The Irish Journal of Education, *2017*, *xlii, pp. 108-127*.

A PILOT STUDY INVESTIGATING THE INTRODUCTION OF A COMPUTER-SCIENCE COURSE AT SECOND LEVEL FOCUSING ON COMPUTATIONAL THINKING

James Lockwood and Aidan Mooney*
Department of Computer Science
Maynooth University

Computational Thinking has been described by Jeanette Wing (2006) as a skill set everybody should be eager to learn and use in daily life. Based on a significant amount of recent research on Computational Thinking and how we can teach it, the PACT (Programming + Algorithms = Computational Thinking) team at the Department of Computer Science at Maynooth University has been working with teachers with a view to incorporating the subject into their classrooms. To that end, a year-long course is currently being designed to teach students about Computational Thinking and Computer Science. This paper presents a brief background and overview of the course along with the design and results from an initial pilot study conducted in one secondary school. The results of the study indicate that the course was generally well received although performance did not improve on a measure of problem solving and was significantly lower for students with no prior programming experience. The course will be taught in more schools in the upcoming school year using feedback obtained from the pilot study. It is hoped that the ideas and content presented here will encourage and equip fellow researchers and educators to introduce Computational Thinking more widely in their contexts.

BACKGROUND

Denning (2009) suggested that Computational Thinking (CT) has been around since the 1950s as 'algorithmic thinking', referring to the use of an ordered precise set of steps to solve a problem and where appropriate to use a computer to do this task. Seymour Papert (1980) is credited with concretising CT in 1980 but it is since the contribution of Jeanette Wing (2006), who popularised the term and brought it to the international community's attention, that more and more focus has been placed on CT within education. In her seminal paper, Wing outlined how she believed that all children should be taught CT, placing it alongside reading, writing and arithmetic in terms of

*Aidan Mooney can be contacted at *aidan.mooney@mu.ie*.

importance. She further described it as representing a "universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" (Wing, 2006, p. 33).

Although academics have failed to agree on a universal definition of CT, Wing defines it as solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to Computer Science. She states that it is not programming and that it means "more than being able to program a computer. It requires thinking at multiple levels of abstraction" (Wing, 2006, p.35). In 2008, Wing posed a question to the Computer Science, learning sciences and education communities: "What are effective ways of learning (teaching) CT by (to) children?" (Wing, 2008, p. 4). This in turn raised further questions about what concepts to teach, the order in which these might be taught, and which tools should be used to teach them.

In the meantime, a lot of work has been done around the world and across all levels of education to introduce CT into schools, colleges, and after-school clubs, mainly through Computer Science or computing classes/courses. As CT is important to a computer scientist, this makes sense; however, it should be noted that being able to think computationally, which includes skills such as decomposition, abstraction, algorithmic thinking and pattern matching, can be of benefit to all disciplines. Bundy (2007) has also made this point, stating that CT concepts have been used in other disciplines and that the ability to think computationally is essential to every discipline.

A wide array of topics has been used to introduce CT to students. In addition to explicitly teaching students what CT is (Grover & Pea, 2013; Li, Hu & Wu, 2016), students may be introduced to concepts such as abstraction (Atmatzidou & Demetriadis, 2016; Shailaja & Sridaran, 2015), modelling (Caspersen & Nowack, 2013), algorithms (Atmatzidou & Demetriadis, 2016; Folk, Lee, Michalenko, Peel & Pontelli, 2015; Mooney et al., 2014), decomposition (Atmatzidou & Demetriadis, 2016) and problem solving/critical-thinking skills (Roscoe, Fearn & Posey, 2014; Shailaja & Sridaran, 2015).

Computational Thinking Courses in Schools in Ireland

In Ireland, various attempts have been made to introduce CT into schools. One of these is the PACT programme which is a partnership between researchers in the Department of Computer Science at Maynooth University and teachers at selected primary and post-primary schools around Ireland. Its objective was to introduce students and teachers to Computer Science through

programming and algorithms, with the aim of improving CT skills in participating students. Now in its fourth year, the programme has been delivered in over 60 schools and to over 1000 students. The PACT team provides teachers with training and resources to enable them to deliver the content to students. To date most of the teacher-level instruction involves a short course in Python, an introduction to algorithms, and the use of problems provided by organisations such as Bebras (discussed below) to introduce both CT and computing concepts to students (see http://www.bebras.org).

