Design Science

Design thinking and computational thinking: a dual process model for addressing design problems

Nick Kelly[1] and John S. Gero[2]

1 Queensland University of Technology, Brisbane, QLD, Australia
2 University of North Carolina at Charlotte, Charlotte, NC, USA

Abstract

This paper proposes a relationship between design thinking and computational thinking. It describes design thinking and computational thinking as two prominent ways of understanding how people address design problems. It suggests that, currently, each of design thinking and computational thinking is defined and theorized in isolation from the other. A two-dimensional ontological space of the ways that people think in addressing problems is proposed, based on the orientation of the thinker towards problem and solution generality/specificity. Placement of design thinking and computational thinking within this space and discussion of their relationship leads to the suggestion of a dual process model for addressing design problems. It suggests that, in this model, design thinking and computational thinking are processes that are ontological mirror images of each other, and are the two processes by which thinkers address problems. Thinkers can move fluently between the two. The paper makes a contribution towards the theoretical foundations of design thinking and proposes questions about how design thinking and computational thinking might be both investigated and taught as constituent parts of a dual process.

Key words: design thinking, computational thinking, design framing, dual process

Received 02 August 2020
Revised 18 March 2021
Accepted 18 March 2021

Corresponding author
N. Kelly
nick.kelly@qut.edu.au

Des. Sci., vol. 7, e8
journals.cambridge.org/dsj
DOI: 10.1017/dsj.2021.7

1. Introduction

The term *design thinking* is widely discussed in design literature and has its foundations in research on how designers know, act and think (Cross 2011; Dorst 2011). A second form of 'thinking', called *computational thinking*, has its foundations in research on how computer scientists know, act and think (Wing 2006; Shute *et al.* 2017). Each form of 'thinking' has found widespread popularity, as indicated by the international adoption of both terms within formal education systems (Grover & Pea 2013; Koh *et al.* 2015), and, in the case of design thinking, within business and government (Brown 2008).

These two forms of thinking are distinct from other kinds of thinking in two ways. First, each was inspired by a body of knowledge and expertise – design and computer science, respectively – that was recognized as valuable, and each can be understood as a transfer of a way of thinking from a particular tradition to something that is useful far more broadly. This is distinct from other popularized forms of thought, such as *critical thinking* (Ennis 1993) or *creative thinking* (De Bono 1995), which have long histories (at least to Greek times) and a clear breadth of application.

Second, design thinking and computational thinking are the only two forms of thinking to gain prominence since the turn of the 21st century.

The relationship between design thinking and computational thinking matters, because these forms of thinking are rapidly being adopted by schools and universities as useful ways of thinking for addressing problems; yet each is commonly taught in isolation from the other. Even basic questions about the relationship between the two kinds of thinking have no established answers: are design thinking and computational thinking discrete, orthogonal or overlapping notions? Consider, for example, the case of a software engineer trying to understand a client's needs. Such activity might readily be understood through the lens of either design thinking or computational thinking. In another example of the blurred boundaries between these terms, Jeanette Wing has suggested that computational thinking is 'a creative process' that 'relies on human ingenuity, flashes of insight and taste in design' (Wing 2019, p. 5) – cognitive abilities that are typically taken to be markers of design thinking, yet they equally seem to be applicable to some definitions of computational thinking.

This paper aims to compare and contrast these two forms of thinking to provide some clarity about the relationship between them, which we conclude to be as dual processes used in addressing design problems. The significance of the work is to position design thinking in relation to another form of thinking that seems, intuitively, to be counterposed to it, contributing to the theoretical foundations of design thinking. The work is ontological in that it aims to specify representational terms that describe a domain (Gruber 1993). The paper highlights the significance of *framing* for both design thinking and computational thinking and contrasts the class of outcomes that each type of thinking produces. This leads to a proposal for two orthogonal variables that frame metacognitive approaches to design problems. In the Discussion section, we suggest that this space has relevance for how design thinking and computational thinking are taught, both inside and outside of formal education. In the rest of the paper, we will refer to the person engaged in either design thinking or computational thinking as *the thinker*, for consistency. Throughout, we will refer to 'problems' and 'solutions' that are arrived at through design thinking and computational thinking; again, this is done for consistency, while recognizing that much design activity does not have clearly defined problems and that design solutions do not necessarily 'solve' problems but address them in a designerly way.

