Assessing computational thinking

Daniel Duckworth
Australian Council for Educational Research

Daniel Duckworth is a Research Fellow in the Assessment and Psychometrics Research Division at ACER, where he has worked from 2007–2013 and from 2016–present. He has worked on a variety of projects such as the 2009 International Civic and Citizenship Education Study, the Programme for the International Assessment of Adult Competencies, the National Assessment Program ICT Literacy (2008, 2011, 2017), the International Computer and Information Literacy Study (ICILS) in 2013 and 2018, and Critical and Creative Thinking assessments for the Victorian Curriculum and Assessment Authority (2015). Daniel specialises in computer and information literacy (CIL) and is a co-author of the ICILS 2018 Assessment Framework, ICILS 2018 Public Report and NAP ICT Literacy 2017 Public Report. *Daniel's work extends beyond the assessment of CIL and includes research on the way natural language processing algorithms can be used to analyse assessment response data. Daniel is also part of the General Capabilities Assessment (GCA) development team, a project of the Centre for Assessment Reform and Innovation. His work on the GCA has contributed to the conceptualisation of the critical thinking, creative thinking and collaboration frameworks and the operationalisation of those frameworks in assessment instruments that are being trialled in schools in Queensland and Victoria.*

Abstract

This paper provides some context for the role of computation thinking (CT) in the Australian Curriculum, an abridged literature review of CT as a problem-solving framework from the ICILS 2018 assessment framework and some examples of how CT has been used to solve real-world problems. Finally, this paper presents ways to teach and assess CT.

Assessing computational thinking

Computational thinking and the Australian Curriculum

The National Assessment Program (NAP) began as an initiative of ministers of education in Australia to monitor outcomes of schooling specified in the 1999 Adelaide Declaration on National Goals for Schooling in the 21st Century (Adelaide Declaration). The NAP was established to measure student achievement and to report this against key performance measures in relation to the national goals, using nationally comparable data in each of literacy, numeracy, science, and information and communication technologies (ICT). In 2008, the Adelaide Declaration was superseded by the Melbourne Declaration on the Educational Goals for Young Australians (Melbourne Declaration).

In 2010, the Australian Curriculum and Assessment Reporting Authority (ACARA) released the Australian Curriculum, which organised the curriculum into learning areas. General capabilities were introduced to the Australian Curriculum in 2012, including the ICT capability, and in 2014 the technologies F–10 learning area was added. This draws together the subjects of design technologies and digital technologies. In the Australian Curriculum, subject content includes descriptions of what students are expected to learn. These include knowledge, understanding and skills, described at a year level or band of years. The content descriptions are accompanied by content elaborations that give teachers ideas about how they might teach the content. Within the digital technologies subject content, the curriculum refers to CT and is defined as:

> A problem-solving method that involves various techniques and strategies that can be implemented by digital systems. Techniques and strategies may include organising data logically, breaking down problems into parts, defining abstract concepts and designing and using algorithms, patterns and models (ACARA, 2014).

From Foundation to Year 2, students develop skills in CT to understand digital systems to organise, manipulate and present data and begin to conceptualise algorithms as a sequence of steps for carrying out instructions. One example given in the content descriptions is identifying the significant steps of making a sandwich. At the most basic level a student might simply provide the instruction, 'make a sandwich'. However, as students develop skills in CT they are able to differentiate between a process and a set of instructions required to complete a process by identifying significant steps such as 'put the bread flat on the table', 'open the jar', 'put the knife in the jar' etc. Sample portfolios accompany the content descriptions that showcase student work that is satisfactory, above satisfactory or below satisfactory. One such example at Foundation to Year 2 is a video demonstration of students who have developed a sequence of steps to program a Bee-Bot® (a small physical robot) to navigate an 8 × 10 grid. Another example at Years 5 and 6 is a video interview with a student who describes a computer network. The student describes the steps involved in sharing information between computers, including the need for a specialised computer (a server or DNS) that distributes unique addresses to other computers (clients) in a network. The student also contextualises this abstract digital system by describing the way it helps her collaborate with her classmates by using a shared folder to share files.

Computational thinking as conceptualised by the ICILS 2018

One aspect of learning to use computer technologies focuses on learning the foundational principles of computing. This aspect was evident in the early stages of the introduction of computers into classrooms in terms of arguments that saw the links between 'programming' and problem-solving as important for educational development (Papert, 1980). In the 1980s, the Logo language used commands to move a cursor or robot (a turtle) on a screen and line graphics. Many educational approaches closely linked to constructionism and oriented to cognitive development were based on Logo (Maddux & Johnson, 1997; McDougall, Murnane, & Wills, 2014; Tatnall & Davey, 2014).

Since those early developments, visual programming languages (where programs are created by manipulating program elements, or blocks, graphically) for children have emerged in addition to text-based programming languages. Scratch is an example of a visual programming language in which students use simple blocks of code to develop projects (Ortiz-Colon & Marato Romo, 2016). Scratch has a potential role in helping cognitive and meta-cognitive development, as well as providing opportunities for introducing the principles of computing in a practical and productive way.