In Ireland, as in other countries, Computer Science (or computing) is not yet a state examination subject. Although steps to include it have been taken – it will be introduced in 2018 in 40 pilot schools and examined for the first time in 2020 (O'Brien, 2017) – all that is currently available in the curriculum is a Junior Certificate coding short course (NCCA, 2016). While programming is a very useful skill and one that may be beneficial to students in a variety of careers and paths in life, it is not the only component of Computer Science. Lu and Fletcher (2009) compare programming in Computer Science to a literary analysis in English or proof construction in mathematics; it is a more advanced skill than reading, writing and arithmetic.

Research shows that an early introduction to computing is an advantage for students. It can build confidence in dealing with complexity and with open-ended problems (Yevseyeva & Towhidnejad, 2012). Problem-solving skills can be extended and transferred (Koh, Repenning, Nickerson, Endo, & Motter, 2013) and students' analytical skills can be improved (Lishinski, Yadav, Endbody & Good, 2016; Van Dyne & Braun, 2014). It has been shown that students' self-efficacy for computational problem solving, abstraction, debugging and terminology can be increased (Webb & Rosson, 2013). It has also been found that teaching CT can provide a better understanding that programming is about solving a problem (not just a code) and that it can improve female students' attitudes and confidence towards programming (Davies, 2008). One especially interesting finding is that CT can be used as an early indicator and predictor of academic success since CT scores have been found to correlate strongly with general academic achievement (Haddad & Kalaani, 2015).

With the introduction of programming to the curriculum and the call from administrators and governments to include more Computer Science content in schools, we have begun developing a comprehensive course for Transition Year (TY) students. By teaching Computer Science topics including programming, web development and non-computer-based (unplugged)

activities, our goal for this course is to develop students' CT skills as well as their knowledge of Computer Science. This paper outlines the delivery of a pilot version of the course and presents feedback and future plans for its development.

Origins of the Pilot CT Course

The idea to develop the course arose from a need identified by our research group as we worked with schools around Ireland. We observed that teachers were keenly interested in delivering Computer Science lessons and this led to more schools and teachers joining the programme. It has been our intention from the outset to expand the content on offer and to investigate what other topics and methods could be used (Mooney et al., 2014).

It was felt that there was an opportunity and a desire to create a more complete and intensive course for Transition Year, with a view to developing it into a Junior Certificate short course. In September 2016, teachers who had previously been involved with our group, as well as others including trainee teachers, were asked for their ideas on course design and content. This feedback, in conjunction with input from our group members and an extensive literature review, led to the setting out of the following aims which are presented in no particular order:

- Introduce students to Computer Science, what it is, how it can affect their lives, how they can be involved
- Improve students' CT and problem-solving skills by making them aware of a problem-solving process and how it can be beneficial in many subjects and areas of life
- Improve students' understanding of Computer Science including their awareness of both gender imbalance in participation rates and stereotyped views of who engages in Computer Science
- Teach students Computer Science concepts such as algorithms, cryptography, and sorting/searching algorithms with a focus not just on the concepts themselves but on real-world applications
- Teach students programming to some level.

Students who have participated in PACT courses in the past have commented that the modules had been both enjoyable and a good way to develop programming and other skills such as team work. However, they also

stated a desire for more practical applications and we have been working to ensure that the topics and methods used in this course reflect their feedback (Mooney et al., 2014).

PILOT LESSONS

Due to the nature of the study, which involved working directly with young people and collecting data from them, ethical approval was sought and received from the Maynooth University Research Ethics Committee. During the spring term of 2017 (March - May) a pilot study was run which took sections of the developed course and delivered them as part of a 10-week series. Over the course of the 10 weeks, two lessons were delivered per week (40 and 35 minutes each) to one TY class (with students aged from 15 to 17 years). The study was conducted in a mixed-gender, fee-paying school (chosen as the first author was an alumnus) in a town in the east of Ireland, with an enrolment of 300 students.

The school was informed of the study requirements and expectations, namely a class of 20+ TY students with access to computers and sufficient classroom space. Staff in the mathematics department of the school selected 22 students whom they felt would benefit from the course. Most were described by their teachers as being stronger than average in mathematics.

