Computational thinking blooms

Introduction

Meet Frank. He was a member of the last cohort of PGCE ICT students, finishing his teacher training in 2013. Frank's first degree was not in either ICT or computer science. He was offered a position as a teacher at one of his placement schools, which delivers computing at KS3 and KS4. Frank had confidence in his abilities to teach the digital literacy and the ICT strands of the computing curriculum (Department for Education, 2013), but the computer science topics were going to be more of a challenge. On top of that, all the focus on computational thinking skills was new to him. What Frank did possess, due to having encountered them in his course, was a good understanding of educational theory, including Bloom's Taxonomy (Bloom, 1956) and the Revised Taxonomy (Anderson, et al. 2001). Therefore, his task was clear: to develop his understanding of computational thinking around the scaffolding he already possessed.

Planting the seeds

Many of the terms used to describe computational thinking were already familiar to Frank, as they will be to teachers of ICT, DT, science, or mathematics. Decomposition, algorithm design, and evaluation are terms used in all of these subjects. Other terms, perhaps less familiar, include abstraction and generalisation. To complicate matters, terms such as visualisation, automation, logical thinking, mathematical/engineering thinking, pattern recognition, and problem-solving are used as well. It's all a bit overwhelming. Luckily, for Frank, there is a smaller set of terms that make a good starting point for understanding computational thinking. These are abstraction, decomposition, algorithmic design, generalisation, and evaluation (Selby and Woollard, 2013).

Abstraction is deciding what details are important and what details can be ignored (Wing 2008). Multiple layers of abstraction can be used to reduce the complexity of a large problem. For example, the cloud is just a word to represent all those machines with attached storage devices where data can be saved. However, the abstraction (the word cloud) allows us to talk about it without concern for how it functions. Frank has certainly encountered abstraction before, including the idea of the World Wide Web and the named subprograms some of his pupils use in their coursework such as calculateAverage().

Decomposition is breaking a problem down into smaller, more easily solved, parts. This is not only relevant in computer science, but is also part of the classic problem solving techniques proposed in mathematics by George Pólya (1985). Science experiments are tasks that benefit from decomposition. For example, a biology class is going to give a presentation on the biodiversity of an area near their school. They divide the tasks into counting flowers in a square metre of grass and counting bugs in a square metre of grass. Together, the results of the individual tasks can be used to create the presentation. Frank saw decomposition being used when a class of pupils was divided into groups with each group writing a script for a sprite to do one dance move. The scripts were joined together to make the sprite perform an entire dance.

Generalisation is a powerful component of problem solving that describes the ability to express a problem solution in generic terms. The advantage of this is that the same solution can be applied to problems that share some of the same characteristics. This definition fits Pólya's description of analogy, the ability to solve a problem based on the known solution to a similar problem (1985). For example, pupils in Year 7 instruct robots to move around a maze as part of an after-school

club. In Year 9, the same pupils are set the task of using LOGO turtles to write their names on the screen. Pupils who identify the commonalities of the two tasks and make the connection that both the maze and the letters all consist of the same type of simple movements, are generalising an approach to a solution. Frank hasn't yet observed generalisation in a classroom, but is looking forward to that day.

Algorithmic design is the development of a step-by-step procedure for solving a problem. David Moursund and Gerald Sussman (National Research Council, 2010) support the association of step-by-step procedures and computational thinking. Although not in her original article (2006), Wing (2011) extends her interpretation of computational thinking to include algorithmic and parallel thinking. Algorithm design, in some form, is practised in many subjects. In ICT, pupils create a plan for how they will conduct a survey and report the results; they will include an order for the work and perhaps deadline dates. These steps are the algorithmic design for producing the survey results. Frank has experienced algorithmic design many times, not just in terms of writing programs, but also in terms of accomplishing everyday tasks, such as making tea or making a sandwich.

Evaluation is identified by Wing (2006), as the skill computational thinkers use when they review algorithms and make trade-offs, in the use of time and space, power and storage. This is subtly different to the way Frank has encountered evaluation before. In his experience, evaluation takes place at the end of a task and it's simply a set of questions such as "does it work?" and "is it suitable?" However, in computational thinking, evaluation goes deeper to ask such questions as "does it satisfy the original objectives?", "are there other ways to solve this problem?", and "are there better (more efficient) ways to solve this problem?"

