Kong, S. C., Sheldon, J., & Li, K. Y.. (Eds.). (2017). *Conference Proceedings of International Conference on Computational Thinking Education 2017*. Hong Kong: The Education University of Hong Kong

An investigation into susceptibility to learn computational thinking in post-compulsory education

Ana C. CALDERON[1*], Tom CRICK[1] and Catherine TRYFONA[1]

[1] Department of Computing & Information Systems, Cardiff Metropolitan University, Cardiff CF5 2YB, UK
{acalderon,tcrick,ctryfona}@cardiffmet.ac.uk

ABSTRACT

This paper presents the results of a preliminary investigation into how the teaching of computational thinking -- particularly algorithmic thinking and programming -- to university undergraduate students varies depending on aptitude and perceived enjoyment of STEM subjects during their secondary-level (pre-university) education. We investigated a specific component of computational thinking, algorithmic thinking, comparing against a student's ability to develop knowledge and understanding of introductory programming.

KEYWORDS

Perceptions, Algorithmic thinking, Computational thinking, STEM

1. INTRODUCTION

Computational thinking [Papert 1996; Guzdial 2008; Wing, J. (2008)] is increasingly being integrated into various national curricula, being regarded as a key skills, with wide potential utility, for school-age children. It is recognised both for its important role in developing knowledge and understanding of foundational computer science concepts, but also for its potential in developing more general-purpose problem-solving skills across the curriculum. This paper investigates whether algorithmic thinking (an integral part of computational thinking) can be as easily taught to those with a natural interest in computational science and those who do not process such an interest, and whether this changes with aptitude to more technical subjects in school. Aptitude and interest are restricted as to what students preferred subjects subjects were at the time of secondary school graduation.

There are many views of computational thinking, for instance a recent report of a workshop shows the range of definitions, and opinions on the subject (NRC 2010) Some researchers adopt the original notions of procedural thinking, as developed by (Papert 1981) to define what Computational Thinking is. This view sees it as a step-by-step list of detailed and unambiguous instructions such that can be interpreted and executed by an automated agent. Others view it as an effort to expand the human capacity for problem solving, by providing abstract tools able to aid in the management of tackling complex tasks. A lot of researchers also dismiss the notions of linking computational to the processing of numbers, whereas some argue it is a way of enabling humans to solve problems by means of providing precise methods for doing so. Whatever viewpoint adopted, most researchers seem to agree that computational thinking is an integral part of computer science [Tedre 2016]. The skill set learn by studying Computational Thinking is complementary to more established areas taught at HE computing degrees. This investigation looks at students' aptitudes to STEM and Humanities in the final two years of school, in an attempt to see whether there are negative or positive correlations to leaning elements of Computational Thinking and of a core element of Computing degrees, programming. Focusing particularly on algorithmically thinking and on object-oriented programming, we found that an aptitude in STEM favoured performance in learning object-oriented programming notions, but found no difference between aptitudes in humanities and in sciences when learning Algorithmically Thinking (Futschek 2006) with a methodology highlighted in later sections.

2. Methodology

2.1 The Research Question

Our interest is on whether particular preferences in secondary school have a positive correlation with ability to learn algorithmically thinking in Higher Education. Using the methodology above we measured data gathered from students about attitudes and aptitudes of STEM-based and other subjects and how well they performed on the particular algorithm course.

2.2 Pedagogical Investigation

The investigation took part over two semesters in one academic year; one semester the students participated in an algorithm class, and the second semester different students participated in an object-oriented programming class. The choice for using different groups of students was due to the transfer of knowledge, performance in a latter module, for instance object-oriented programming could have been enhanced by attending an earlier, for instance, algorithmic thinking module.

We designed a one semester course such focusing on teaching algorithmic thinking to first-year, first-semester students enrolled in three undergraduate degree programmes: Computer Science, Software Engineering and Business Information Systems. Students participated in a total of 11 weekly sessions, where each session consists of three components, distributed during the week.

Algorithmic Thinking

The sessions consisted of:

- Part A consists of a one hour session (workshop) of a hands-on puzzle solving activity.
- Part B consists of a formative learning session (a one hour lecture)

- Part C consists of a one hour session (workshop) of a puzzle that includes writing pseudocode.

For the workshops (Parts A and B) students were required to work in groups. The fist session was purposely kept simple, and we now use it as an example of the methodology, it consisted of:

- Part A (workshop): present students with physical copies of Tower of Hanoi puzzles with a large number of even and of odd disks.
- Part B (lecture): lecture on recursion
- Part C: (workshop) Tower of Hanoi puzzles handed out to students again, and asked them to write pseudocode to solve a Tower of Hanoi with either an even or an odd number of disks (students who do not immediately recognize recursion are given extra support until they are able to connect the concept from the lecture to the example from the workshop).

