled "Text and Information Management" (TIM), and consisted almost exclusively of the use of productivity software such as word processors, spreadsheets and presentation tools. At this time, course credits earned in computing classes did not contribute to a student's eligibility for university entrance after high school.

In the years between 1974 and 2010, digital technologies progressed rapidly. Use of sophisticated software became ubiquitous in the sciences; advances in programming languages and development environments made software development a practical approach to complex problem-solving even for non-specialists; access to computer hardware evolved to near universality. The knowledge high school students needed to succeed in modern digital society was completely transformed, but the secondary school curriculum remained unchanged.

Conference'10, Month 1–2, 2010, City, State, Country.

DOI: 10.475/1234

To address this serious problem, academics in New Zealand, led by computer scientists at the Universities of Canterbury and Otago and in consultation with a number of high school teachers, designed a set of modern computer science standards for secondary schools. The goal was to accurately reflect the role of computing in the academic, professional and lay communities, and to increase the rigor of the computing curriculum to make it equivalent to that of mathematics and the physical and biological sciences. The standards define a set of topics and assessment specifications for each of the final three years of high school [4]. The standards became mandatory for Level 1 (the second year of high school) in 2011. Level 2 (the penultimate year of high school) was introduced the following year, and Level 3 (the final year of high school) in 2013. The computing subject area is now titled "Digital Technologies" and counts toward university eligibility.

The new standards were a drastic change from the existing ones, introducing elements such as computer programming and computer science theory, which were completely absent from the old TIM standards. While such a change was essential to ensure quality education for New Zealand young people, the actual introduction of the standards was, for many in-service teachers, alarmingly abrupt [30]. Teachers, some of whom had decades of experience in TIM, were suddenly required to teach and assess subjects such as computer programming, database design and algorithmic analysis -- complex and difficult topics in which they had no personal experience or training.

Under the oft-cited pedagogical model of Shulman [29, 9] these teachers had extensive general teaching skills (pedagogical knowledge; PK) but now had to teach a subject in which they had neither personal expertise (content knowledge; CK) nor knowledge of the specific activities, techniques and interventions that are effective in teaching it (pedagogical content knowledge; PCK). Similar problems have arisen in other countries consequent upon the introduction of updated computer science curricula [e.g. 14, 16] but were particularly acute in New Zealand, due to its small population base and high number of rural schools, where a single teacher is often responsible for delivering Digital Technologies to all high school grades.

This situation was dangerous for both teachers and students. Without qualified instruction, students could not receive the preparation that tertiary educators would now expect from their incoming high school graduates. Teachers, suddenly required to

teach subjects in which they had neither CK nor PCK, experienced stress and frustration [cf. 20]. Feedback from in-service teachers made it immediately clear that training and support was urgently required to prevent severe problems in the delivery of the new standards.

2. IN-SERVICE TEACHER TRAINING OPTIONS

A number of specific approaches were taken to support teachers after the introduction of the Digital Technologies standards in 2011. Among these were complete courses of study (lasting one or two semesters) through a tertiary institution, online self-paced courses (see, for example, www.cs.otago.ac.nz/schools/faq.php for details of an online video course in basic Python programming) and large teacher training conferences (e.g. cosc.canterbury.ac.nz/ cs4hs/2013/index.html).

Each of these approaches provides valuable educational and professional development opportunities for in-service secondary school teachers. However, each of them also has significant shortcomings, especially for teachers from the schools outside the main urban centres. Spending a year, or even a single semester doing a tertiary qualification is impractical for most high school teachers. Online courses have anecdotally proved too difficult for many of the teachers who have attempted them. Existing research on online learning of programming has highlighted the challenges associated with this delivery model [6]. Both of these approaches also tend to focus exclusively on CK, not PCK (i.e. the teacher will improve his or her own programming skills but will not learn validated educational techniques for *teaching* programming). The large teacher training conferences do explore PCK, but tend to be expensive and to require substantial travel for rural teachers.

