CHAPTER 3

Computational thinking framework

3.1 Overview

The ICILS CIL construct was established and measured in response to the pervasive belief in the value of CIL-related competencies for participation in the 21st century. At the same time as ICILS 2013 was being developed, there was the beginning of a resurgence of interest from researchers, educators, and policymakers in the importance of CT in education (Voogt et al. 2015). The inclusion of CT as an international option in ICILS 2018 was, in part, a response to a growing belief in the importance of computer science and computational thinking in schooling and efforts across countries to expand students' access to these areas of learning (Yadav et al. 2018).

While CT has been recognized "since the beginning of the computing field in the 1940s" (Denning 2017, p 34), many researchers refer to the work of Papert in the late 20th century (Papert 1980, 1991; Shute et al. 2017; Voogt et al. 2015) as a touchstone for CT research. More recently, Wing's (2006) article on CT has been regarded by researchers as the catalyst, or at least as a common point of reference, for the re-emergence of interest in CT (see, for example, Barr and Stephenson 2011; Bower et al. 2017; Grover and Pea 2013; Shute et al. 2017; Voogt et al. 2015). In this article, Wing characterized CT as "a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" (Wing 2006, p. 33). However, despite the high level of interest in CT and the rapid increase in curriculum and educational resources, along with research concerned with CT, there has been confusion about its definition (Denning 2017; Grover and Pea 2013; Selby and Woollard 2013). This confusion is partly attributable the broad range of perspectives on CT that abound. For example, the National Academic Press reported on a 2009 workshop on the nature of CT that cited the following perspectives on CT (National Research Council 2010, pp. 11–12):

- CT is "closely related to, if not the same as…procedural thinking…that includes developing, testing, and debugging procedures"
- CT is about "expanding human mental capacities through abstract tools that help manage complexity and allow for automation of tasks"
- CT is primarily about processes and is a subset of computer science
- CT is "the use of computation-related symbol systems (semiotic systems) to articulate explicit knowledge and to objectify tacit knowledge to manifest such knowledge in concrete computational forms"
- CT is about "rigorous analyses and procedures for accomplishing a defined task"
- CT "is a bridge between science and engineering—a meta-science about studying ways or methods of thinking that are applicable to different disciplines"
- CT is "what humans do as they approach the world [that is, their framing, paradigm, philosophy, or language], considering processes, manipulating digital representations (and [meta] models)," and hence all humans engage in computational thinking to some extent already in their daily lives"
- CT "plays a role in the manipulation of software in support of problem solving"
- "What makes computational thinking especially relevant is that computers can execute our 'computational thoughts'."

J. Fraillon et al., *IEA International Computer and Information Literacy Study 2018 Assessment Framework*,
https://doi.org/10.1007/978-3-030-19389-8_3

The range of different perspectives listed above exemplify some of the tensions that exist in approaches to CT. These tensions are associated with identifying where CT should be located on a spectrum of capabilities that, at one end, are characterized by algorithmic procedural thinking associated with computer programming and, at the other end, are described by a broader suite of problem solving capabilities and dispositions (see, for example, Barr et al. 2011; Barr and Stephenson 2011; Voogt et al. 2015). In reflecting on attempts to define CT, Voogt et al. (2015, p. 718) described the tension between "thinking of the 'core' qualities of CT versus certain more 'peripheral' qualities").

For ICILS, the definition and explication of CT, as for CIL, must be considered in the context of the ICILS assessment parameters. In this case the assessment of CT must be:

- Applicable to students in their eighth year of schooling
- Applicable across a broad range of country and curriculum contexts
- Complementary to the ICILS assessment of CIL
- Minimally overlapping with assessment content in other curriculum areas (such as in mathematics or science).

With these parameters in mind, the conceptualization of CT in ICILS is that CT combines the competencies associated with (a) framing solutions to real-world problems in a way that these solutions could be executed by computers; and then (b) implementing and testing solutions using the procedural algorithmic reasoning that underpins programming.

