Article

Journal of Educational Computing Research
2022, Vol. 60(4) 877–905
© The Author(s) 2021
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/07356331211055379
journals.sagepub.com/home/jec
SAGE

Unleashing the Potential of Abstraction From Cloud of Computational Thinking: A Systematic Review of Literature

Ndudi O. Ezeamuzie[1], Jessica S.C. Leung[1], and Fridolin S.T. Ting[2]

Abstract
Although abstraction is widely understood to be one of the primary components of computational thinking, the roots of abstraction may be traced back to different fields. Hence, the meaning of abstraction in the context of computational thinking is often confounded, as researchers interpret abstraction through diverse lenses. To disentangle these conceptual threads and gain insight into the operationalisation of abstraction, a systematic review of 96 empirical studies was undertaken. Analysis revealed that identifying features of entities, extracting relevant features, discovering patterns, creating rules and assembling the parts together were the core actions of abstraction. With the primary aim of simplifying practical procedures, abstraction was operationalised as the sophistication of a program, the matching of patterns, the creation of alternative representations, the transfer of solutions, the measurement of a learner's activity and reading program codes. There is an obvious need for researchers to align the conceptual meanings they have established of abstraction with the practical facts of operationalisation. The need to empirically validate emerging models and the implications for future research are discussed.

[1]Faculty of Education, University of Hong Kong, Hong Kong
[2]Department of Applied Mathematics, The Hong Kong Polytechnic University, Hong Kong

Corresponding Author:
Ndudi Ezeamuzie, Faculty of Education, University of Hong Kong, Pokfulam Road, Hong Kong.
Email: amuzie@connect.hku.hk

Keywords
abstraction, computational thinking, problem-solving, decomposition, generalisation

Introduction

In 2006, Wing called on computer science (CS) educators to look beyond their domain and teach non-CS majors 'ways to think like a computer scientist' (Wing, 2006, *p*. 35). Wing's call to make the computer scientist's ways of thinking accessible to everyone, commonly referred to as computational thinking (CT), is often understood to be a seminal moment in the development of CT (Grover & Pea, 2013; Shute et al., 2017). With the common aim of deconstructing what it means to think like a computer scientist, researchers have framed CT in different ways (e.g. Barr et al., 2011; Brennan & Resnick, 2012; Csizmadia et al., 2015; Korkmaz et al., 2017; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016). These frameworks contain overlapping constructs, which emphasise the complex nature of CT and the associated difficulty in delineating the potentially boundless thinking styles of computer scientists into neat categories. For example, 'object-oriented' mapping is a powerful problem-solving approach in CS, but it is rarely mentioned in CT discourse. Therefore, reaching consensus in this area appears to be an uphill battle, and may not be feasible (Ezeamuzie & Leung, 2021). To advance CT research and practice, it is important to avoid the trap of seeking an overarching definition of CT and to instead explore how the constituent cognitive practices of CT might be harnessed to solve everyday problems.

In this study, we focus on abstraction (or abstract thinking), one of the core cognitive practices that computer scientists deploy to solve problems. According to Kramer (2007), a computer scientist's propensity to produce well-designed programs is directly related to the ability to think abstractly. This view was echoed by Colburn and Shute (2007), who showed that breakthroughs in the design of programming languages, operating systems, network architecture and software have been primarily rooted in abstraction. Similarly, the centrality of abstraction in both the theories and practices of CT is undisputable. It is a consistent feature of most CT frameworks and a core component of CT Wing (2008). Moreover, abstraction has been the most significant component of CT in empirical studies measuring learners' CT development (Ezeamuzie & Leung, 2021). According to Grover and Pea (2013), abstraction is not just the keystone of CT but the very element that distinguishes it from other thinking styles.

Abstraction is neither a new nor an emerging idea. Rather, it has a rich history that spans different knowledge domains, such as mathematics (Cetin & Dubinsky, 2017) and CS (Colburn & Shute, 2007). Similar to complex constructs such as problem-solving, intelligence and critical thinking, abstraction is operationalised differently across different domains and often best demonstrated with examples. For instance, the abstract of any scholarly article, which summarises the article's argument, methods and findings, may be regarded as an abstraction of the article. In this case, the quality of the abstraction is a subjective assessment of how well the abstract summarised the article. However, it would be too simplistic to make an analogy from this case to instances of abstraction in other domains. The nature of abstraction varies by context, as demonstrated in Barr and Stephenson's (2011) examples of abstraction in mathematics, science, social studies and language arts.