The aims of the pilot study were to examine the extent to which the course was enjoyable for the target age group and its content appropriate. The study also examined student outcomes including whether students had learned something from the course and whether there were any improvements in problem-solving skills or changes in attitudes to Computer Science. After completing each lesson students filled out a feedback form (see below) which sought information on their understanding of the topics covered and on the extent to which they had enjoyed the lesson and learned something from it. Over the course of a 10-week period 20 classes were delivered to the students. In general, the classes consisted of non-computer-based ('unplugged') lessons and Scratch programming lessons. Scratch (https://scratch.mit.edu/) is a programming language designed to introduce students, mainly children and adolescents, to the ideas of programming. It is a drag-and-drop type language which lends itself well to teaching and interactivity. An overview of the main topics covered in each lesson is presented in Table 1, which precedes in-depth descriptions of two of the lessons.

Table 1
Schedule of Lessons Taught during the Pilot Study

Week	Lesson 1	Lesson 2
1	Surveys, Introduction to course	Problem-solving test
2	Introduction to CS	Introduction to CT
3	Algorithms 1	Algorithms 2
4	Searching & Sorting	Searching & Sorting
5	Cryptography	Cryptography
6	Programming concepts	Scratch
7	Scratch	Scratch
8	Scratch	Scratch
9	Graphs	Problem-solving test
10	Surveys, logic puzzles/games	Logic puzzles/games

Algorithms 1 (Week 3, Lesson 1)

To begin this lesson, the following situation was proposed: *Imagine an alien has come to Earth and wants to learn how to do some basic human tasks (maybe hearing about these through radio waves!). The tasks include things like making a cup of tea, baking a cake, ordering pizza etc.*

Students were asked to write up a list of instructions to complete the tasks. After 10 minutes or so, we brought the class together to tell the students that they had just developed an algorithm! We explained that an algorithm is important in a variety of different areas of life, and especially in computer programming, and, by giving examples of real-life algorithms (recipes, directions, etc.), we helped to solidify the idea. Next, we introduced a useful way of developing algorithms called step-wise refinement. This involves beginning with two or three 'big' steps and refining these down into smaller steps. Students then had a go using this method with a different task, this time in pairs.

After students had completed this task, and received feedback from us, we proposed that the aliens wanted to learn a game. Using rock-paper scissors, students were asked to write an algorithm for how to play the game. This task can be used to introduce ambiguity and testing, for example when students

swap their instructions with those of another pair and try to play the game according to the algorithm given.

Programming Concepts (Week 6, Lesson 1)

This lesson was designed as a brief introduction to basic programming concepts and it is intended that it will be split into two lessons in further development of the curriculum. The goal of the lesson was to provide students with fundamental knowledge of programming concepts, namely, loops, conditionals, variables and user input. Examples of these four concepts were given. For example, in the case of loops, the example of a runner running around a track of a specific length was used, as well as a written pseudo-type code variant provided on the whiteboard. After students understood the general concepts, they were given the task of designing a game using them.

For the game design, students were given a variety of objects including dice, chess pieces, counters and playing cards and given free-reign to make up a new game using the objects and the programming concepts. A basic example of keeping score (using variables) was outlined. A player then rolls the dice (user input), increasing his or her score by the amount displayed and a second player does the same (a loop) until one of them reaches a score of 10 (conditional).

This lesson was very well received. However, it was hard to judge whether key concepts were learned effectively. We believe more time is needed to consolidate key concepts, a process that may be helped when students learn programming using Scratch and Python.

EVALUATION TOOLS

For the assessment of the course, several surveys and feedback forms were created. A problem-solving test was also developed.

Personal Survey

In the first week of the course, students completed a personal survey to obtain demographic data and other personal data such as age, gender, and previous programming/Computer Science experience.

Lesson Feedback Form/End-of-Course Survey

Upon completion of each lesson or class, students filled out an anonymous feedback form. They were asked about whether they had enjoyed the lesson, what they had enjoyed/not enjoyed and what they had learned. An anonymous end-of-course survey with similar types of questions was also prepared.

Problem-Solving Tests

To determine whether any impact was made on students' problem-solving skills, which we believe are indicative of CT skills, problem-solving tests were devised and administered at the beginning and end of the course.

The format for the test is based on the Bebras competition, in which the PACT team is heavily involved, and a method used by Grover and Pea (2013). The Bebras competition is held annually in many countries. Bebras problems are designed to introduce Computer Science concepts and to test CT skills that do not require any prior technical knowledge. Questions are submitted and then vetted and edited to suit different age groups. There are six age-group levels, all with an A, B and C section. Each question relates to a Computer Science concept and this is highlighted by the competition organisers with a description of how it reflects 'Computational Thinking'.