3.2 Defining computational thinking

In a review of CT literature, Selby and Woollard (2013) identified three consistently shared constituent components of CT: (a) a *thought process* (a way of thinking about computing); (b) *abstraction* (describing the common underlying properties and functionality of a set of entities); and (c) *decomposition* (breaking a complex problem into well-defined parts). Voogt et al. (2015, p. 720) suggested that many definitions of CT focus on the "skills, habits and dispositions needed to solve complex problems with the help of computing."

Following is a selection of definitions and descriptions of CT that have been used to inform the development of the definition of CT established for use in ICILS.

(1) "Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (Wing 2011, as cited by Grover and Pea 2013, p. 39).

(2) "We consider computational thinking to be thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms" (Aho 2012, p. 832).

(3) "It [computational thinking] is a cognitive or thought process that reflects:
- the ability to think in abstractions,
- the ability to think in terms of decomposition,
- the ability to think algorithmically,
- the ability to think in terms of evaluations, and
- the ability to think in generalizations" (Selby and Woollard 2013, p. 5).

(4) "Computational thinking describes the processes and approaches we draw on when thinking about how a computer can help us to solve complex problems and create systems" (Digital Technologies Hub 2018).

(5) "Computational thinking is the process of recognizing aspects of computation in the world that surrounds us, and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes" (Royal Society 2012, p. 29).

(6) "Computational thinking is a problem-solving process that includes:
- Formulating problems in a way that enables us to use a computer and other tools to help solve them
- Logically organizing and analyzing data
- Representing data through abstractions, such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
- Generalizing and transferring this problem-solving process to a wide variety of problems" (Barr et al. 2011, p. 21).

(7) "Computational thinking is a term often used to describe the ability to think with the computer-as-tool" (Berland and Wilensky 2015, p. 630).

Common to these definitions of CT is the idea that CT is regarded as a form of problem solving in which the problems and solutions are conceptualized so that algorithmic, procedural (step-by-step) solutions can be established and implemented using a computer. These characteristics are consistent with the ICILS conceptualization of CT as focusing on problem solving to generate computer-based solutions. While it can reasonably be argued that the core of this conceptualization of CT may be applied to other learning domains, the ICILS test of CT does not include measurement of cross-domain applications of CT.

The definition of CT established within the context of ICILS is:

> Computational thinking refers to 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.

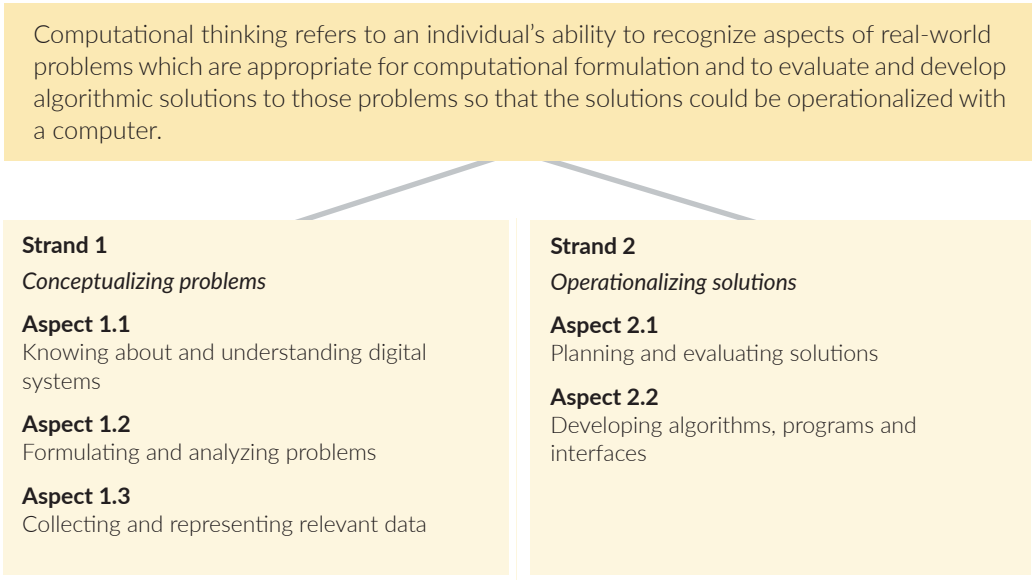
3.3 Structure of the ICILS 2018 computational thinking construct

The CT construct includes the following elements:

- Strand: This refers to the overarching conceptual category for framing the skills and knowledge addressed by the CT instruments.
- Aspect: This refers to the specific content category within a strand.