At ANON_INST, our current focus is on providing professional development that addresses the CK and PCK needs of in-service teachers in our region (which includes many rural schools), with a delivery model that is practical and effective for them. To this end, in 2013 and 2014, faculty in the Department of Information Technology offered a number of short training workshops for high school teachers adjusting to the new Digital Technology standards. In this discussion, we consider only the workshop that focused specifically on the teaching of computer programming. This workshop was offered once in each year, with slight changes in the schedule and content (discussed below) being made between the 2013 and 2014 occurrences. It is intended for in-service secondary school teachers with little or no programming experience. In the following sections, we first present the theoretical framework underpinning our workshop design, and then describe the specific structure and content, and preliminary outcomes.

3. THEORETICAL FRAMEWORK

Our workshops are carefully designed to ensure that any content knowledge or techniques the teachers learn can be successfully transferred into their classrooms ***after the workshop is over***. Both research evidence [7] and anecdotal feedback from local teachers indicates that a major shortcoming of both single instance workshops and online training and tutorials is that the teachers can complete the workshop tasks, or work through the online tutorials, but are unable to move beyond those specific exercises sufficiently to deliver an entire programming course to their students. To avoid this problem, our approach follows these main principles (each discussed in detail below).

Our workshops:

1. Develop both CK and PCK.
2. Emphasise the fundamental principles of computer programming, not simply the use of a specific tool or language.
3. Are the beginning of an ongoing relationship between the high school teachers and the ANON_INST faculty, to provide necessary support.
4. Are pragmatically accessible to teachers from our catchment.

Our workshops develop both CK and PCK. Many people can program; few can teach programming. Several decades of computer science education has demonstrated the inherent difficulty of teaching computer programming well. This same literature has demonstrated evidence for the effectiveness (or lack thereof) of specific educational approaches and techniques. Simply helping a teacher learn to program does not acquaint him or her with these techniques [10, 25, 26]. In our workshops we demonstrate and explore specific educational practices that teachers can immediately use in their own classrooms. For example, we cover the use of Activity Diagrams [23] as a tool to support problem decomposition and computational thinking. We model the use of a simplified Pair Programming methodology, that has been used effectively with novice programmers [32]. We provide sample classroom materials for delivering course content and structuring a software development lifecycle. We also discuss purely pedagogical issues such as the value of analogy and metaphor when explaining abstract computing concepts like flow of control. Further details of each of the practices are given below.

Our workshops emphasise the fundamental principles of computer programming, not simply the use of a specific tool or language. One of the most common criticisms of computing education in schools is that it too often teaches "tool use" rather than computing theory and concepts [13, 21, 24]. The limitation of this approach is self-evident -- concentrate exclusively on the details of the tool, and all you have taught is how to use that particular tool. An understanding of core concepts is essential to enable students (and teachers) to move independently to new languages, systems, environments and computational problems.

Our workshops are the beginning of an ongoing relationship between the high school teachers and the ANON_INST faculty, to provide necessary support. No brief training experience, no matter how carefully designed and implemented, can provide a novice programmer with all the skills and knowledge necessary to design, deliver and assess a full computer programming course. As our workshop participants move to their own classrooms, they will inevitably hit snags, have questions and need help. In consequence, we offer an ongoing mentoring relationship with our teachers following the workshop.

Our workshops are pragmatically accessible to teachers from our catchment. High school teachers in our region are extremely busy, and have very limited funding for professional development. Our workshops are therefore short (three days in 2013, shortened to two days in 2014) and inexpensive (free in 2013, NZ$25 in 2014). They are scheduled on weekends and during school holidays to minimise the amount of leave teachers must take.

4. WORKSHOP STRUCTURE

4.1 Programming Environment

In our workshops, we use the Scratch programming environment [17]. We use Scratch to teach programming fundamentals to the participants, and as a tool they can take into their own classrooms to teach the New Zealand Digital Technologies standards.

Scratch is a widely-used Visual Programming Language (VPL). When using a VPL, code is not written as text. Rather, it is built as a collection of graphical tokens, each of which corresponds to one or more machine operations. The graphical tokens snap together like building blocks. In Scratch, as in most VPLs, the "snapping together" is constrained so as to prevent syntax errors.