2. Design thinking

Design thinking has become an overloaded and ambiguous term, but has its roots in the scientific study of design cognition and design methods. Design thinking is considered here to be the knowledge that has been developed relating to how people reason when engaging with design problems (Lawson 2006; Cross 2011; Dorst 2011), also described as 'designerly ways of knowing, thinking and acting' (Cross 2001) and as *designerly thinking* (Johansson-Sköldberg *et al.* 2013). There are currently multiple discourses using the term *design thinking*, as is described in two papers tracing its origins (Kimbell 2011; Johansson-Sköldberg *et al.* 2013). Johansson-Sköldberg *et al.* (2013) trace the development of both a *designerly* and a *management* discourse on design thinking. Within the designerly discourse, Kimbell (2011) describes different accounts of design thinking as a cognitive style, as a general theory of design, and as an organizational resource. In a subsequent paper, Kimbell (2012) critiques the way that some of these accounts: (1) create a dualism between thinking and acting/knowing; (2) fail to recognize the (socially and environmentally) situated nature of design and (3) essentialize what it is that designers do, ignoring the diversity in ways of thinking and knowing. The definition of design thinking adopted here recognizes the interrelationship between design science and practice (e.g., the scholarship of Cross and Lawson). It also recognizes that design is a situated phenomenon (e.g., the scholarship of Gero and Dorst). Finally, although people who undertake design activity think in diverse ways, there may be commonalities between their ways of thinking (where that word assumes a situated understanding of cognition) that can be understood through science.

Two further counts of disambiguation are required. First, design thinking is occasionally used in the public sphere as shorthand for the suite of tools, skills and mindsets that are well-suited for teaching nondesigners how to approach complex problems in a designerly way (e.g., Goldman *et al.* 2012). This notion of design thinking, which Kimbell (2011) refers to as 'design thinking as an organizational resource', is not what we are referring to here, as it focusses upon some of the useful outcomes from studying design cognition rather than the cognition itself. Second, claims about design thinking are sometimes made on the basis that they were developed by studying *what designers do*. It is true that seminal studies in design thinking were made by investigating the practices and habits of design professionals; yet it does not follow that the science of design thinking only applies to those who practice in professions that bear the label 'design'. Rather, it applies to anyone who is engaged in design activity.

These two terms of *design problems* and *design thinking* are related. Gero (1990) defines design as 'a goal-oriented, constrained, decision-making, exploration and learning activity which operates within a context which depends on the designer's perception of the context'. Design problems are then those problems which are both *goal-oriented and constrained* and which *depend upon a designer's perception of the context of the problem.* This allows for design problems to be distinguished from problems which may not be goal-oriented and constrained, such as some artistic problems involving self-expression. It also allows them to be distinguished from problems in which all variables are known at the outset, such as engineering optimization problems or some mathematical problems, in which the designer's perception of the context of the problem is irrelevant to the potential solutions. It follows that design problems occur in a wide range of contexts; they are only related to the formally recognized design professions in that professional designers tend to frequently engage in working with design problems. Design thinking is then understood as *the knowledge that has been developed relating to how people reason when engaging with design problems*, where this notion of reasoning is understood as situated, embodied cognition (Clancey 1997; Barsalou 2008) rather than as its impoverished Cartesian conception.

The core of design thinking is an understanding that *a designer creates the frame within which design activity is undertaken* (Schön 1983/2017; Gero 1990; Dorst 2015). This idea of a frame has multiple origins. One can be found in *situated cognition* (where a *frame* is referred to as the *situation*), in the empirically supported argument that knowledge (e.g., concepts) cannot be abstracted from the situation (or frame) within which it is used and learned (Brown *et al.* 1989;

Clancey 1997). Another can be found in Minsky's suggestion that *frames* can be used as a way to specify the top-level of organization in a system for structuring knowledge in the context of artificial intelligence (Minsky 1974). A third can be found in the sociological tradition of Goffman (1974), who described frames as social constructions of reality that are used by individuals to organize experience. All three can be linked to Simon's (1969/2019) discussion of agents with limited capacity of knowledge and reasoning, operating within a complex environment.

The common thread and relevance for design is that while a designer might 'know' a great deal about the world, that knowledge is not stored in discrete, abstract chunks waiting to be 'deployed' during design activity. Rather, when faced with a design problem, a designer cognitively, but unconsciously, constructs a complex assemblage of interrelated knowledge – referred to as the frame. This frame forms the lens through which the object of attention – in this case, the design problem – comes to be understood and within which design actions that might be taken are available.

Many insights into designerly ways of acting and knowing relate to the ability that designers have to frame problems, and the strategies that they have for acting such that the frame changes in a desirable way (sometimes referred to as reframing; e.g., Beckman 2020). One such insight is that designers typically have a conception of the understanding of the problem, a problem space, and a conception of possible solutions, a solution space, that both form a part of the frame. Designers, when observed, appear to be coevolving these two different spaces, changing the understanding of possible designs and possible solutions in parallel and in an interdependent way (Poon & Maher 1997; Dorst & Cross 2001). Another frame-based design phenomenon, identified by Suwa, Gero and Purcell, is that designers make unexpected discoveries within their own external representations (e.g., sketches) that change the trajectory of the design process (Suwa *et al.* 2000). These insights fit with the description of design as something that occurs 'within a context which depends on the designer's perception of the context' (Gero 1990). This has broad implications, for example, that the same design problem approached by the same designer at two different times might be conceived in entirely different ways (Gero 1998; Kelly & Gero 2015).