The questions for the tests used in this study were taken from problems included in the first round of the UK competition in 2015/16. Thirteen problems were selected from a wide range of topics. A trial run of the test to be taken at the beginning of the course (Test 1) was given to eight people (males and females of different ages) from a variety of work and educational backgrounds. This helped to determine both time requirements and level of difficulty. In the second test (Test 2), designed to be taken at the end of the course, each question was selected so that it corresponded in difficulty level to its equivalent in Test 1. An effort was also made to select questions similar in style and content for both tests but this wasn't always possible. The same group of eight people completed a trial run of Test 2. An online version of the tests can be found at: http://www.cs.nuim.ie/~amooney/CT/.

View of Computer Science Survey

This survey was administered before and after the course and was designed to better understand and evaluate students' views of what Computer Science is, what it involves, and who a computer scientist is. The questions were based on a survey developed by Taub, Armoni and Ben-Ari, (2012) and are reproduced in Table 5 in the next section.

RESULTS

One of the aims of this small-scale pilot study was to determine whether the Computer Science course the PACT team developed was enjoyable and appropriate for TY students. Care should be exercised in interpreting the results as the number of participating students is small, and students were purposively selected to take part. Although 22 students took the course, not all of them attended all the lessons or completed all the evaluation activities.

Personal Survey Outcomes

The personal survey provides demographic and other personal data such as age, gender, previous programming/Computer Science experience. Table 2 presents a summary of these data. The gender breakdown is 12 males and 9 females. Nineteen had taken higher level mathematics in the Junior Certificate

Table 2
Demographic Data and Backgrounds of Participating Students

Demographic & other personal information	No. of students
Gender of Students	Male - 12 Female - 9
Student Age	15 years old - 7 16 years old – 11 17 years old – 3
What level of maths did you take in the Junior Certificate Examination?	Higher Level - 19 Ordinary Level – 2
Previous programming experience	Yes - 9 No - 12
Average rating of programming level	Mean = 2. 44 *Likert Scale (1-5) 1 = very poor, 5 = very good*
How often do you program	Mean = 1.89 *Likert Scale (1-5) 1 = not at all, 5 = daily*
Parents work in IT-related jobs	Yes - 9 No - 12
Programming languages previously used	Python, C#, Scratch, Javascript, C, C++
Any website/app development experience?	Yes - 6 No - 15
If yes, what did you use?	HTML, Javascript, XCode, App Inventor
Have you heard of. . . . . ?	Codecademy - 4, Khan Academy - 1, Call to Code - 1, None of the given options or similar - 16

N = 21.

Examination and two had taken the subject at Ordinary level. Fewer than half of the students reported some previous programming experience while a smaller proportion had used web technologies. The average self-assessed programming level reported was 2.44 out of 5.

Lesson Feedback Outcomes

Results of analysis to the question, *"Did you enjoy the class?",* in the lesson feedback form are provided in Table 3 along with a gender breakdown. A Likert Scale rating of 1-5 was used for responses, where 1 = *"I did not enjoy the class at all",* and 5 = *"I really enjoyed this class"*.

Overall mean ratings are similar across lessons (ranging from 4.1 to 4.6). The range in values is somewhat larger for male students (3.5 - 4.6) than for female students (4.0 - 4.7). Female students had the highest enjoyment ratings for Introduction to Computer Science while male students had the highest enjoyment ratings for algorithms. Introduction to CT had the lowest enjoyment rating for male students whereas female students had the lowest enjoyment rating for Cryptography. There are no statistically significant gender differences. Take, for example, the lesson on Searching & Sorting. The average enjoyment level reported was 4.4 out of 5, with female students having a higher enjoyment rating of 4.6 compared to 4.3 for male students. The difference was not statistically significant, χ^2 (15,16) = 19.5, p = .70.

Table 3
Lesson Feedback – Average 'Enjoyment of Class' Ratings by Gender

Topic name	Total	Males	Females
Introduction to CS	4.6 (8)	4.5 (2)	4.7(6)*
Introduction to CT	4.1 (9)	3.5 (2)	4.3(7)
Algorithms	4.5 (16)	4.6 (10)	4.3 (6)
Searching & Sorting	4.4 (16)	4.3 (7)	4.6 (9)
Cryptography	4.2 (12)	4.4 (8)	4.0 (4)
Programming concepts	4.5(13)	4.4 (9)	4.5 (4)
Graphs	4.4 (13)	4.4 (9)	4.3 (4)

*Mean ratings (5 = high, 1 = low). Numbers of students in brackets. CS = Computer Science; CT = Computational Thinking.