Scratch code units are bound to image elements called "sprites". Scratch command blocks are provided that permit a variety of animation and modification operations on the sprites. Data variables of all traditional types are supported, as are standard mathematical and logical operations, and typical flow of control structures such as conditionals and both deterministic and conditional loops (i.e. for and while loops). Scratch has a large and active online community (wiki.scratch.mit.edu) and it is sufficiently rich that it can be used to implement quite complex algorithms [31]. It has been used frequently as a teaching tool [e.g. 15] and as a teacher training tool [e.g. 3]. A detailed discussion of the Scratch environment and the principles underlying its design can be found in the work of Maloney and his colleagues [17].

The most frequently claimed advantage of Scratch is that novices find it easy and enjoyable to use [e.g. 27]. Several features of Scratch ensure this ease: The block combination constraints prevent syntax errors; many complex behaviours (e.g. collision detection) are implemented as single statement primitives; the interpreted execution model provides immediate visual feedback, etc. These features mean that Scratch is likely to be a good choice both for our novice teachers and for their novice students.

However, we maintain that some of these same features make Scratch *easy to teach with*. That is, it can be easier to develop good pedagogical technique when working with Scratch than when working with a traditional text-based language. For example, a recommended pedagogical approach is to clarify abstract concepts with concrete examples [e.g. 12]. When working with a text-based language, providing a concrete example of an abstract programming concept such as "conditionals" would involve a verbal description (potentially difficult to produce) or advance preparation of some sort of demonstration program (time-consuming and inflexible). In Scratch, conditional behaviours can be explicitly and immediately demonstrated by constructing a conditional syntax block and clicking on it, producing alternative visible behaviours for the associated sprite.

In spite of these strengths, using Scratch has certain documented risks. Meerbaum-Salant and her colleagues [19] found that close analysis of children's Scratch code samples reveals a number of bad habits including "bottom-up programming" and large numbers of fragmented code modules. They argue that these problems can best be prevented by explicitly teaching good programming habits, rather than encouraging students to simply "have a play" with Scratch. This position -- that Scratch is not a toy, but a tool for real programming education -- is core to our approach, as discussed below.

A second commonly suggested shortcoming of VPLs in general is the potential difficulty students will encounter when migrating (as they inevitably must) from the VPL environment to a traditional text-based programming language. This problem extends beyond the obvious difficulties of acquiring a qualitatively different syntax and into the core semantics of programming. The logic for implementing some common functionality is very different in Scratch than in traditional languages. For example, modern Object Oriented programming languages enable communication between entities through the use of method calls. That is, an object A can communicate with an object B by calling one of B's methods, perhaps passing in data and receiving a returned result. In Scratch, this sort of communication is not possible. Sprites communicate exclusively through *broadcast*. Any sprite can broadcast a message signaling that some event has occurred. Any other sprite can define a code block to be executed in the event of that message being broadcast. All the broadcasting and responding happens in parallel. A Scratch program is therefore closer, in this respect, to a complex multi-threaded application than to the typical beginner program written in C# or Java, where individual objects communicate explicitly. Students (and teachers) who learn to build applications in Scratch might find difficulty in migrating to a traditional language and architecture. Some authors have found evidence of this problem [e.g. 11] while others have not [e.g. 2]. In face of these equivocal results, it seems sensible to make teachers aware of the potential dfficulties and be prepared to provide support for these teachers when they first migrate themselves and their students to industrial languages.

4.2 Workshop Schedule

The workshop contains three separate sections. In the original 3-day offering in 2013, each of these sections was taught over an entire day (from 9.00 am to 4.00 pm). Participant feedback indicated that three days was too long for most teachers, both because they had to arrange for classroom cover, and because those teachers coming from the rural towns needed to arrange two nights' accommodation in ANON_CITY. In the 2014 offering, therefore, the workshop was compressed to two days, with section 1 taught on the first day and sections 2 and 3 on the second day. The three sections are:

1. Essential programming concepts and how to use Scratch to teach them.
2. Complete Scratch programming projects suitable for the high school classroom.
3. Teaching children good software development processes.

Section 1: Essential programming concepts and how to use Scratch to teach them. The first section begins with a general outline of the workshop plan, and an explanation of what Scratch is, and why we have selected it for the workshop. We discuss both the strengths and weakness of Scratch in this context, and emphasise the importance of teaching programming, not teaching Scratch.