Shute, Sun & Asbell-Clarke (2017, p. 142) argued that CT is required to solve problems algorithmically (with or without the assistance of computers) by applying solutions that are reusable in different contexts. They elaborated that CT is 'a way of thinking and acting, which can be exhibited through the use of particular skills, which then can become the basis for performance-based assessments of CT skills.' They suggested that CT involves six elements: decomposition, abstraction, algorithm design, debugging, iteration and generalisation. The ICILS 2018 assessment framework defines CT as 'an individual's ability to recognize aspects of real-world problems which are appropriate for computational formulation and to evaluate and develop algorithmic solutions to those problems so that the solutions could be operationalized with a computer' (Fraillon et al., 2019).

Solving real-world problems with computational thinking

Numerous real-world problems have been solved with computational thinking. In 1936, Alan Turing invented the automatic machine (more commonly known as the Turing machine), a mathematical model of computation. Global communications via the internet were enabled by the development of the TCP/IP protocol by the Defense Advanced Research Projects Agency (DARPA) in the late 1960s (Cerf & Edward, 1983).

The Byzantine generals' problem (Lamport, Shostak, & Pease, 1982) was solved by combing Merkle Trees and cryptography to create blockchain technology (an immutable and distributed ledger), further enabling censorship resistant applications and decentralised cryptocurrencies such as Bitcoin (Nakamoto, 2008). Computer vision has surpassed human performance (He, Zhang, Ren, & Sun, 2015) to enable autonomous vehicles assisted by cameras and is the result of deep learning algorithms that utilise the perceptron (Minsky & Papert, 1969), stochastic gradient descent (Bottou, 2004) and backpropogation (Hecht-Nielsen, 1992).

Examples of CT curriculum and assessment

CT does not necessarily involve developing or implementing a formal computer code (Barr, Harrison, & Conery, 2011). Wing (2006, p. 33) argued that the concept of CT is applicable to all individuals rather than just computer scientists. Goode and Chapman (2013) developed the curriculum resource *Exploring Computer Science* (ECS) to help elaborate the meaning of CT. This curriculum package includes resources, lesson plans, and professional development for teachers. Its focus is on 'conceptual ideas of computing', but it includes consideration of 'computational practices of algorithm development, problem-solving and programming' (Goode & Chapman, 2013, p. 5) in contexts of real-life problems (using the Scratch programming tools).

ECS is linked to the Principled Assessment of Computational Thinking (PACT; see https://pact.sri.com/index.html), which is concerned with the assessment of secondary computer science outcomes (Rutstein, Snow, & Bienkowski, 2014). This approach involves designing 'assessment tasks to measure important knowledge and practices by specifying chains of evidence that can be traced from what students do' (Bienkowski, Rutstein, & Snow 2015, p. 2; see also Grover, Pea, & Cooper, 2015; Grover, 2017). PACT is based on design patterns for major CT practices and involves judging the quality of the instructions (or coding steps) that have been assembled.

There have also been other approaches to the assessment of CT. Chen et al. (2017) developed an instrument for primary school students to assess CT that was based on coding in robotics and reasoning of everyday events and linked to a 'robotics curriculum'. Zhong, Wang, Chen, & Li (2016) developed a three-dimensional assessment framework based on the concepts of directionality, openness and process. The assessment included three pairs of tasks that were based on a three-dimensional programming language: i) closed forward tasks and closed reverse tasks, ii) semi-open forward tasks and semi-open reverse tasks, and iii) open tasks with a creative design report and open tasks without a creative design report. Students' codes were assessed by the research team based on sets of rubrics reflecting elements of CT. They concluded that semi-open tasks were more discriminating than others, but that a combination of tasks was needed to assess the various elements of CT. What appear to be common elements in assessments of CT are the capturing of instructions developed by students (almost always using a computer environment) and the judging of the quality of those instructions against a set of criteria reflecting aspects of CT.

Visual coding approaches are of relevance for assessing CT, as they focus on the algorithmic logic underpinning coding across all coding tasks. A visual coding environment is also considered to be accessible to novice users and translatable (code block names can be translated into the target languages) while eliminating the confounding effect of keyboard errors because no typing of code is involved. Assessments of CT are typically set in computer environments because those facilitate the capturing of the data that reflect the steps in problem-solving. These steps usually involve developing or assembling instructions (often including blocks of code) that are necessary to accomplish a task (Brennan & Resnick, 2013).

The ICILS 2018 included two assessment modules that assessed two strands of CT: one on conceptualising problems and the other on operationalising solutions (Fraillon et al., 2019). The tasks in the CT module focused on conceptualising problems related to planning aspects of a program to operate a driverless bus. This included visual representation of real-world situations in ways to support the development of computer programs to execute automated solutions. Examples of these are path diagrams, flow charts, and decision trees. Further tasks related to the use of simulations to collect data and draw conclusions about real-world situations that can inform planning the development of a computer program. In the operationalising solutions module, students worked within a simple visual coding environment to create, test and debug code (blocks of code that have some specified and some configurable functions) to control the actions of a drone used in a farming context. In this module, the tasks were incrementally more difficult as the students advanced through the assessment. The difficulties of the tasks related to the variety of code functions that are available and the complexity of the sequence of actions required by the drone for completion of the task objectives.