According to Guzdial (2008), making CT (including abstraction) mainstream in all areas of knowledge will require tapping into and drawing on the strengths of other domains (e.g. psychology, sociology and education) in the reform of CS education. For computer scientists, abstraction is often interpreted as either the process of focusing on the main features in a system or the process of extracting common features from several instances and generalising the model (Hill et al., 2008; Kramer, 2007). However, as we have begun to shift between delivering traditional CS education and extending the constructs of CS to CT research, the clarity of abstraction as a concept has emerged as a point of concern. The difficulties revealed in these trends reminded us of Nardelli's (2019) warning that promoting CT as a new subject may potentially harm CS education. Obviously, we have not been alone in the quest to disentangle the core meaning of abstraction from its operating models. Similar disagreements between researchers over the fundamental meaning of abstraction were highlighted in Cetin and Dubinsky's (2017) discussion of how abstraction might act as a bridge between CT and theories in mathematics.

The more we read, the more researchers' ways of operationalising abstraction left us confused. By operationalisation, we meant the actual implementation of abstraction in CT empirical studies, which is different from the conceptual definitions. We concluded that gaining a deeper understanding of abstraction is an important step in advancing our own CT research and that of others. Apart from being motivated by our natural curiosity about the recent revelations on abstraction, we were interested in uncovering issues in aligning the various conceptual meanings of abstraction with concrete examples of its operationalisation. Therefore, we chose to systematically review the CT literature and expand our understanding of the ways that abstraction has been conceptualised and operationalised. The findings from our review will provide a platform for educators, including those who are interested in joining the CT community, to clearly grasp the conceptualisation and operationalisation of abstraction. In addition, because abstraction is an interdisciplinary cognitive ability, clarifying this pillar of CT will help to make the boundaries between CT research and adjacent fields more permeable, and thus facilitate access to researchers from other fields, who may not be well informed about the theory and subtleties of abstraction from CS research.

In this systematic review, we addressed the following research question – *How has abstraction been conceptualised and operationalised in CT studies assessing learners' abstraction skills?*

Note that the terms conceptualisation and operationalisation were contradistinguished in this study. *Conceptualisation* referred to the prosaic meaning of abstraction used in a given article while *operationalisation* referred to the action that measured the implementation of abstraction.

Theoretical Framework

This section describes the philosophical underpinnings of abstraction as a concept and its relation to CT. To disentangle various proposed meanings of abstraction from each other, this theoretical discourse sought to understand the evolution of the concept of abstraction and thus generate a framework for analysing this construct.

Computational Thinking

Wing's (2006) call for everybody to acquire CT was rooted in the hypothesis that the computer scientist's thinking style could provide a general problem-solving strategy by harnessing the intellectual power inherent in CS for purposes far beyond CS. This assertion was not peculiar to Wing, appearing continually in the historical development of CT (Tedre & Denning, 2016). Similar statements appeared in Alan Perils' call for programming to be adopted in the liberal arts (Perils, 1962, as cited in Guzdial, 2008), Donald Knuth's view of how teaching a computer to perform tasks might increase conceptual clarity (Knuth, 1974) and Seymour Papert's vision of the epistemic development that occurs when children teach computers to think (Papert, 1980).

When Wing first shared this vision, CT was associated with various concepts, such as problem-solving, system design, recursive thinking, parallel processing, data interpretation, abstraction, decomposition and heuristic reasoning (Wing, 2006). Although this conceptualisation of CT has drawn criticism for being overly broad and somewhat vague (Mannila et al., 2014), Wing's central argument was vivid – anyone can develop the thinking style of a computer scientist and apply it to solve problems. Since Wing's first proposal, several frameworks have emerged that potentially clarify the best way to model CT, such as the framework developed by the International Society for Technology in Education (Barr et al., 2011; Barr & Stephenson, 2011), Computing at School (Csizmadia et al., 2015) and others (e.g. Brennan & Resnick, 2012; Korkmaz et al., 2017; Selby & Woollard, 2013; Shute et al., 2017; Weintrop et al., 2016).