End-of-Course Survey Outcomes

On the final day of the course, a feedback form was distributed and completed by 17 students who were present on the day. The questions and responses are presented and discussed below.

1. Did you enjoy the course?
   Analysis of responses to this question indicates a class average rating of 3.8 out of 5. Males (with a score of 4.0) had a more enjoyable experience of the course than females (who had a score of 3.5).
2. What did you like?
   Students enjoyed learning about Computer Science for the first time and learning new things about mathematics and computers. Group work was also mentioned as something students liked along with the games. Referring to the algorithms lessons, students stated that they "enjoyed...working with my friends" and "liked how practical the class was". The unique concepts and variety of classes were also mentioned as highlights.
3. What didn't you like about the course?
   Not much information was given in response to this question. One student reported not liking programming with Scratch. From observation and conversations with students, we believe that hands-on activities were much more popular than Scratch programming. Two students stated that the course seemed very long, although it should be noted that one had very low attendance. It was also stated that the course was complicated and that there was not enough help.
4. Favourite class/activity or topic covered?
   Problem solving, Scratch, principles of computer science, algorithms, doing the test, and linear sorting were identified as favourite activities.
5. What was your least favourite class/activity or topic covered?
   Scratch, including understanding the tutorials, some of the puzzles, algorithms, and different ways to filter code were mentioned as least favourite activities.
6. Do you feel that you learned something during the course?
   Apart from one negative response, all students responded positively to this question.
7. What did you learn?
   In response, students mentioned new computer skills, how to solve different types of problems, new Computer Science terms, how to programme using Scratch, how to make games, what Computational Thinking is and how it works, and different types of sorting.
8. If you could change something about the course what would it be?

> Of the small number who responded to this question, the suggestions included making the tutorials easier, providing more help, setting aside programming, making the classes longer, adding more practical material, and identifying more everyday uses.

Overall, students reported learning a wide array of skills. These included core Computer Science concepts such as programming, sorting and problem solving. In addition, they gained an understanding of what Computer Science is, what Computational Thinking is, and the differences between them.

In analysing the feedback, it was observed that the problem-solving aspects of the course were well received with several students commenting that they enjoyed the different puzzle types used. We plan to add more puzzles to the course to allow students to improve their problem-solving skills – a key requirement for a computer scientist. Students also enjoyed the variety of concepts introduced in the course and the different teaching styles that were used. Using this feedback, we will make further adaptations to include more tutorials and support documentation to help those students who are struggling. We have already added more practical materials to allow students to interact with each other and with different groupings in their class.

Problem-Solving Test Results

The first problem-solving test was administered to 22 students at the beginning of the course (Test 1). Of these, two students completed Test 1 after attending one class/lesson and one other student completed it after attending two lessons. The second problem-solving test (Test 2) was completed by 19 students. Results of the pre-test and post-test are presented in Table 4.

The average scores are 7.18 for Test 1 (N = 22) and 6.84 for Test 2 (N = 19. The difference between the scores is not statistically significant ($t(39) = 0.51, p = 0.61$). The average scores of students who took both tests (n = 19) are 7.47 (Test 1) and 6.8 (Test 2). Again the differences are not statistically significant ($t(18) = 1.00, p = 0.32$). A finding of note is that, on both tests, those students who had previous programming experience (n = 9 in both) performed better than those who did not have that experience (n = 12 for Test 1, n = 10 for Test 2). Those with prior experience averaged a score of 8.3 for Test 1 and 8.0 for Test 2, while those without experience averaged 6.8 for Test 1 and 5.8 for Test 2. For the first group (those with prior experience), the difference in average scores between the two tests is not statistically significant ($t(8) = 2.13$, $p = 0.55$). For those without prior experience, however, the score difference is

statistically significant ($t(9) = 3.45$, $p = 0.007$), suggesting that experience mattered.

Table 4
Outcomes of the Pre- and Post- (Problem-Solving) Tests

Student	Test 1	Test 2	Student	Test 1	Test 2
A	8	10	L	8	7
B	7	-	M	8*	5
C	8	7	N	5	6
D	10	11	O	10	7
E	9	7	P	10	7
F	6	5	Q	9	7
G	7	6	R	1	-
H	8	-	S	5	3
I	7	6	T	5**	7
J	9	9	U	3*	4
K	8*	8	V	7	8
			Mean	7.2	6.8

Maximum score = 13 (1 per problem); * test taken after one class; ** test taken after two classes.