Scoring students' responses to a task involved capturing how many of the task objectives were completed, whether any irrelevant actions were performed by the drone and the efficiency with which the objectives were completed. Students that could develop an algorithm that completed exactly all the objectives with the minimum necessary code blocks received the highest score. Students that used more code blocks than necessary, completed some of the objectives or included irrelevant actions for the drone received partial credit.

References

Australian Curriculum and Reporting Authority (ACARA) (2012). *The Australian Curriculum: technologies, key ideas*. Sydney, NSW: ACARA. Retrieved from https://www.australiancurriculum.edu.au/f-10-curriculum/technologies/key-ideas/

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning and Leading with Technology, 38*(6), 20–23.

Bienkowski, M., Rutstein, D., & Snow, E. (2015). *Computer science concepts in the next generation science standards*. Paper presented at the 2015 annual meeting of the American Educational Research Association (AERA), Chicago, IL. Retrieved from https://www.aera.net/Publications/Online-Paper-Repository/AERA-Online-Paper-Repository

Bottou, L. (2004). Stochastic learning, *Advanced Lectures on Machine Learning*, LNAI, 3176. doi: 10.1007/b100712

Brennan, K., & Resnick, M. (2013). Imagining, creating, playing, sharing, reflecting: How online community supports young people as designers of interactive media. In C. Moza & N. Lavigne (Eds.), *Emerging technologies for the classroom: A learning sciences perspective* (pp. 253–268). New York, NY: Springer. Retrieved from https://doi.org/10.1007/978-1-4614-4696-5

Cerf, V. G., & Edward, C. (1983). The DoD internet architecture model. *Computer Networks, 7*(5), 307–318. doi.org/10.1016/0376-5075(83)90042-9

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, *109*, 162–175. Retrieved from https://doi.org/10.1016/j.compedu.2017.03.001

Fraillon, J., Ainley, J., Schulz, W., Ainley, J., Duckworth, D., & Friedman, T. (2019). *International Computer and Information Literacy Study 2018: Assessment framework*. Cham, Switzerland: Springer.

Goode, J., & Chapman, G. (2013). *Exploring computer science*. Menlo Park, CA: Stanford Research International (SRI).

Grover, S. (2017). *Designing programming tasks for measuring computational thinking*. Paper presented at the Annual Meeting of the American Educational Research Association, San Antonio, TX.

Grover, S., Pea, R., & Cooper, S. (2015). *Systems of assessments for deeper learning of computational thinking in K–12*. Paper presented at the Annual Meeting of the American Educational Research Association, Chicago, IL.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1026–1034).

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network, based on *Proceedings of the International Joint Conference on Neural Networks 1*, 593–611, June 1989, Academic Press, 65–93.

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals' problem. *ACM Transactions on Programming Languages and Systems, 4*(3), 382–401.

Maddux, C. D., & Johnson, D. L. (1997). Logo: A retrospective. *Computers in the Schools Monographs/Separates, 14*(1–2). New York, NY: CRC Press.

McDougall, A., Murnane, J., & Wills, S. (2014). The education programming language Logo: Its nature and its use in Australia. In A. Tatnall & B. Davey (Eds.), *Reflections on the history of computers in education: Early use of computers and teaching about computing in schools*. IFIP Advances in Information and Communication Technology Vol. 424. Berlin, Germany: Springer.

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system*. Retrieved from https://bitcoin.org/bitcoin.pdf

Ortiz-Colon, A. M., & Marato Romo, J. L. (2016). Teaching with Scratch in compulsory secondary education. *International Journal of Emerging Technologies in Learning, 11*(2), 67–70. Retrieved from https://doi.org/10.3991/ijet.v11i02.5094.

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

Rutstein, D. W., Snow, E. B., & Bienkowski, M. (2014, April). *Computational thinking practices: Analyzing and modeling a critical domain in computer science education*. Paper presented at the annual meeting of the American Educational Research Association, Philadelphia, PA.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142–158. Retrieved from https://doi.org/10.1016/j.edurev.2017.09.003

Tatnall, A. & Davey, B. (Eds.), *Reflections on the history of computers in education: Early use of computers and teaching about computing in schools*. IFIP Advances in Information and Communication Technology Vol. 424. Berlin, Germany: Springer.

Turing, A.M. (1936). *On computable numbers, with an application to the Entscheidungs problem*. *Proceedings of the London Mathematical Society*. *42*(1), 230–265.

Wing, J. M. (2006). *Computational thinking. Communications of the ACM*, *49*, 33–35. doi:10.1145/1118178.1118215

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research, 53*(4), 562–590. Retrieved from https://journals.sagepub.com/doi/pdf/10.1177/0735633115608444