Charting the growth

Having some idea of the different computational thinking concepts, Frank now has to find a way of connecting it to the knowledge he already has about how pupils learn. Asking other teachers is an approach that appeals to Frank, given he had enough time. Again, he can take advantage of others' research.

A group of secondary and post-16 teachers participated in a project involving computational thinking and the teaching of programming (Selby, 2014). Their responses were analysed to identify connections between the use of computational thinking concepts and the levels of Bloom's Taxonomy.

The following table presents a computational thinking taxonomy aligned with Bloom's Taxonomy. The last column provides examples of behaviours that Frank may see in a classroom.

Bloom's Taxonomy	**Computational Thinking Taxonomy**	**Examples Frank may see in a classroom**
Knowledge		
Comprehension		• Translating between representations of algorithms (flowchart, pseudo-code) • Understanding the purpose of an algorithm as a whole
Application	Generalisation	• Using generalisations (patterns of solutions) • Using abstractions (subprograms, blocks)

Analysis	Abstraction (functional) Abstraction (data) Decomposition	• Recognising a pattern • Recognising a potential generalisation • Recognising connections/relationships • Decomposition of functionality/data • Debugging
Synthesis	Algorithm design	• Designing an algorithm/plan • Creating a representation (flowchart, pseudo-code) of an algorithm • Creating an abstraction of functionality (subprograms) or data (record, array) • Deriving new relationships • Make hypotheses (predict testing results)
Evaluation	Evaluation	• Make trade-offs (time, storage) • Evaluate in terms of original objectives

Harvesting the blooms

Like Frank, teachers already familiar with Bloom's Taxonomy can, when introducing computational thinking into their classroom, use it as a scaffold to identify associated levels for the new concepts. Ordering the computational thinking concepts into a taxonomy parallel to Bloom's allows equivalences to be identified. For example, using programming blocks (abstractions of functionality) belongs at the application level. Creating the programming blocks (abstractions of functionality) belongs at the synthesis level.

Frank can use the computational thinking taxonomy to provision and monitor progress in the same way that he currently uses Bloom's Taxonomy. For example, when he reminds pupils to use the same techniques they have used previously, he is encouraging work at the application level of Bloom's and the generalisation level of the computational thinking taxonomy. However, independently recognising that a new context requires techniques from a previous solution is assigned to the analysis level of both Bloom's and the computational thinking taxonomy, both at the next higher level.

Recognising the connections, between Bloom's Taxonomy and the computational thinking taxonomy, means that Frank does not have to view computational thinking as a set of concepts unrelated to what he and his learners already do in their classroom. Using Bloom's Taxonomy as a scaffold has allowed Frank to more easily incorporate computational thinking into schemes of work and identify pathways of progression for his learners.

References

ANDERSON, L., KRATHWOHL, D., AIRASIAN, P., CRUIKSHANK, K., MAYER, R., PINTRICH, P., RATHS, J. & WITTROCK, M. (eds.) 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives,* New York: Addison Wesley Longman, Inc.

BLOOM, B. (ed.) 1956. *Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook 1 Cognitive Domain,* New York: McKay.

DEPARTMENT_FOR_EDUCATION. 2013b. The national curriculum in England, Framework document. Available: www.education.gov.uk/nationalcurriculum [Accessed 13-08-2013].

NATIONAL_RESEARCH_COUNCIL 2010. Report of a Workshop on the Scope and Nature of Computational Thinking. The National Academies Press.

PÓLYA, G. 1985. *How To Solve It, 2nd ed,* London, Penguin.

SELBY, C. & WOOLLARD, J. 2013. Computational Thinking: The Developing Definition Available: http://eprints.soton.ac.uk/356481/ [Accessed 07-02-15].

SELBY, C. 2014. *How can the teaching of programming be used to enhance computational thinking?* Doctoral, University of Southampton.

WING, J. 2006. Computational thinking. *Commun. ACM,* 49**,** 33-35.
WING, J. 2008. Computational thinking and thinking about computing. *Philosophical Transactions of The Royal Society A,* 366**,** 3717-3725.
WING, J. 2011. Research Notebook: Computational Thinking - What and Why? *The Link.* Pittsburgh, PA: Carneige Mellon.

Dr Cynthia C Selby (c.c.selby88@gmail.com)
Bay House School and Sixth Form College (Computing Teacher)
Southampton Education School, University of Southampton (ITE Tutor)