For another illustrative example, we detail the second session. The main aim behind this session was to develop understand of sorting algorithms. Students were given cardboard pieces with numbers written on it, ranging 1-100, and asked to find the maximum. Following the same pattern as all other sessions, students were placed in groups. Differently from other sessions, they were asked (in their groups) to first think about attempting to find the maximum value of the numbers (sorting the cards) if they could only work by themselves, then if they could only work within the group, and finally to think about how they would solve if the groups could talk to each other and divide the cards. The idea behind this is to aid participants in teaching themselves what an algorithm is as well as to bring their awareness to the existence of parallelism as a means to efficiency. This session is based on ideas developed in (Adams 2005).

For the formative learning portion of the session students were taught the concept of a sorting algorithm and presented with some standard examples of sorting algorithms, namely insertion sort, selection sort, merge sort, heapsort, quicksort, bubble sort and variants. For the final workshop (Part B) of this particular session, students were given Rubik's cubes and given 3 sequences of moves, then asked to use these sequences to solve the cube, and write a pseudocode for their solution (an algorithm that would sort all sides to the desired configuration).

Programming

Teaching introductory programming within Higher Education can be particularly challenging due to the diversity of educational background of incoming undergraduate students, as a single annual intake of students is likely to include a broad range of prior learning experiences. As a consequence of school-level computer science education reform (Brown *et al*, 2014), an increasing number of first year students are likely to have had some exposure to programming in schools or colleges. Some students, perhaps through their own extracurricular efforts, may have developed considerable technical skills. This variance in ability seemingly increases the risk of disengagement because the teaching material may either be viewed as too difficult (Mohd *et al*, 2013) or too simplistic. It could be argued, however, that software development and programming is an art as much as it is a science and that undergraduate students can best develop their programming skills through apprentice-style learning (Kolling and Barnes, 2008; Bennedsen and Caspersen, 2008). Recently, there has been more emphasis placed on the importance of “software carpentry” skills, so that student can develop a sense of “craftsmanship” towards the design and development of software solutions to real world problems. Seminars and tutorials can particularly lend themselves to this style of delivery, where experienced teaching staff are not only able to demonstrate the technical skills, but also explain the thinking behind the decisions that they make (Kolling and Barnes, 2008).

Given that sound computational thinking skills aids in most stages of the software development process, there is an increasing and explicit emphasis on developing these skills in modern undergraduate computing curricula. By focusing on key skills such as algorithmic thinking from early on in a programmer’s career, students can more readily contextualise programming as a tool to be used for expression of creativity and for problem solving. Students are able to analyse problems and formulate a solution computationally (Cesar *et al*, 2017). An emphasis on computational thinking within the context of apprentice-style learning, may reduce the risk of disengagement as more technically-able skills will have the opportunity to refine their skills under the guidance of a more experienced academic member of staff.

Similarly to algorithmic thinking, the sessions were broken down into formative and practical learning, namely they consisted of:

- Part A consists of a formative learning session (a one hour lecture)
- Part B consists of a two hour practical session (coding the concepts learnt in the lecture).

In particular, during the term each week (note that each week contained Part A together with Part B), was given by:

- Week 1: Introduction to programming, including varying programming paradigms.
- Week 2: Introduction to integrated development environments.
- Week 3: Understanding how to perform operations, and their implications to varying paradigms.
- Weeks 4 and 5:Understading statements and directing values.
- Week 5: Manipulating Data.
- Weeks 6, 7 and 8: Object Oriented concepts.

3. Results

We compared students’ aptitude to STEM subjects and humanities at both A-levels and GSCE with their ability to learn algorithmic thinking, with the methodology highlighted above. More specifically, we focused on students who had grade C and above at a combination of mathematics, computing and physics at A-level, and those who had a grade C and above at a combination of history, literature and drama. The performance of both groups was similar; the first group had an average grade of 62.4%, with a standard deviation of 13.4, whereas the humanities group had an average grade of 61.3% with a standard deviation of 9.4 (see Figure 1 for more details). Of the 92 students used for the first study (algorithmic thinking), 23 had taken the requirements of aptitude in the three stem subjects: mathematics, computing and a science subject, and 17 satisfied the requirements of having taken the humanities English literature, history and drama. For the second study (programming) 21 had taken the requirements of aptitude in the three stem subjects: mathematics, computing and a science subject, and 18 satisfied the requirements of having taken the humanities English literature, history and drama. Although the difference between STEM and humanities for the algorithmic group was significantly small, the difference for a more traditional approach to teaching object-oriented programming was more significantly different, the average programming grade for students with a STEM aptitude was 17.9%, with a standard deviation of 67.1, and those with an aptitude in humanities was 16.7% with a standard deviation of 47.5, more details can be found on Figure 1. This suggests that Computational Thinking approaches are more readily taught to varied skilled students, as compared to the core elements of Computer Science. This suggests that along side standard computer science subjects, HE students might benefit from having a dedicated module of "Computational Thinking" as that would "even the playfield" and thus allow educators to keep the levels of motivation similar to students regardless of their background. We also analysed their ability to write pseudocode.