Concepts like CT have multiple definitions because each researcher tends to interpret concepts through a unique lens. Reasonably, the advantages of the variety of CT definitions cannot be ignored; they contribute to a richer understanding of the multifaceted nature of CT. Although it would be desirable for the purposes of learning and assessment to have a focused definition of CT, Nardelli (2019) cautioned against attempts to map it out as a school subject and suggested that CT should be interpreted 'as a shorthand' for teaching CS to every student (*p*. 32). Most empirical studies measuring participants' development of CT have operationalised it as a composite of programming concepts, with a preference for assessment-based frameworks (Ezeamuzie & Leung, 2021). However, a sizable number of these studies had no clear model distinguishing CT from programming (Ezeamuzie & Leung, 2021).

In this study, we interpreted CT through Wing's (2006) lens as thinking like a computer scientist. This holistic approach aimed to avoid commonly encountered pitfalls such as exaggerated claims and narrow views about computing in CT education (Denning et al., 2017; Tedre & Denning, 2016).

Abstraction

Abstraction permeates every aspect of computing. It depicts the high-level cognitive practices that learners use to break down tasks into manageable fragments, modularise solutions and design their algorithms. The act of 'assessing abstraction' often implies measuring students' aptitude to demonstrate the use of abstraction in solving problems holistically (Hill et al., 2008). The centrality of abstraction is apparent in Kramer's (2007) description of how none of the 60 course modules in a 4 year Master of Engineering programme was tailored towards abstraction, yet the programme's students had to apply abstraction to problem-solving in all of its modules. In another study, Perrenet et al. (2005) were concerned about how well their department's undergraduate CS curriculum performed in training students to think like computer scientists. They equated 'thinking like computer scientists' with mastering the level of abstraction that students demonstrate in designing algorithms. Therefore, the designation of abstraction as a core and consistent dimension in almost every CT framework is a standard of the field.

The concept of abstraction has featured in studies of students' thought processes. For example, Aharoni (2000) observed a three-level continuum of abstraction at work in the thinking used by students when programming in a data structure class. From low to high levels of abstraction, the three elements of the continuum were programming-language oriented thinking, programming-oriented thinking and programming-free thinking. Another experiment studied the use of abstraction in an algebra class; Hazzan (1999) discovered that students coped with new concepts by interpreting them at a level of abstraction lower than that at which the concepts were introduced, as measured by the following three-dimensional framing of abstraction:

(a) *The quality of the relationship between an object and the thinking person* – This implies that for a given concept (the object), a continuum from the concrete to the abstract exists, and the level of abstraction observed is not inherent to the object but rather is a property of the person's mind; in this dimension, how the person perceives the level of abstraction will be influenced by the person's prior experience of the object.
(b) *A reflection of the process–object duality* – In this dimension, concepts are conceived either as a process (low-level abstraction) or as an object (high-level abstraction).
(c) *The degree of complexity of the concept of thought* – This assumes that complex entities are abstract.

Considering these levels of abstraction (Aharoni, 2000; Hazzan, 1999), it is unclear how to ascertain whether a particular level of abstraction should be deemed a desirable learning target. Colburn and Shute (2007) argued that abstraction objectives in CS and mathematics are ‘information hiding’ and ‘information neglecting’, respectively. However, this distinction may still not resolve the ambiguity of the levels of abstraction without the aid of a proper context. For example, how is low-level abstraction to be understood? According to Hazzan (1999), viewing a concept as a process rather than an object denoted a low level of abstraction. On this view, achieving a high level of abstraction (i.e. object representation) was designated the appropriate learning target in the context of mathematics (e.g. abstract algebra). Yet, in Aharoni’s (2000) investigation of the class on data structures, students developed programming-context thinking (i.e. programming-language oriented thinking and programming-oriented thinking), which represented a low level of abstraction. In this different context, achieving low-level abstraction was a useful learning goal. These analogies reinforced the need to be wary of ‘across-the-board’ promotion of low-level or high-level abstraction. Context matters!