The CT construct comprises two strands. One strand contains three aspects and the other comprises two aspects (summarized in Figure 3.1 and described in detail in section 3.4). The aspects encompass the set of knowledge, skills, and understandings held in common across the range of definitions of CT as discussed previously.

Figure 3.1: ICILS 2018 CT framework



The structure shown above does not presuppose a sub-dimensional structure of the CT construct. The primary purpose of describing CT using this structure is to organize the CT content in a way that allows readers to clearly see the different related aspects of CT and to support the auditing of the CT instruments against the full breadth of content in the CT construct. We hypothesize that CT will form a single measurement dimension. However, analyses of the dimensionality of the ICILS 2018 CT data will be used to determine whether CT is reported as a single or as multiple dimensions.

3.4 Strands and aspects of computational thinking

3.4.1 Strand 1: Conceptualizing problems

Conceptualizing problems acknowledges that before solutions can be developed, problems must first be understood and framed in a way that allows algorithmic or systems thinking to assist in the process of developing solutions. This strand comprises three aspects:

- Knowing about and understanding digital systems
- Formulating and analyzing problems
- Collecting and representing relevant data.

Aspect 1.1: Knowing about and understanding digital systems

Knowing about and understanding digital systems refers to a person's ability to identify and describe the properties of systems by observing the interaction of the components within a system.

Systems thinking is used when individuals conceptualize the use of computers to solve real-world problems, which is fundamental to computational thinking.

At a declarative level a person can describe rules and constraints that govern a sequence of actions and events, or they are able to provide a prediction for why a procedure is not working correctly by observing the conditions of the error. For example, imagine a student was required to design a game. The student would first need to specify the initial state of the game, the winning condition of the game and the parameters of the permissible actions, and sequence of actions within the game.

At a procedural level, a person can monitor a system in operation, make use of tools that help to describe a system (such as tree diagrams or flow charts), and observe and describe outcomes of a processes operating within a system. These procedural skills are based on a conceptual understanding of fundamental operations such as iteration, looping, and conditional branching, and the outcomes of varying the sequence in which they are executed (control flow). An understanding of these operations can enhance a person's understanding of both the digital world and the physical world; and it can therefore assist in solving problems. With reference to the example of a student designing a game, at the procedural level the student might initiate and adjudicate the game play. The student would need to monitor the players' actions and the consequent outcomes according to the specified rules and conditions of the game. In doing this, the student may observe problems with the game, such as unresolvable or ambiguous situations and be able to modify the game parameters accordingly. It is not always necessary that the game be created as a computer application, as digital systems thinking can also be applied to non-digital systems. In the context of ICILS, digital systems thinking could be applied to describe the actions of a physical system (such as filling a glass with water from a tap) in such a way that these actions could later be controlled by a computer program.

The following examples reflect tasks that provide evidence of an individual's ability to know about and understand digital systems:

- Exploring a system to describe rules about its behavior
- Operating a system to produce relevant data for analysis
- Identifying opportunities for efficiency and automation
- Explaining why simulations help to solve problems.

Aspect 1.2: Formulating and analyzing problems

Formulating problems entails the decomposition of a problem into smaller manageable parts and specifying and systematizing the characteristics of the task so that a computational solution can be developed (possibly with the aid of a computer or other digital device). Analyzing consists of making connections between the properties of, and solutions to, previously experienced and new problems to establish a conceptual framework to underpin the process of breaking down a large problem into a set of smaller, more manageable parts.