Problems that are well suited to design thinking are problems in which frames that include a useful solution are not immediately available to a thinker – the solution space needs to be evolved in some way for a solution to become apparent. *Wicked problems* are a good example of the type of problem in which design thinking is needed, in which variables are unknown and the knowledge needed to address the problem is incomplete, such as the kinds of problems encountered in social planning (Rittel & Webber 1973). A problem in which all variables are known at the outset (e.g., solving a tangram puzzle) is not a good candidate for design thinking. Skill in addressing this type of problem requires a capability for working with frames in a particular way. Expert designers have metacognitive skills that enable them to observe the frame that they have created for the design problem and, critically, have access within this frame to conceive of appropriate actions that might expand their understanding of the problem in a useful direction.

Many renowned examples of outstanding design – examples like Jørn Utzon's *Sydney Opera House* and Frank Lloyd Wright's *Fallingwater* – can be recast as tales of exceptional capability in framing and bringing things into or out of the frame. The Sydney Opera House was designed through a competition, where other entrants heeded the rules in their designs (keeping them in the frame), and Jørn Utzon broke them in service to his vision for a design by making the footprint of his design larger than the site. Frank Lloyd Wright introduced many novel ideas in his design of Fallingwater, driven, in part, by his inclusion in the frame the idea that the design of any house should enhance the landscape within which it is situated – leading him to challenge many of the requests of the client including even the location of the house.

In schools of design around the world, whether they are studying architecture, industrial design, engineering, fashion design, web design and so on, students are taught how to think about the *user* of their design, how to think about the *context/site/situation* and how to do *research* about the design. All of these can be considered useful guides for actions to take when confronted with a novel design problem; they all also denote strategies that will implicitly change the frame in a direction that is useful for moving towards a design solution.

The outcome from the design process is design documentation, some kind of communicable representation of the design solution. A notable feature of design solutions is that they tend to be specific to the problem that they were created for – they cannot (typically) be taken and applied directly to other users or other design scenarios. For example, the architectural design of a house needs to be specific to the site where it is located – aspect, topography, landscape, surroundings and history – as well as its inhabitants – the specific needs of the people who will be living in it – to be considered an example of good design. The reuse of the design of a house from one design situation to another in cookie cutter fashion generally results in poor design outcomes. This tendency for design solutions to be specific to a design problem is common across different design disciplines.

3. Computational thinking

The term *computational thinking* has its origins in the recognition that computer science has been the foundation for much innovation and discovery in solving human problems in the modern world, and that there is a broad need for laypeople in society to have the foundational cognitive capabilities that underpin computer science (Wing 2008). The fruits of computational thinking now underpin much of modern life. Because of its importance to society and to economies (National Research Council 2010), there has been widespread uptake of computational thinking as an explicit form of learning in educational systems worldwide. Countries such as Russia, South Africa, New Zealand and Australia have already brought computational thinking into the K–12 curriculum, and there has been a move towards making computational thinking a part of compulsory education in many nations (Grover & Pea 2013; Voogt *et al.* 2015). Computational thinking has its origins in the kinds of thinking that are used by computer scientists but is recognized as a form of thinking that is useful for anybody for solving problems that they may encounter in personal or professional lives.

There is a pattern for review articles about computational thinking to commence by noting its importance, its widespread uptake within education, but also its lack of robust definition (Brennan & Resnick 2012; Grover & Pea 2013; Shute *et al.* 2017). The definitional confusion around computational thinking is summarized by Shute *et al.*, who suggest that '[computational thinking] definitions vary in their operationalization of [computational thinking] in certain studies, and are not particularly generalizable' (Shute *et al.* 2017). There are two trends in attempts to define computational thinking. The first trend defines computational thinking based on the types of reasoning that are used. An example of this is Wing's initial work in suggesting that 'computational thinking involves solving problems, designing systems and understanding human behaviour, by drawing on the concepts fundamental to computer science' (Wing 2006, p. 33). The second trend defines computational thinking based on the types of solutions that it produces. Many papers in the literature refer to a definition developed by Wing and colleagues as '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, p. 21), where the idea is that an information-processing agent can be either human or computational. Here, we will first consider the cognitive markers of computational thinking, and then discuss the types of solutions that it produces. The term *computational thinking* is not used to suggest that humans reason in ways that are similar to computers (e.g., theories subscribing to computationalism); it is shorthand for referring to the problem-solving approaches that computer scientists make use of.

The primary cognitive ability required for computational thinking is abstraction and the competencies that support abstraction. The 'value of abstraction as [computational thinking]'s keystone (distinguishing it from other types of thinking) is undisputed' (Grover & Pea 2013, p. 39), where abstraction is concerned with defining patterns as generalized from specific instances. Abstraction is the type of reasoning that involves moving from specific instances to general patterns, keeping relevant information and discarding irrelevant data. Through abstraction, people 'glean relevant information (and discard irrelevant data) from complex systems to generate patterns and find commonalities among different representations' (Shute *et al.* 2017, p. 144). A cliché in teaching computational thinking is to take the process of making toast, break it down into individual steps and specify an algorithm for how anybody could make toast. This is a useful activity, because it demonstrates an algorithm as a sequence of steps while simultaneously making it clear that it is useful to specify certain things in the algorithm (e.g., putting toast in the toaster) while unhelpful to specify others (e.g., how to coordinate the hand to reach inside a bag of sliced bread). One of the core skills in computational thinking is learning to find the right abstraction.