We then work through the essential principles of computer programming: problem decomposition, sequence, conditionals, loops and data. We also cover some more advanced concepts that are primitive to Scratch, such as event-driven design (e.g. responding to mouse and keyboard events) and multi-threading (the way that all sprites in a Scratch program are running their scripts in parallel).

The materials we use for this section (PowerPoint slides and programming exercises) are set at a level which would also be suitable for use with high school students. Teachers are encouraged to view this section of the workshop as providing materials they can use and a model they can follow when they return to their classrooms. This structure ensures that even non-programmers will be able to follow the presentation, and gives the teachers the immediate confidence of knowing that their introductory classroom materials are already prepared.

Each concept in turn is introduced and discussed in its abstract form; that is, without reference to a particular language or development environment. For example, the initial discussion of conditionals focuses on the need for problem solutions to be able to follow alternate paths depending on the problem state -- no time is spent worrying about semi-colons or parentheses. This approach highlights the essential difference between programming as an intellectual practice and the programming tools we use to perform that practice. After the principle has been discussed abstractly, we demonstrate how to depict the associated logic in an Activity Diagram. This introduces an effective teaching tool and again emphasises the importance of underlying concept over syntactic detail. Next we illustrate the details of the principle's implementation in Scratch, and support the participants through one or more simple programming exercises. There is evidence that presenting a single concept, followed by extensive rehearsal, produces good learning outcomes in introductory programming [18]. By using this structure in the workshop, we simultaneously build the skills of the participants and demonstrate an effective structure for their own teaching practice.

The early, very simple exercises are performed with each participant at his or her own machine. As the problem complexity increases, we introduce the simplified Pair Programming protocol [32]. By providing the classroom materials, modeling the effective presentation of programming concepts and explicitly demonstrating specific validated pedagogical techniques, we ensure that our participants are developing both CK and PCK.

This section of the workshop fills the first day, taking approximately six hours of focused work time, divided by brief tea and lunch breaks. Participants in both workshops have responded very positively to this section, but have also indicated that it requires a great deal of concentration, and can be fatiguing. We therefore end the workshop in the late afternoon and do not schedule any evening activities, to allow time for rest and reflection on the day's material.

Section 2: Complete Scratch programming projects suitable for the high school classroom. In the second section of the workshop we move from the basic principles to more pragmatic considerations of how to structure a multi-week module on introductory computer programming. Our primary goals in this section are to 1) strength the participants' developing programming skills, 2) to provide complete classroom materials and confidence in their delivery and solution and 3) to direct participants to useful teaching resources for Scratch. The second and third of these again address the concern teachers have about not being able to transfer what they learn in a workshop to their own classrooms. The discussion of how to locate quality resources is especially appreciated by teachers who have tried to "teach themselves Scratch online" and have become lost in the vast amount of user-generated content available, not all of which is pedagogically sound.

In this section, we work through two complete programming projects based on examples found in the excellent online book *Starting from Scratch*, by Jeremy Scott [28]. We recommend this book, and its companion tutor's manual, as a rich source of example projects. We find especially effective the number of extension exercises that come with each project, as they promote essential rehearsal. For our workshop, we make minor modifications to the Scott material to further explain certain points and, occasionally, to adjust the language for the New Zealand reader.

All workshop participants complete the two programming projects. The first project is done individually; the second is done using our modified Pair Programming protocol. The workshop facilitators assist the teachers as needed to ensure that all are able to produce working solutions (again, giving them the confidence necessary to use the materials in their own classrooms). This session is, without question, the part of the workshop that the participants enjoy the most. It is gratifying to see teachers who had previously been very negative about the possibility of learning to program and the responsibility of having to teach it, experience the great satisfaction that comes from successfully constructing a dynamic, interactive digital artefact.