The following examples reflect tasks that provide evidence of an individual's ability to formulate and analyze problems:

- Breaking down a complex task into smaller, more manageable parts
- Creating a self-contained sub-task that could potentially be applied more than once
- Exploring the connection between the whole and the parts.

Aspect 1.3: Collecting and representing relevant data

In order to make effective judgements about problem solving within systems it is necessary to collect and make sense of data from the system. The process of collecting and representing data effectively is underpinned by knowledge and understanding of the characteristics of the data and of the mechanisms available to collect, organize, and represent these data for analysis. This could involve creating or using a simulation of a complex system to produce data that may show patterns or characteristics of behavior that are otherwise not clear when viewed from an abstract system level.

The following examples reflect tasks that provide evidence of an individual's ability to collect and represent data:

- Identifying an abstracted representation of map directions
- Using a route simulation tool to store data
- Displaying data to help draw conclusions and inform planning
- Using simulation tool to collect data and evaluate outcomes.

3.4.2 Strand 2: Operationalizing solutions

Operationalizing solutions comprises the processes associated with creating, implementing and evaluating computer-based system responses to real-world problems. It includes the iterative processes of planning for, implementing, testing, and evaluating algorithmic solutions (as the potential bases for programming) to real-world problems. The strand includes an understanding of the needs of users and their likely interaction with the system under development. The strand comprises two aspects:

- Planning and evaluating solutions
- Developing algorithms, programs and interfaces.

Aspect 2.1: Planning and evaluating solutions

Planning solutions refers to the process of establishing the parameters of a system, including the development of functional specifications or requirements relating to the needs of users and desired outcomes and with a view to designing and implementing the key features of a solution. Evaluating solutions refers to the ability to make critical judgements about the quality of computational artefacts (such as algorithms, code, programs, user interface designs, or systems) against criteria based on a given model of standards and efficiency. These two processes are combined in a single aspect because they are iteratively connected to the process of developing algorithms and programs. While the process of developing algorithms may begin with planning and end with evaluation, throughout the process there is a constant iteration between planning, implementation, evaluation, and revised planning (or resolution). Typically, there is a broad array of potential solutions to any given problem and, consequently, it is important to be able to plan and evaluate solutions from a range of perspectives, and to understand the advantages, disadvantages, and effects on stakeholders of alternative solutions.

The following examples reflect tasks that provide evidence of an individual's ability to plan and evaluate computational solutions:

- Identifying the starting point for an algorithmic solution to a problem by reflecting on solutions to similar problems
- Designing components of a solution taking into account the limitations of the system and the needs of users
- Testing a solution method against a known outcome and adjusting it as necessary
- Comparing the relative advantages and disadvantages of a solution against alternative solutions
- Locating a faulty step in an algorithm
- Describing solutions and explaining why they are the best solution among many
- Implementing and managing strategies to test the efficacy of a solution (such as user testing).

Aspect 2.2: Developing algorithms, programs and interfaces

ICILS 2018 does not presuppose that students are familiar with the syntax and features of any particular coding language. This aspect focuses on the logical reasoning that underpins the development of algorithms (and code) to solve problems. It can involve developing or implementing an algorithm (systematically describing the steps or rules required to accomplish a task) and also automating the algorithm, typically using computer code in a way that can be implemented without the need for students to learn the syntax or features of a specific coding language. Creating an interface relates to the intersection between users and the system. This may relate to development of the user interface elements in an application including implementation of specifications for dynamic interfaces that respond to user input.

The following examples reflect tasks that provide evidence of an individual's ability to develop algorithms, programs, and interfaces include the following:

- Modifying an existing algorithm for a new purpose
- Adapting visual directions into instructions for a computer
- Creating visual representations of instructions for a computer
- Creating a simple algorithm
- Using a new statement in a simple algorithm
- Creating an algorithm that combines simple command statements with a repeat or conditional statement
- Correcting a specified step in an algorithm.