View of Computer Science Survey Outcomes

Of the 22 students enrolled to take part in this course, 18 filled out the pre-course survey, while 15 completed the post-course survey. The data obtained are displayed in Table 5.

The data show that the course does not appear to have had much impact on students' interest in Computer Science. In fact, more students indicated interest in the subject at the beginning of the course than at the end of the course. While the numbers are small and it is difficult to identify any distinct patterns in the data, it is worth noting that after the course students were inclined to agree less that programming is central to Computer Science, that Computer Science is an area that relates to maths and that boys/men are more likely to study Computer Science than girls/women. Another indication of a change in views, or broadening of opinion, is reflected in the finding that students agreed more after the course that a computer scientist should be good at working with people. However small these indications, we believe that students who are exposed to Computer Science at an early age will be able to make more

informed decisions in the future. Currently students do not have exposure to Computer Science at second level and any exposure, be it positive or negative for students, can contribute to decisions about choice of third-level courses. Computer Science has one of the highest drop-out and non-continuation rates across all third-level subjects (Quille, Bergin & Mooney, 2015). Hence, if students find out at second level that they don't like studying Computer Science, or are not suited to it, this is valuable.

Table 5
Results of the View of Computer Science Survey

Question	Before (18) %	After (15) %
Have you considered studying CS in university? (Yes)	16.7	20.0
Is CS interesting to you? (Yes)	55.6	46.7
Is CS challenging? (Yes)	38.9	53.3
Have you heard of the term CT?	27.8	60.0
	Mean	Mean
Using the internet is central to CS	3.4	3.3
Using Word, Excel etc. is central to CS	3.2	3.1
Installing software (e.g. Windows, iTunes) is central to CS	3.7	3.2
Programming is central to CS	4.6	4.1
Being able to solve different problems is central to CS	4.7	4.6
CS is an area related to maths	4.1	3.6
A computer scientist should be good at working with others	3.6	4.0
Boys/men are more likely to study CS then girls/women	3.3	2.6
Work in CS can be done without a computer	2.6	3.1

In addition to a Yes/No options for the first three questions, there was also a 'Maybe' option. CS = Computer Science. CT = Computational Thinking. The numerical values range from 1-5 (1 = strongly disagree, 5 = strongly agree).

PRACTICAL LESSONS LEARNED

Attendance

Due to the fact that TY students were involved in the study, a problem arose in relation to inconsistent attendance. TY is a programme in which students are encouraged to try out new subjects, engage in new activities and pursue extra-curricular activities (DES, n.d.). This makes it an ideal fit for trialling a Computer Science course. However, extra-curricular and other activities can disrupt class attendance, which, in turn, can impact in a negative way on programming lessons. Programming is a skill that is learned incrementally and so students who miss lessons will need to catch up as the topics and tools taught in each lesson are vital to the next phase of learning.

Facilities

Two rooms were used for the duration of the study. The first was an open-spaced room with movable chairs and tables. This was a perfect location for the concept lessons which involved individual pen and paper activities as well as group activities. The classroom structure was also beneficial for active learning and student-led activities; it was set up with tables allowing groups of four to sit around and work either together or in two pairs on most activities. From our experience this kind of group work is both enjoyable and promotes learning, though students took a while to adapt to the sequence of teaching employed in the course. In general, this involved a brief introduction followed by group, pair or individual work. Several comments were made in feedback forms to the effect that students enjoyed the interactive nature and group work.

The second room used for the study was a computer room with roughly 20 computers. These were older machines and several of them crashed during the lessons, losing internet connection on occasion. Also, there was no screen and no wall for projecting onto, so we could not show students how to use Scratch, create an account, or navigate to tutorials. These flaws were mentioned by students who would have appreciated a better introduction to Scratch. Considering our experience of this course, or any similar course that includes programming, it is important that good-quality facilities are made available. Primarily, this means students having one-to-one access to a personal computer with fast internet access and the memory and processing capability required to run multiple software programmes. Projectors or TV monitors which allow teachers to show students examples and 'live-code' are also necessary to make the teaching experience easier for both students and educators.

Class Time

Instruction time was limited to one 40-minute class and one 35-minute class per week, separated by a lunch break. This meant that many students did not have an opportunity to complete as much of the Scratch course as they would have liked. Another issue was that with the break, and students returning late, the second lesson was often reduced to 30 minutes. This, combined with the need to fill out a feedback form at the end of each lesson/class, reduced instruction time considerably. This, in turn, meant that a few exercises were not tested. All things considered, however, the experience was beneficial as we have changed our approach to lesson plans for teachers in the course that is currently being developed. All lessons will be designed with enough content to last for one hour for schools that have hour-long classes, but will allow

schools with shorter classes to remove certain (recommended) activities whilst including the more core parts of the course.