Problems that require computational thinking are typically highly structured; or rather, the way that the problem is framed requires that a well-structured solution (e.g., an algorithm) be a part of that frame. They are also typically recurrent problems, problems that either occur in many places or recur within the same place. The value in taking the time to solve a problem in such a way that a computer can carry out the process is so that the solution can be deployed in other circumstances with similar problems. For example, businesses often invest significant resources in developing specific software that automates workflow, as they know that eventually the software will pay for itself in time saved or in a competitive advantage. Software developers can create a solution to a very specific problem in their immediate circumstance and then suddenly find that, by solving this problem better than anyone else, they have a global market of people interested in it.

Computational thinking requires thinking about problems in a way that enables solutions to be found – often, but not always, using computers – that apply to many other similar problems and where the steps required to solve those problems can be represented and used in applicable circumstances. For example, the problem of wayfinding within a city can be solved by the development of navigational software combined with GPS-enabled smartphones and data about city geography (e.g., Google Maps). Solving the problem in one city enables the solution to be deployed in another city with only a change of the data being used – the algorithms and hardware can remain unchanged.

The solution to a computational thinking problem is typically a representation of a solution at the appropriate level of abstraction to allow the solution to be applied in other similar circumstances, as typified by an algorithm. In this respect, the solutions provided by computational thinking aim to be generally applicable. However, they also tend to have clearly stated conditions on applicability – such as the kind of variables that a function can accept – and solutions are transferable and repeatable to other situations in which these conditions apply.

4. Design thinking and computational thinking

Despite the popularity of both terms, there is little in the literature that compares design thinking and computational thinking to consider the relationship between the two – perhaps due to the lack of a consensus on definition for both terms. One exception is a brief comparison by Shute *et al.*, who suggest that the main difference lies in the domains where each type of thinking operates, but do not discuss differences in the types of thinking utilized (Shute *et al.* 2017). Additionally, there is theory within the design literature that recognizes the nature of design problems as being inherently embedded within complex systems (e.g., technical, political, economic, ethical etc.), where the embedded systemic way of 'seeing the whole' can be juxtaposed with the need for designers to, at times, see component parts and the relationships between them (Nelson & Stolterman 2003). This view is taken up in the Discussion section of this paper, suggesting that design thinking and computational thinking can be considered archetypes within a spectrum of metacognitive approaches to addressing problems, and where the thinker is may well move around within this spectrum during the one problem.

There are clearly activities that are almost entirely focussed on design thinking – say, the engineering design of a machine – with little or no computational thinking involved; and the inverse also, tasks such as sorting a list which require computational thinking and little or no design thinking. There are also tasks that appear to involve both computational thinking and design thinking, such as a web designer responding to a client's brief. In all three tasks, there is a thinker, a design problem to be addressed, and a solution. On what basis can these three tasks be compared?

We identify two variables that can form the basis for differentiation between design thinking and computational thinking as: (1) the generality/specificity of solutions and (2) the generality/specificity of the frame.

4.1. Specificity of solutions

A synthesis of the descriptions of design thinking and computational thinking suggests that solutions from each type of thinking seem to fall at opposite ends of a continuum. A typical design solution is highly specific to the design problem that the thinker is addressing – specific to the users, the situation, the context etc. It is rare that a design solution from a prior problem can be directly transferred into a

specific
general
solution

Figure 1. The specificity of the solution in relation to the problem as an ontological category with values ranging from specific to general.

new problem without further design thinking being done to adapt it. In contrast to this, most computational thinking solutions can be applied to new problems without any further computational thinking being done – for some scholars, this is a definitional quality of computational thinking.

This leads to the proposal that one variable that explains the difference between design thinking and computational thinking is the *specificity of the solution in relation to the problem to which it pertains.* In design thinking, solutions aim to solve, as coherently as possible, the unique problem being addressed, with little or no thought for the reapplication of that solution in other circumstances. In contrast, in computational thinking, solutions aim to be more general than the problem that they are created to solve. Figure 1 depicts this as an ontological category with values ranging from specific to general.

The continuum can be further understood by considering the two extreme ends. Thinking about solutions is taken to be maximally *specific* if the thinker is entirely concerned with solving *this particular* problem, with no thought for how their thinking or its produce might be used elsewhere. Thinking about solutions is taken to be maximally *general* if the thinker is entirely concerned with how their thinking can be applied to something at a level of abstraction above the present problem. As an ontological category, the claim is not that thinkers are likely to sit at either end of the continuum, but rather that the continuum exists and that all problem-solving sits somewhere between these two ends, and that a thinker will typically move to different places along it while addressing a problem.