Kramer (2007) illustrated how abstraction could be described through dual lenses – the *decomposition* of a task or the *generalisation* of a pattern. Through the first lens, abstraction is the process of removing some of the properties of a complex system with the aim of focusing on the system’s main features. This way of understanding abstraction involves a decompositional, top-down, reductionist or modelling perspective (e.g. the use of maps to create abstractions of complex railway lines). In this mode, the fine-grained operations of a system or subsystem are hidden, as evidenced in modelling (Israel et al., 2015). Through the second lens, abstraction is the process of formulating a general concept by identifying and extracting common features from several instances. This adopts generalisation or the bottom-up convention in sampling instances to generate a common theme. Examples of this form of reasoning include the use of data structures, such as classes in programming, for generalising the properties and behaviour of objects, and the creation of a rule for sorting system files, program files and documents into an appropriate folder structure in the Windows operating system (Harimurti et al., 2019).

The nature of abstraction has also been modelled in other frameworks. These include the extraction, decontextualisation or essence classes of abstraction (Cetin & Dubinsky, 2017) and the conceptual, formal or descriptive classes of abstraction (Hill et al., 2008). Although the relevant dimensions have different labels, these abstraction frameworks are similar in meaning. According to Cetin and Dubinsky (2017), the extraction class illustrates the practice of finding the common features of an object, the decontextualisation class depicts the representation of concepts in forms that are independent of their original contexts, and the essence class is associated with formally representing a concept through decomposition. In the abstraction model created by Hill et al. (2008), formal abstraction involved the ability to simplify and create symbolic representations, which is identical to Kramer’s (2007) abstraction form of decomposition and Cetin and Dubinsky’s (2017) essence class. Similarly, descriptive abstraction focusses on the ability to construct a generic model by identifying the similarities and differences between items in a group of objects, which aligns with Kramer's (2007) generalisation class and Cetin and Dubinsky's (2017) extraction class of abstraction.

Perrenet et al. (2005) conceived abstraction differently, positing four levels of abstraction that presumably could be clearly distinguished from each other when computer scientists design algorithmic solutions. These included the *execution level* (the algorithm is precisely designed to run on a specific, concrete machine), the *program level* (the algorithm is a process designed by focusing on specific programming language), the *object level* (the algorithm is independent of any programming language) and the *problem level* (the algorithm is a black box for solving different problems with different levels of complexity).

Although Perrenet et al. (2005) framed the levels of abstraction according to a CS context, the object level and the problem level of abstraction were not restricted to programming and are therefore suitable for categorising other instances of abstraction. The non-restrictive and comprehensive design of the four levels of abstraction make it an ideal model for explaining the patterns of abstraction. In addition, Kramer's (2007) description of the nature of abstraction overlapped with the models described in Cetin and Dubinsky (2017) and (Hill et al., 2008). Therefore, Perrenet et al. (2005) and Kramer (2007) were invaluable for understanding how abstraction has been operationalised in the course of reviewing the empirical articles.

Method

We took the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) statement as a set of guidelines for the framing of our work (Moher et al., 2009). PRISMA contains sets of items that support researchers in conducting and reporting quality systematic reviews and meta-analyses.

Identification of Studies

As established in the description of our theoretical framework, abstraction is a complex concept whose meaning differs depending on the context and knowledge domain. To contextualise our investigation of abstraction in CT and to maximise our coverage of the literature, we searched for articles that contained 'computational thinking' in their title, abstract or keywords fields. The enclosing quotation marks were part of the search term; a schema to restrict the results to exact but case insensitive matches. We ran our search on three leading research databases: Web of Science (WoS), Scopus and Education Resources Information Center (ERIC). These are key indexing databases for multidisciplinary educational research, with comprehensive coverage of quality journal titles and proceedings.

With no further restrictions apart from the search term, the cut-off date for the literature search was 5th of May 2021. The identification stage yielded 5499 documents: WoS (n = 2074), Scopus (n = 3010) and ERIC (n = 415). We considered extending our search results through journal scouring, reference backtracking and researcher checking (Alexander, 2020). However, undertaking these procedures with a collection of 5499 articles was not practical at this stage.

Screening of Documents

With the initial pool of documents identified, we established the inclusion criteria and embarked on the screening work. Tying back to the research question – how has abstraction been conceptualised and operationalised in CT – an article was selected if it reported an empirical study and assessed learners' abstraction (whether measured as a standalone construct or a component of CT). The implementation of the inclusion criteria took place across three sequential screening stages: pre-processing, abstract screening and full-text screening.