CONCLUSIONS

Overall Response to the Course

While overall feedback from students was positive, we are aware that considerable work is required to prepare for the delivery of Computer Science courses in schools and the integration of CT into other subjects. As a result of the study, we have feedback to fine tune programmes such as Scratch to ensure that the materials we are currently developing are sufficiently challenging, interesting for students, and age-appropriate. We also know more about the standards of facilities and instruction time that will be required in schools for optimum delivery.

Performance on the Problem-Solving Test

As noted in the results section, students on average scored slightly lower on the second problem solving test compared to the first. Although not statistically significant, this is a disappointing result. We believe several factors might have led to this. One is the sporadic attendance of students as discussed in the practical lessons learned section. Another is the duration and intensity of the course, as sufficient time may not have provided for students to learn the lesson content fully. There is also the possibility that the two tests are not exactly equal in difficulty. Although efforts were made to ensure this, and a more thorough verification process is currently taking place, this possibility should not be discounted. It should also be noted that although the course includes lots of activities which are designed to improve problem-solving skills the students are not 'trained' on Bebras problems, so they may not necessarily improve in this test environment given the short time period involved.

Future Work

Several schools have been contacted with a view to rolling out a revised course. So far, teachers have been enthusiastic and many have committed to becoming involved. Our intention is that upwards of 100 students will take part in at least a portion of the course and provide us with more feedback to help us further enhance its development. We will also be seeking the views of teachers

and asking them to critique course content with a view to adapting it based on their recommendations and experiences.

One problem that can be foreseen from rolling the course out in multiple schools for varying amounts of time is data collection. To this end, we hope to develop an online system to collect data and deliver lessons. This will include all feedback collection, problem-solving test data as well as all content, lesson plans and additional exercises. This will hopefully enable teachers to customise lessons to suit their purposes and contexts.

Longer-term, the new Leaving Certificate Computer Science course will require content to be developed and it is a possibility that the content created as part of this project can be adapted and incorporated into a course that meets all the recently-published curriculum specifications. Although both this course and the Leaving Certificate course are at an early stage of development it is our belief that some of the teaching methods and content we have developed could be of benefit – in particular, the use of 'unplugged' lessons to teach concepts and key topics such as algorithms and searching and sorting. Our use of projects also aligns well with the proposed Leaving Certificate curriculum.

We believe that a number of topics in our course will match at least a few of the proposed curriculum learning objectives. These include the following: developing an understanding of how Computational Thinking presents new ways to address problems; using CT to analyse problems and design, develop and evaluate solutions; reading, writing, testing and modifying computer programmes; developing an understanding of how computers work, the component parts of computer systems and how these interrelate, including software, data, hardware, communications, and users; and working independently and collaboratively, communicating effectively, and becoming responsible, competent, confident, reflective, and creative users of computing technology. Our course interfaces with three proposed strands for a new Computer Science Leaving Certificate course. Overlap between the strands and our pilot course is shown in Table 6. We hope that our on-going work and the lessons learned will influence and inform developments at Leaving Certificate level.

Table 6
Links between Strands in Proposed Leaving Certificate Computer Science Course and Pilot Course Topics

Strand	Lessons in our course
Practices and Principles	Introduction to CT, Algorithms, Introduction to Computer Science, various projects
Cross-cutting Core Concepts	Algorithms, Data, Intro to CT, Finite Automata, Searching & Sorting
Computer Science in Practice	Python, Web Systems, App Development

ACKNOWLEDGEMENT

Our thanks to the students and school staff who participated in this study for facilitating our work. Also, thanks to all the teachers who shared their ideas with us about course content and design.

REFERENCES

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender-relevant differences. *Robotics and Autonomous Systems*, *75*, 661-670.

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, *1*(2), 67-69.

Caspersen, M. E., & Nowack, P. (2013, January). Computational thinking and practice: A generic approach to computing in Danish high schools. In A Carbone and J. Whaley (Eds.), *Proceedings of the Fifteenth Australasian Computing Education Conference, 136* (pp. 137-143). Darlinghurst, Australia: Australian Computer Society.