4.2. Specificity of framing

A second variable that can be used to explain differences between design thinking and computational thinking relates to the way that the thinker frames the problem. In design thinking, many activities – doing user research, playing with materials, researching theory and cases etc. – are oriented towards an *expansion* of the frame of understanding surrounding the problem. For example, it is usually an indicator of good design if parts of culture relating to the design are given consideration by the thinker and are brought into the frame. In contrast, computational thinking involves abstraction – capturing what is the core relationship between information and processes and abstracting away what can be removed. In the context of computational thinking, culture tends to be abstracted away.

This leads to the proposal that a second ontological category that is used to explain the difference between design thinking and computational thinking is the *specificity of the frame in relation to the problem to which it pertains.* Figure 2 depicts this as an ontological category with values ranging from specific to general. In addressing a design problem, a thinker is likely to move between different parts of this spectrum. In moving between different frames for a problem, a thinker can be seen to have an orientation towards making the frame *more specific* or *more general,* in relation to frames. The suggestion is that design thinking has an orientation towards frames that are more general, although a designer may well use, say, a highly specific frame as one part of the overall design activity.

At one end of this continuum, a thinker can be entirely concerned with cognitive strategies that make the frame more specific to the problem, an overall narrowing of the frame: abstracting away all that can be removed from the thinker's understanding of the problem (e.g., by removing variables and reducing ambiguity). At the other end of this continuum, a thinker can be entirely concerned with cognitive strategies that provide a more general understanding of the problem and its context, an overall expanding of the frame (e.g., adding variables and increasing ambiguity). Again, a thinker likely moves to different points along this continuum as they address a problem.

4.3. An ontology for reasoning about problems

These two categories, of solution specificity and frame specificity, are independent of one another. Given that these categories are orthogonal, a space can be created, which we propose is a useful ontology for how people reason about design problems using design thinking or computational thinking (see Figure 3). It is an ontology in that it specifies representational terms or categories – the two axes – that are useful for specifying a domain of the ways in which humans reason about problems.

specific
general
framing

Figure 2. The specificity of the framing in relation to the problem as an ontological category with values ranging from specific to general.

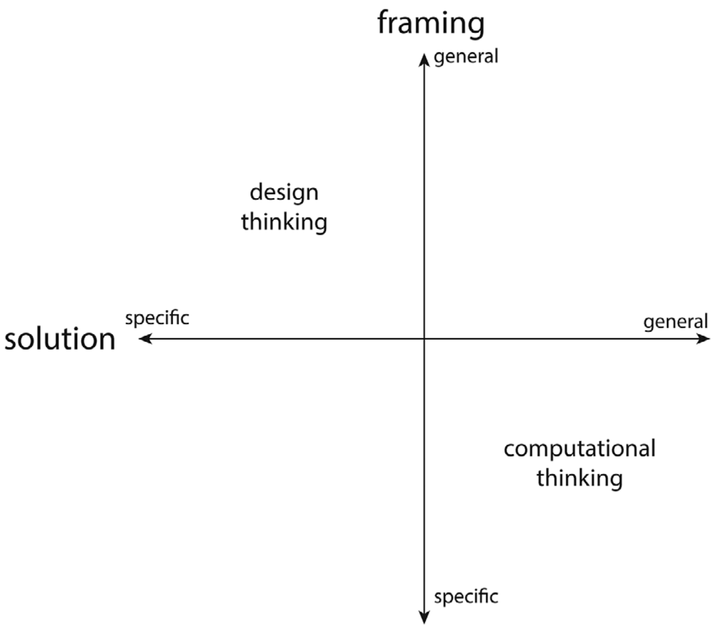

Figure 3. Space created by graphing the two orthogonal ontological categories, with design thinking and computational thinking located in the space.

In the space created by these two axes, the upper left-hand quadrant is a good match with the characterization of design thinking that is provided in the literature. Here, the thinker is aiming for a very specific solution and is getting there by trying to gain a broad understanding of the problem and the context within which it is situated. The lower right-hand quadrant is a good fit with the characterization of computational thinking. The thinker is producing general solutions that can be applied in many places. They are getting there by honing a precise understanding of the problem in a form that has the most helpful abstraction possible.

5. Discussion and conclusions

5.1. Utility of the ontology

The utility of the ontology can be demonstrated by discussion of specific phenomena observed in problem-solving. We return to the example of a web designer addressing a client's brief as illustrative of the notion that in addressing any problem a thinker might move to different locations within the space shown in Figure 3.

Consider that the web designer in our example commences in addressing the problem of designing a website for a client by spending a few hours with the client to understand their needs, and then undertakes further user experience research by investigating similar websites. All of this activity sits very much within the upper-left, design thinking quadrant. It is about *expanding the frame* for the problem by learning more about the client, the context for the problem and the cultural domain within which a solution will need to fit, and it is geared towards the creation of a *one-off solution to meet the needs of the client.*

Yet suppose that, at some point during the creation of an early mock-up of the website, this same designer realizes that they can save time by, say, using a spreadsheet to generate some code. Without knowing the specifics of this activity, we can know that this same spreadsheet could be used *in other similar situations* and that in producing it the designer needed to create an *abstraction of the problem.* This activity, as a part of addressing the same problem, is computational thinking, and sits in the lower-right quadrant of Figure 3.