Section 3: Teaching children good software development practices. In the final section of the workshop we discuss how to support a complete software development project. The programming industry has a number of development protocols designed to ensure smooth progress from initial concept, through logical and physical design, to implementation, deployment and testing. Such approaches are not, however, commonly included in introductory programming courses. However, for the secondary school teacher, we believe incorporating a formal software development life cycle has a number of benefits. First, it helps to build good software practices from the very beginning of a student's education, avoiding problems such as those identified by Meerbaum-Salant and her colleagues [19]. Second, it greatly extends the educational opportunities of programming education. For example, producing required documentation allows students to practice literacy skills, and writing pseudocode encourages problem decomposition and computational thinking. Third, the new Digital Technology Standards explicitly divide computer programming into planning and implementation steps[1], each of which are marked separately, and for each of which the student can earn university entrance credits. By demonstrating an age-appropriate software development life cycle (SDLC) to our workshop participants, we provide a valuable structure for potentially several weeks of class time.

This third section of the workshop[2] begins with a brief discussion of the role of a formal SDLC, the general structure used in industrial software development, and the purpose of each of the steps (requirements analysis, logical design, physical design, etc.) We discuss very briefly terms like "agile", "test-driven development" and "extreme programming", so that the teachers will be comfortable with these concepts if they encounter them in their independent reading. We then present a simplified SDLC, which we have designed specifically to be suitable for beginner or younger programmers, and for small interactive applications of the type most often created with Scratch. We eschew terms such as "logical analysis" in favour of more vernacular language, and

[1] Standards 91075 and 91076. Details at www.nzqa.govt.nz/ncea/assessment.

[2] Complete PowerPoint and materials available from the authors.

eliminate irrelevant steps such as maintenance. We discuss each step and provide worksheets that can be given to students to organise and support their progress through the development.

Our simplified SDLC is:

Table 1. Simplified SDLC

Step	Summary
Program idea	The student should explain succinctly but thoroughly what he or she wants to create.
How will it work?	The student answers two questions: • What does the program do? • What does the user do?
Design	Storyboarding and pseudocoding.
Implementation	Use Scratch to build the application.
Testing	Construction of a test set, and live user-testing

After discussing the entire process, we distribute a set of supporting worksheets that have been partially completed, as though by a typical high school student, for an interactive maze game. The workshop participants take on the role of this student and complete the worksheets and, time permitting, implement the game. This exercise allows us to clarify any confusion teachers may have about terminology, level of detail, etc. It gives them supported experience using the worksheets and a complete, if somewhat abbreviated experience of the entire proposed SDLC. After this session, our expectation is that they will be able to take the entire process directly into their own classrooms.

5. PRELIMINARY RESULTS

The 2013 offering of the Scratch workshop was attended by six teachers; the 2014 workshop was attended by 8 teachers. Participants included teachers from local high schools (i.e. those in ANON_CITY) and those from the rural centres up to 200 kms distant. Classroom situations for the participants ranged from a large all-girl high school with classes of 30 students and a well-equipped computer lab, to a small rural school with typically only one or two senior students studying computing.

While the need for in-service teacher support is generally agreed, it can be difficult to measure accurately the efficacy of such support. A common approach in the literature is to perform a brief survey at the end of a workshop or training activity, asking participants to self-report their satisfaction, learning, etc. using Likert scales [e.g. 1, 7]. It is generally acknowledged, even by the authors of these studies, that such data have limited statistical validity, and do not speak at all to the extent to which participating in the activity has equipped to teacher to produce competent programmers. We therefore chose to omit this survey process, deeming it to be somewhat intrusive and of limited value.

To truly measure the efficacy of our approach we would like to gather two kinds of information:

- The extent to which teachers who, having taken our Scratch workshop, are subsequently able to introduce the material into their own classrooms.
- The performance of the students of these teachers on internal and external assessments.

A complete picture of the value of the in-service training can only be obtained from both these data sources. For example, if our technique allows teachers to confidently introduce Scratch into their classrooms, but the students are unable to perform well on their assessments, we cannot claim that the workshops are effective. As discussed below, obtaining such data accurately is problematic and we have, in fact, not yet been able to do so. However, we maintain it as a long-term goal of the research programme.