Davies, S. (2008). The effects of emphasizing computational thinking in an introductory programming course. In *Frontiers in Education Conference, 2008. Proceedings. 38th Annual* (pp. T2C-3). Piscataway, NJ: Institute of Electrical and Electronic Engineers.

Denning, P. J. (2009). The profession of IT: Beyond computational thinking. *Communications of the ACM*, *52*(6), 28-30.

DES. (Department of Education and Skills). (no date). Transition year. Accessed at: https://www.education.ie/en/Schools-Colleges/Information/Curriculum-and-Syllabus/Transition-Year-/

Folk, R., Lee, G., Michalenko, A., Peel, A., & Pontelli, E. (2015). K-12 dissect: Incorporating computational thinking with K-12 science without computer

access. In *Launching a New Vision in Engineering Education. Frontiers in Education Conference 2015. Proceedings* (pp. 1315-1322). Piscataway, NJ: Institute of Electrical and Electronic Engineers. Accessed at: http://fie2015.org/sites/fie2015.fie-conference.org/files/FIE-2015_Proceedings_v11.pdf

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38-43.

Haddad, R. J., & Kalaani, Y. (2015). Can computational thinking predict academic performance? In *Integrated STEM Education Conference (ISEC), 2015 IEEE* (pp. 225-229). Piscataway, NJ: Institute of Electrical and Electronic Engineers.

Koh, K. H., Repenning, A., Nickerson, H., Endo, Y., & Motter, P. (2013, March). Will it stick? Exploring the sustainability of computational thinking education through game design. In *Proceeding of the 44th Technical Symposium on Computer Science Education* (p. 597-602). New York: Association of Computing Machinery.

Li, W. L., Hu, C. F., & Wu, C. C. (2016). Teaching high school students computational thinking with hands-on activities. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 371-373). New York: Association of Computing Machinery. Accessed at: https://sgd.cs.colorado.edu/wiki/images/4/44/Will_It_Stick-submit_CR.pdf

Lishinski, A., Yadav, A., Enbody, R., & Good, J. (2016, February). The influence of problem-solving abilities on students' performance on different assessment tasks in CS1. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 329-334). New York: Association of Computing Machinery.

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin*, *41*(1), 260-264.

Mooney, A., Duffin, J., Naughton, T., Monahan, R., Power, J. F., & Maguire, P. (2014). PACT: An initiative to introduce computational thinking to second-level education in Ireland. In *Proceedings of the international conference on engaging pedagogy (ICEP)*. Athlone: Institute of Technology. Accessed at: *http://icep.ie/paper-template/?pid=112*

NCCA. (National Council for Curriculum and Assessment). (2016). Short course: Coding. Specification for Junior Cycle. Dublin: Author. Accessed at: https://www.curriculumonline.ie/getmedia/cc254b82-1114-496e-bc4a-11f5b14a557f/NCCA-JC-Short-Course-Coding.pdf

O'Brien, C. (2017, February 6th). Computer science to be fast-tracked onto Leaving Certificate. Irish Times. Accessed at: https://www.irishtimes.com/news/education/computer-science-to-be-fast-tracked-onto-leaving-cert-1.2964672

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York: Basic Books.

Quille, K., Bergin, S., & Mooney, A. (2015). PreSS#, a web-based educational system to predict programming performance. *International Journal of Computer Science and Software Engineering (IJCSSE)*, *4*(7), 178-189.

Roscoe, J. F., Fearn, S., & Posey, E. (2014). Teaching computational thinking by playing games and building robots. In *Proceedings of the International Interactive Technologies and Games Conference* (ITAG), pp. 9-12. Piscataway, NJ: IEE Press.

Shailaja, J., & Sridaran, R. (2015). Computational thinking: the intellectual thinking for the 21st century. *International Journal of Advanced Networking & Applications, May 2015 Special Issue*, 39-46.

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education* (TOCE), *12*(2), 8.

Van Dyne, M., & Braun, J. (2014, March). Effectiveness of a computational thinking course on student analytical skills. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 133-138). New York: Association for Computing Machinery.

Webb, H., & Rosson, M. B. (2013). Using scaffolded examples to teach computational thinking concepts. In *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 95-100). New York: Association for Computing Machinery.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725.

Yevseyeva, K., & Towhidnejad, M. (2012). Work in progress: Teaching computational thinking in middle and high school. In *Frontiers in Education Conference (FIE), 2012* (pp. 1-2). Piscataway, NJ: Institute of Electrical and Electronic Engineers.