Consider an analogous example in the use of *parametric design methods* within engineering design (Woodbury 2010). An engineer, during their design activity and after attempting to expand the frame by understanding the problem, takes the time to set up a parametric model to generate potential solutions to the problem. The creation of the parametric model requires computational thinking to create the right abstraction for an algorithm that can generate usable designs. Yet the results from this parametric modelling feed back into design thinking about the needs of the users, and so on. In a similar way, a designer – even in the scope of one 'overarching' design problem – may develop a *shape grammar* (Stiny & Gips 1971; computational thinking focussed) and then apply that shape grammar as a part of finding a solution (design thinking focussed).

These examples illustrate the way that, within the scope of a single problem, a thinker might move between design thinking and computational thinking – and may likely do so many times over. This can be described as what Markauskaite & Goodyear (2017) refer to as *epistemic fluency*, the ability to be 'flexible and adept with respect to different ways of knowing about the world' (Markauskaite &

Goodyear 2017, p. 1). This introduces the idea that *fluency* between design thinking and computational thinking might be a desirable trait in many professions; a notion that has not been adequately explored.

The two constructs, of design thinking and computational thinking, have both been widely taken up within formal secondary education, yet they are largely taught and discussed in isolation from one another. It may be appropriate to consider that, given that these two forms of thinking are complementary ways of approaching problems, they might be taught in a way that emphasizes this relationship.

5.2. Are design patterns design thinking or computational thinking?

Design patterns are ways of capturing the essence of a design solution at a level of abstraction that allows the thinking behind it to be reused (Alexander 1977). The idea of design patterns has become popular in many domains (such as architecture, experience design, teaching and software engineering), because 'each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice' (Alexander 1977, p. x). Given that design patterns are about creating a useful abstraction from a problem, do they constitute design thinking (having design in their name) or computational thinking according to this ontology? The response is that they are typically *used* during design thinking, but they are *created* using computational thinking.

Designers tend to use design patterns during design activity, where they are available, to help them in their design thinking. They are employed as a part of expanding the frame of the problem, to help the designer shift their frame for the problem in the direction of useful solutions. As such, this sits in the upper-left quadrant of Figure 3.

In contrast to this, if we consider *a thinker trying to address the problem of creating a design pattern,* then this activity fits within the lower-right quadrant as computational thinking. The thinker is trying to create a solution that can be generally applied 'a million times over'. This can only be done by finding the level of abstraction that is appropriate – which necessarily involves, at some stage in the thinking, a narrowing of the frame.

5.3. What is in the upper-right and lower-left quadrants?

The ontological space portrayed in Figure 3 has a lower-left quadrant and an upper-right quadrant that have not yet been discussed. We propose that these spaces are not utilized by thinkers in addressing design problems. If this is true, it follows that *all thinking in addressing a design problem fits into either design thinking or computational thinking.*

The upper-right quadrant is characterized by an orientation towards a solution that can be generally used, together with an orientation towards expanding the frame for the problem. This is analogous to a doctoral student attempting to create a model of a phenomenon, who is continually reading additional research and introducing new variables. Such thinking might be labelled *ineffective* in addressing a problem, as it seems unlikely (perhaps impossible) that such thinking will lead to a solution. This quadrant seems unusable in attempts to address design problems based on its lack of convergence.

There are some forms of thinking that do fit into the upper-right quadrant. For example, the development of theory (e.g., Einstein's theory of relativity or Kuhn's structure of scientific revolutions) may involve both a general solution and general framing. Yet such thinking does not relate to addressing design problems.

The lower-left quadrant is characterized by an orientation towards a specific solution with an orientation towards narrowing of the frame. This is analogous to a designer who, upon being given a set of requirements, moves towards a single solution by removing any part of the problem that may bring complications, ending up with a solution that does not meet the requirements. This seems to be a *paradoxical* thinking strategy – if the problem requires a singular, specific solution, how can moves to narrow the frame produce a solution?

Similarly, there are forms of thinking that do fit into the lower-left quadrant. For example, solving a mathematical equation for given boundary conditions has a (very) specific solution, and occurs within a very specific frame. Yet again, this kind of thinking does not relate to addressing design problems.

5.4. A dual process model of design thinking and computational thinking?

In the ontological space for reasoning in addressing problems proposed in Figure 3, there seems to be little use for the upper-right and lower-left quadrants. This suggests that design thinking (upper-left) and computational thinking (lower-right) are two different ways in which people reason when addressing problems. This is a *dual process model* in which two different processes – design thinking and computational thinking – together give rise to the phenomenon of humans addressing problems (see Table 1).