5.1 Teacher Feedback

To determine the ability of teachers to transfer the contents of the workshop to the classroom, we made contact with all 14 participants of both workshops via email, asking the following questions:

1. Do you have any plans to use Scratch in the classroom this year?
2. If so, tell us how you will use it.
3. If not, any thoughts why not?
4. Now that you've had some time to reflect, do you have any comments or suggestions about the workshops?
5. Do you need any follow-up support?
6. The Digital Technologies curriculum is coming up for review this year. Is there any feedback you would give to the Ministry of Education?

We received responses from nine teachers, six from the 2014 cohort and three from the 2013 cohort. All respondents were either using, or planned to use Scratch in their own classrooms in the current semester[3].

All respondents intended to use Scratch with the first year of the formal achievement standards (the second year of high school), and with one or more earlier years. These younger students do not follow a national curriculum and are not yet earning credits toward university eligibility. Thus the teachers were introducing the new material into their courses not to provide formal credits, but simply for its educational value. None of the teachers intended to teach the Digital Technologies standards for more senior students, indicating, presumably, that they do not yet feel confident about teaching programming at that more advanced level.

Several participants mentioned the specific aspects of the workshops which they had found most useful. Interestingly, the majority of these comments did not refer to the teachers' own improved programming skill (CK), but to the resources, materials and techniques which were included in the workshop, especially those that can be used when teaching languages other than Scratch (PCK). The activity diagrams, SDLC planning tool and Pair Programming technique were all identified as being valuable, as was the focus on Scratch as a means to teaching programming, not an end in itself.

Three respondents indicated the need for additional support, specifically requesting more classroom exercises and resources. In addition, two participants (both from 2014) have independently made contact with us, asking for help with materials. One teacher requested a template for structuring an assessment, the other asked for a model solution to an intended classroom project. Other

[3] For many of the 2014 cohort, the scheduled time for Digital Technologies is late in the February-December academic year.

professional development programmes for in-service teachers have observed a similar need to support transfer of knowledge from the workshop to the classroom. For example Ahamed and his colleagues [1] describe a workshop for STEM teachers interested in adding computing (and hence computational thinking) to their curricula. While self-report responses to the workshops were positive, the authors note that teachers' attempts to transfer the workshop materials into live high school classes were not entirely successful. In another recent study [7], researchers attempted to observe this transfer directly by having participants develop lesson plans during the workshop, and assessing their quality. According to the rubric they developed, the standard of the lesson plans was, on average, only moderate, and the authors state that "the quality of lesson plans developed leaves ample room for improvement in next year's workshop."

The construction of additional resources is the focus of the next phase of our programme of in-service teacher professional development. Given the pragmatic complexity involved in both delivering and attending workshops, alternative support is obviously needed, and the provision of high-quality classroom materials specific to the NZ secondary school curriculum seems to be the most efficient, as well as the most requested option.

5.2 Student Performance

It has not yet been possible to explore formally the performance of the students of our participants. There are two potential avenues for obtaining such data. First, one could observe actual assessment results for the new achievement standards. Unfortunately, these data are not publically available. Even if they were, they would be of limited use to us at this time, since the majority of the participants have, as mentioned above, introduced Scratch in their younger classes, and these students have yet to sit any nationalised assessments. Second, one could measure directly student programming ability after a period of instruction provided by the workshop participants. Unfortunately, this approach would suffer from the widely recognised lack of validated measures of programming skill, particularly at the novice level [e.g. 8]. Therefore, the only direct evidence of student performance we have currently are the small number of student Scratch projects that have been sent to us by the teachers. Although we have been extremely encouraged by the quality of this work, we are reluctant to make generalisations from examples selected by the teachers for our inspection and feedback, as these are likely to be the most complex available. Obtaining a more accurate picture of classroom outcomes is an intended focus of future research.

6. DISCUSSION

As secondary school standards and curricula change to reflect the modern role of computing, many in-service teachers are placed in the unenviable position of having to teach materials and skills with which they are unfamiliar. Tertiary institutions who aim to provide support to these teachers must be sensitive to their pragmatic concerns. In-service training needs to be convenient and directly transferable back to the classroom. Special events like conferences and workshops should be seen as only the first step in an ongoing supportive relationship. In this paper we describe an approach that attempts to fulfill these goals. Preliminary results indicate that our approach is effective in that it has enabled the majority of participants to introduce computer programming into their own classrooms at the novice level. However, these results also highlight clearly the teachers' ongoing requirements, especially for high-quality classroom materials.