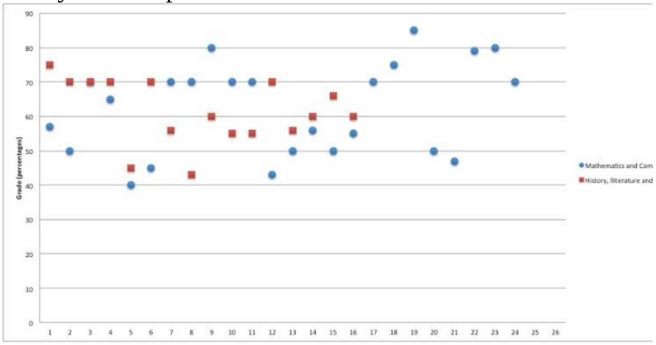

Figure 1. Distribution of grades for algorithmic thinking against humanities and STEM preferences at A-levels

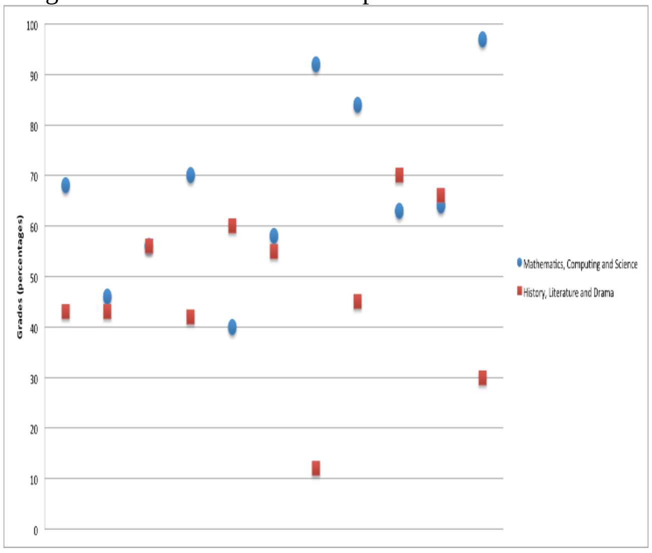

Figure 2. Distribution of grades for programming against humanities and STEM preferences at A-levels

4. CONCLUSION

We presented the beginnings of an on-going investigation into how susceptible students, of varying aptitudes and attitudes, are to learning computational thinking skills.

5. REFERENCES

Adams, R., Bell, T., McKenzie, J., Witten, I. H., & Fellows, M. (2005). Computer Science Unplugged: An enrichment and extension programme for primary-aged children.

Bennedsen, J., & Caspersen, M. (2008). Exposing the Programming Process. In J. Bennedsen, M. Caspersen, & M. Kolling (Eds.), Reflections on the Teaching of Programming: Methods and Implementation. New York: Springer.

Brown, N., Sentance, S., Crick, T., & Humphreys, S. (2014). Restart: The Resurgence of Computer Science in UK Schools. ACM Transactions on Computing Education, 14(2), 9:1–9:22.

Cesar, E., Cortés, A., Espinosa, A., Margalef, T., Moure, J. C., … Suppi, R. (2017). Introducing computational thinking , parallel programming and performance engineering in interdisciplinary studies ☆. J. Parallel Distrib. Comput. http://doi.org/10.1016/j.jpdc.2016.12.027

Futschek, G. (2006, November). Algorithmic thinking: the key for understanding computer science. In International Conference on Informatics in Secondary Schools-Evolution and Perspectives (pp. 159-168). Springer Berlin Heidelberg.

Guzdial, M. (2008). "Education: Paving the way for computational thinking". Communications of the ACM 51 (8): 25.

Kolling, M., & Barnes, D. (2008). Apprentice-based Learning Via Integrated Lectures and Assignments. In J. Bennedsen, M. Caspersen, & M. Kolling (Eds.), Reflections on the Teaching of Programming: Methods and Implementation (p. -). New York: Springer.

Matti Tedre and Peter J. Denning. 2016. The long quest for computational thinking. In Proceedings of the 16th Koli Calling International Conference on Computing Education Research (Koli Calling '16). ACM, New York, NY, USA, 120-129. DOI: https://doi.org/10.1145/2999541.2999542

Mohd, S., Shukur, Z., & Mohamad, H. (2013). Analysis of Research in Programming Teaching Tools : An Initial Review. Procedia - Social and Behavioral Sciences, 103, 127–135.
National Research Council. (2010) Report of a Workshop on the Scope and Nature of Computational Thinking. Washington, DC: The National Academies Press. doi:10.17226/12840.. Chapter 2, page 4.
Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc..]

Seymour Papert, 1981, Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic Books)
Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.
Wing, J. (2008) Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A, 366(1881), 3717-3725