The use of the term *dual process model* to describe reasoning dates back to the work of Wason & Evans (1974), who were attempting to explain how participants were reasoning during an experiment. They proposed that their results might not be attributable to a single kind of reasoning – either a heuristic process or an analytical process – but rather that both kinds of reasoning were perhaps being used. Kahneman (2011) and others built on such work to propose a dual process model for the way that people use 'fast' (System 1) and 'slow' (System 2) thinking in responding to the world. In both cases, the experience of reasoning does not 'feel' to subjects as being bifurcated into two different systems – it is just experienced as reasoning – yet empirical evidence supports the presence of distinct processes (Kannengiesser & Gero 2019).

This model is useful for addressing some issues within the literature. The lack of clear definitions for either design thinking or computational thinking is understandable given that they are currently discussed in isolation from each other. It is only through understanding – and experimenting with – how thinkers move fluently from one type of reasoning to the other and back again, when addressing a problem that such definitions may grow in clarity. Design thinking and computational thinking are separate and ontologically distinct, and problem-solving appears to, in general, require both of them – just not at the same time.

One way that this dual process model might be used is to understand historical – and perhaps future – tools that support designers. In different fields of design, there

Table 1. A dual process model of design thinking and computational thinking

Design thinking	Computational thinking
Thinker is trying to expand the frame of the problem to capture its complexity.	Thinker is trying to narrow the frame of the problem to abstract away unnecessary complexity.
Thinker is aiming to create a specific solution to the problem.	Thinker is aiming to create a general solution to the problem.

have been development of a range of computational tools that aim to support both design thinking and computational thinking and fluid movement between them. In architecture, for example, Frank Gehry has been an advocate of technologies that allow for a Master Model of a building that the whole team can use, and that looks towards fluid movement between these two processes (Gehry *et al.* 2020). Similar support for both processes can be seen in the current generation of tools used by UX/UI designers that explicitly support movement from conceptual designing (e.g., freehand sketching) through to deployment at scale (e.g., automated generation of code; Bexiga *et al.* 2020). Tools of the future may further support the fluid movement between design thinking and computational thinking.

6. Conclusions

This paper has positioned design thinking in relation to computational thinking and, in doing so, contributed to the theoretical foundations of design thinking. The proposed ontology places design thinking and computational thinking in relation to each other as regions within an ontological space of approaches for addressing problems, with axes of specificity of framing and specificity of solutions.

The paper raises new questions about the ways in which people move with fluency between design thinking and computational thinking when addressing problems. For example: What are the expectations of how different professionals (e.g., designers and computer scientists) might move within this space when addressing problems? What are the implications for how design thinking and computational thinking are taught within formal education, at all levels?

We suggest that design thinking and computational thinking are not mutually exclusive – as might be implied by the lack of literature addressing the relationship between them – but rather are mirror images of each other in relation to the two ontological categories of solutions and framing.

Acknowledgment

Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Financial support

This work is supported in part by a grant from the US National Science Foundation to the second author (grant number CMMI-1762415).

References

Alexander, C. 1977 A Pattern Language: Towns, Buildings, Construction. Oxford University Press.

Barsalou, L. W. 2008 Grounded cognition. *Annual Review of Psychology* **59**, 617–645.

Beckman, S. L. 2020 To frame or reframe: where might design thinking research go next? *California Management Review* **62** (2), 144–162.

Bexiga, M., **Garbatov, S.** & **Seco, J. C.** 2020 Closing the gap between designers and developers in a low code ecosystem. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings,* Association for Computing Machinery pp. 1–10.

Brennan, K. & **Resnick, M.** 2012 New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vol. 1, Vancouver, Canada.* American Educational Research Association (AERA).

Brown, T. 2008 Design thinking. *Harvard Business Review* **86** (6), 84–92.

Brown, J. S., **Collins, A.** & **Duguid, P.** 1989 Situated cognition and the culture of learning. *Educational Researcher* **18** (1), 32–42.

Clancey, W. J. 1997 *Situated Cognition: On Human Knowledge and Computer Representations*. Cambridge University Press.

Cross, N. 2001 Designerly ways of knowing: design discipline versus design science. *Design Issues* **17** (3), 49–55. http://www.jstor.org/stable/1511801.

Cross, N. 2011 *Design Thinking: Understanding How Designers Think and Work*. Berg.

De Bono, E. 1995 Serious creativity. *The Journal for Quality and Participation* **18** (5), 12–18.

Dorst, K. 2011 The core of 'design thinking' and its application. *Design Studies* **32** (6), 521–532.

Dorst, K. & **Cross, N.** 2001 Creativity in the design process: co-evolution of problem–solution. *Design Studies* **22** (5), 425–437.

Dorst, K. 2015. *Frame innovation: Create new thinking by design*. MIT press.

Ennis, R. H. 1993 Critical thinking assessment. *Theory into Practice* **32** (3), 179–186.