7. REFERENCES

[1] Ahamed, S.I., Brylow, D., Ge, R., Madiraju, P., Merrill, S. J., and Struble, C. A., Early, J.P., 2010. Computational Thinking for the Sciences. In SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education. 42-46. DOI=10.1145/1734263.1734277

[2] Armoni, M., Meerbaum-Salant , O., and Ben-Ari , M. 2015. From Scratch to "Real" Programming. ACM Transactions on Computing Education, 14,4,25. DOI= 10.1145/2677087.

[3] Bell, S., Frey, T., and Vasserman, E. 2014. Spreading the Word: Introducing Pre-Service Teachers to programming in the K-12 Classroom. In SIGCSE' 14 Proceedings of the 45th ACM technical symposium on Computer science education. 187-192. DOI= 10.1145/2538862.2538963.

[4] Bell, T. 2014. Establishing a Nationwide CS Curriculum in New Zealand High Schools. Communications of the ACM, 57, 2, 28-30. DOI= 10.1145/2556937.

[5] Bell, T., Andreae, P., and Lambert, L. 2010. Computer Science in New Zealand High Schools. In ACE 2010. Proceedings of the 12th Australasian Computing Education Conference, 103, 15-22.

[6] Benda, K., Bruckmand, A., and Guzdial, M. 2012. When Life and Learning Do Not Fit: Challenges of Workload and Communication in Introductory Computer Science Online. ACM Transactions on Computing Education, 12, 4, Article 15. DOI= 10.1145/2382564.2382567.

[7] Bort, H., and Brylow, D. 2013. CS4 Impact: Measuring Computational Thinking Concepts Present in CS4HS Participant Lesson Plans. In SIGCSE '13 Proceedings of the 44th ACM technical symposium on Computer science education. 427-432. DOI= 10.1145/2445196.2445323.

[8] Byckling, P., and Sajaniemi, J. 2006. A role-based analysis model for the evaluation of novices' programming knowledge development. In ICER '06: Proceedings of the 2006 international workshop on Computing education research, 85-96. DOI= 10.1145/1151588.1151602.

[9] Cochran, K. F. 1993. Pedagogical Content Knowing: An Integrative Model for Teacher Preparation. Journal of Teacher Education, 44, 4, 263-272. DOI=10.1177/0022487193044004004

[10] Cooper, S., Grover, S., Guzdial, M., and Simon, B. 2014. A future for computing education research. Communications of the ACM, 57, 11, 34-36. DOI= 10.1145/2668899.

[11] Duncan, C., Bell, T., and Tanimoto, S. 2014. Should Your 8-year-old Learn Coding. In WiPSCE'14, Proceedings of the 9th Workshop in Primary and Secondary Computing Education, 60-69. DOI= 10.1145/2670757.2670774.

[12] Felder, R. M., Woods, D. R., Stice, J. E., & Rugarcia (2000). The future of education II. Teaching methods that work. Chem. Engr. Education, 34,1, 26-30.

[13] Gutierrez, J. M., and Sanders, I. D. 2009. Computer Science Education in Peru: A New Kind of Monster? ACM SIGSCE Bulletin. 41, 2 86-89. DOI= 10.1145/1595453.1595481.

[14] Hubbard, A., and Kao, Y. 2014. Industry Partnerships to Support Computer Science High School Teacher's Pedagogical Content Knowledge. In SIGITE '14: Proceedings of the 15th Annual Conference on Information

technology education, 89-90. DOI= 10.1145/2656450.2656481.

[15] Kaučič, B., and Asič, T. 2011. Improving Introductory Programming with Scratch. In MIPRO Proceedings of the 34th International Convention on Information and Communication Technology, Electronics and Microelectronics, 1095 - 1100.

[16] Liberman, N., Kolikant, Y B-D., and Beeri, C. 2009. In-service teachers learning of a new paradigm: a case study. In ICER '09 Proceedings of the fifth international workshop on computing education, 43-50. DOI= 10.1145/1584322.1584329.

[17] Maloney, J., Resnick, M., Rusk, N., Silverman, and B., Eastmond, E. 2010. The Scratch Programming Language and Environment. ACM Transactions of Computing Education, 10, 4, (November 2010). DOI= 10.1145/1868358.1868363.

[18] Meerbaum-Salant , O., Armoni, M., and Ben-Ari , M. 2010. Learning computer science concepts with Scratch. In ICER '10 Proceedings of the Sixth international workshop on Computing education research. 69-76. DOI= 10.1145/1839594.1839607.

[19] Meerbaum-Salant , O., Armoni, M., and Ben-Ari , M. 2011. Habits of Programming in Scratch. In ITiCSE '11: Proceedings of the 16th annual conference on Innovation and technology in computer science education. 168-172. DOI= 10.1145/1999747.1999796.

[20] Ni, L., and Guzdial, M. 2012. Who AM I?: understanding high school computer science teachers' professional identity. In SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education, 499-504. DOI= 10.1145/2157136.2157283.

[21] Opel, S., and Brinda, T. 2013. Arguments for Contextual Teaching with Learning Fields in Vocational IT Schools – Results of an Interview Study among IT and CS Training Companies. WiPSCE '13: Proceedings of the 8th Workshop in Primary and Secondary Computing Education, 122-131. DOI= 10.1145/2532748.2532749.

[22] Opel, S., Höpfl, A., and Brinda, T. 2013. Practical Implementation of Learning Fields in Vocational IT/CS Education – A Guideline on Designing Learning Situations. WiPSCE '13: Proceedings of the 8th Workshop in Primary and Secondary Computing Education, 132-135. DOI= 10.1145/2532748.2532760.

[23] Parsons, D., Wood, K. and Haden, P. 2015. What Are We Doing When We Assess Programming?. In ACE '15: Proceedings of the Seventeenth Australasian Computing Education. 160, 119-127.

[24] Ragonis, N. 2012. Integrating the Teaching of Algorithmic Patterns into Computer Science Teacher Preparation Programs. ITiCSE '12: Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. 339-344. DOI= 10.1145/2325296.2325375.

[25] Ragonis, N., and Hazzan, O. 2008. Tutoring Model for promoting teaching skills of computer science prospective teachers. ITiCSE '08: Proceedings of the 13th ACM annual conference on Innovation and technology in computer science education. 276-280. DOI= 10.1145/1597849.1384345.

[26] Ragonis, N., Hazzan, O., and Gal-Ezer, J. 2010. A Survey of Computer Science Teacher Preparation Programs in Israel Tells Us: Computer Science Deserves a Designated High School Teacher Preparation! In SIGCSE '10: Proceedings of the 41st ACM technical symposium on Computer science education. 401-405. DOI= 10.1145/1734263.1734402.

[27] Ruf, A., Mhuling, A., and Hubwieser, P. 2014. Scratch vs. Karel – Impact on Learning Outcomes and Motivation. In WiPSCE '14: Proceedings of the 9th Workshop in Primary and Secondary Computing Education. 50-59. DOI= 10.1145/2670757.2670772.

[28] Scott, J. 2013. Starting from Scratch. Retrieved from http://www.royalsoced.org.uk/1050_AnIntroductiontoComputingScience.html

[29] Shulman, L. S. 1986. Those Who Understand: Knowledge Growth in Teaching. Educational Researcher, 15, 2, 4-14. DOI=10.3102/0013189X015002004.

[30] Thompson, D., and Bell, T. 2012. Adoption of new Computer Science high school standards by New Zealand teachers. In WiPSCE' 13. Proceedings of the 8th Workshop in Primary and Secondary Computing Education, 87-90. DOI= 10.1145/2532748.2532759.

[31] Ward B., Bell, T., Marghitu, D., and Lambert, L. 2010. Teaching computer science concepts in Scratch and Alice. Journal of Computing Sciences in Colleges. 26, 2, 173-180.

[32] Wood, K., Parsons, D., Gasson, J., and Haden, P. 2013. It's never too early: pair programming in CS1 . In ACE '13: Proceedings of the Fifteenth Australasian Computing Education, 136, 13-21.