Gehry, F., **Lloyd, M.** & **Shelden, D.** 2020 Empowering design: Gehry partners, Gehry technologies and architect-led industry change. *Architectural Design* **90** (2), 14–23.

Gero, J. S. 1990 Design prototypes: a knowledge representation schema for design. *AI Magazine* **11** (4), 26.

Gero, J. S. 1998 Conceptual designing as a sequence of situated acts. In *Artificial Intelligence in Structural Engineering* (ed. I. Smith), pp. 165–177. Springer.

Goffman, E. 1974 *Frame Analysis: An Essay on the Organization of Experience*. Harvard University Press.

Goldman, S., **Carroll, M. P.**, **Kabayadondo, Z.**, **Cavagnaro, L. B.**, **Royalty, A. W.**, **Roth, B.**, **Kwek, S. H.** & **Kim, J.** 2012 Assessing d.learning: capturing the journey of becoming a design thinker. In *Design Thinking Research: Measuring Performance in Context* (Eds. H. Plattner, C. Meinel, & L. Leifer), pp. 13–33. Springer.

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

Gruber, T. R. 1993 A translation approach to portable ontology specifications. *Knowledge Acquisition* **5** (2), 199–220; doi:10.1006/knac.1993.1008.

Johansson-Sköldberg, U., **Woodilla, J.** & **Çetinkaya, M.** 2013 Design thinking: past, present and possible futures. *Creativity and Innovation Management* **22** (2), 121–146.

Kahneman, D. 2011 *Thinking, Fast and Slow*. Macmillan.

Kannengiesser, U. & **Gero, J. S.** 2019 Design thinking, fast and slow. *Design Science* **5**, e10; doi:10.1017/dsj.2019.9.

Kelly, N. & **Gero, J. S.** 2015 Situated interpretation in computational creativity. *Knowledge-Based Systems* **80**, 48–57.

Kimbell, L. 2011 Rethinking design thinking: Part I. *Design and Culture* **3** (3), 285–306.

Kimbell, L. 2012 Rethinking design thinking: Part II. *Design and Culture* **4** (2), 129–148.

Koh, J. H. L., **Chai, C. S.**, **Wong, B.** & **Hong, H.-Y.** 2015 *Design Thinking for Education: Conceptions and Applications in Teaching and Learning*. Springer.

Lawson, B. 2006 *How Designers Think: The Design Process Demystified (4th ed.)*. Elsevier.

!Markauskaite, L. & **Goodyear, P.** 2017 *Epistemic Fluency and Professional Education.* Springer.

Minsky, M. 1974 A framework for representing knowledge. *MIT-AI Laboratory Memo* 306, June 1974.

National Research Council 2010 *Report of a Workshop on the Scope and Nature of Computational Thinking*. National Academies Press.

Nelson, H. G. & **Stolterman, E.** 2003 *The Design Way: Intentional Change in an Unpredictable World: Foundations and Fundamentals of Design Competence*. Educational Technology.

Poon, J. & **Maher, M. L.** 1997 Co-evolution and emergence in design. *Artificial Intelligence in Engineering* **11** (3), 319–327.

Rittel, H. W. & **Webber, M. M.** 1973 Dilemmas in a general theory of planning. *Policy Sciences* **4** (2), 155–169.

Schön, D. A. 1983/2017 *The Reflective Practitioner: How Professionals Think in Action.* Routledge.

Shute, V. J., **Sun, C.** & **Asbell-Clarke, J.** 2017 Demystifying computational thinking. *Educational Research Review* **22**, 142–158.

Simon, H. A. 1969/2019 *The Sciences of the Artificial*. MIT Press.

Stiny, G. & **Gips, J.** 1971 Shape grammars and the generative specification of painting and sculpture. In *Information Processing '71 (IFIP)* (ed. C. V. Freiman), pp. 1460–1465. North-Holland.

Suwa, M., **Gero, J.** & **Purcell, T.** 2000 Unexpected discoveries and S-invention of design requirements: important vehicles for a design process. *Design Studies* **21** (6), 539–567.

Voogt, J., **Fisser, P.**, **Good, J.**, **Mishra, P.** & **Yadav, A.** 2015 Computational thinking in compulsory education: towards an agenda for research and practice. *Education and Information Technologies* **20** (4), 715–728.

Wason, P. C. & **Evans, J. S. B. T.** 1974 Dual processes in reasoning? *Cognition* **3** (2), 141–154; doi:10.1016/0010-0277(74)90017-1.

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 A: Mathematical, Physical and Engineering Sciences* **366** (1881), 3717–3725.

Wing, J. M. 2011 Research notebook: computational thinking – what and why. *The Link Magazine* **6**, 20–23.

Wing, J. M. 2019 *A conversation about computational thinking*, Educational Future Frontiers, 1–10, https://education.nsw.gov.au/our-priorities/innovate-for-the-future/education-for-a-changing-world/media/documents/future-frontiers-education-for-an-ai-world/Computational-Conversation_1_A.pdf.

Woodbury, R. 2010 *Elements of Parametric Design*. Routledge.