Pre-processing Screening. To conduct a high-quality and practical systematic review, researchers often need to set reasonable boundaries. Alexander (2020) called these boundaries 'delimitations'. For example, the search for eligible articles in WoS, Scopus and ERIC was the first gatekeeping measure to maximise search coverage without compromising quality. As the literature search was conducted in three independent databases, the first pre-processing task was to remove all duplicates. The duplicates (n = 1576) were removed, and the remaining 3923 articles were retained for further processing.

The quality of the grey literature varied significantly (Rothstein & Hopewell, 2009). However, disregarding all instances of the grey literature, such as proceedings, might have introduced publication bias. Therefore, we selected peer-reviewed articles (whether published in journals or proceedings) and restricted eligibility to English-language articles published from 2006 onwards. This restriction by year was determined by the year of Wing's (2006) seminal modelling of CT as 'thinking like a computer scientist', which was adopted as the conceptual meaning of CT for this review. This pre-processing screening resulted in the removal of 230 articles and the remaining 3693 were passed to the abstract screening stage.

Abstract Screening. The first step in the abstract screening process was to exclude reviews (e.g. Lye & Koh, 2014; Zhang & Nouri, 2019), editorials (e.g. Denning, 2017; Tissenbaum et al., 2019) and books or book chapters (e.g. Bers, 2017; Yadav et al., 2017). Based on the document types, 888 articles were flagged for not meeting the inclusion criteria (i.e. not empirical studies that assessed learners' abstraction).

The second step was to read the abstracts of the remaining articles (n = 2805). Apart from not meeting the inclusion criteria, other types of excluded studies were theoretical discourses, such as teachers' knowledge in designing and teaching CT in K–6 (Angeli et al., 2016), the required conditions for implementing CT in schools (Repenning et al., 2010) and meta-analyses/syntheses (Denner et al., 2019; Scherer et al., 2019). This process culminated in the exclusion of 2474 articles. The remaining articles (n = 331), consisting of 202 journal articles and 129 proceedings papers, were selected for full-text screening.

Full-Text Screening. This last screening procedure provided a finer layer of inspection – an important step in assessing the eligibility of articles for cases in which determinations could not be based on abstracts. Specifically, a study was excluded if it involved neither the definition nor the operationalisation of abstraction. For example, Durak and Saritepeci (2018) used structural equation modelling to examine the relationship between CT and learners' variables, such as thinking style, academic performance and attitude towards mathematics. Although abstraction was mentioned eight times in their study, it was neither defined nor was it a component of their measurement instrument, which used the CT scale created by Korkmaz et al. (2017). Therefore, the study was excluded. Similarly, studies that measured CT with an instrument that did not identify abstraction as a component (e.g. Korkmaz et al., 2017; Román-González et al., 2017) were excluded if such studies did not define abstraction. For example, a study on the influence of repeated participation in after-school programming on CT (Mouza et al., 2020) was excluded because abstraction was neither defined in the theoretical discussion nor assessed in the multiple choice questions, which asked students to match Scratch blocks to CT concepts. Other studies – such as the use of eye tracking to investigate developmental differences in CT according to gender (Papavlasopoulou et al., 2020) – that did not provide additional information on abstraction in their rubrics were excluded too.

Finally, we scoured the content of journals that appeared more than once in our search and looked at the publication records of authors who had contributed two or more studies after full-text screening. All of the identified articles were found in our initial pool. This could be attributed to the increasing sophistication of database search algorithms and the use of single-term keywords in searches. Of these 331 articles, 235 studies were excluded because they did not define or operationalise abstraction. The screening phase culminated in a list of 96 eligible studies (see Supplementary table) to be charted. Figure 1 summarises the application of this model to identify and screen eligible studies.

Charting and Coding of Data

This stage was the most important stage of the study and required several rounds of discussion between the authors. Given the number of articles identified for a comprehensive review, we charted our data with a spreadsheet. With a focus on the research question, we captured all relevant information to address the conceptualisation (literary meaning) and operationalisation (concrete and measurable implementation) of abstraction.

We were confronted with the challenge of making sense of abstraction from a large number of studies. Dinsmore et al. (2008) encountered similar difficulties during a systematic review study, which aimed to disentangle the definitional boundaries between self-regulation, self-regulated learning and metacognition. Informed by Murphy

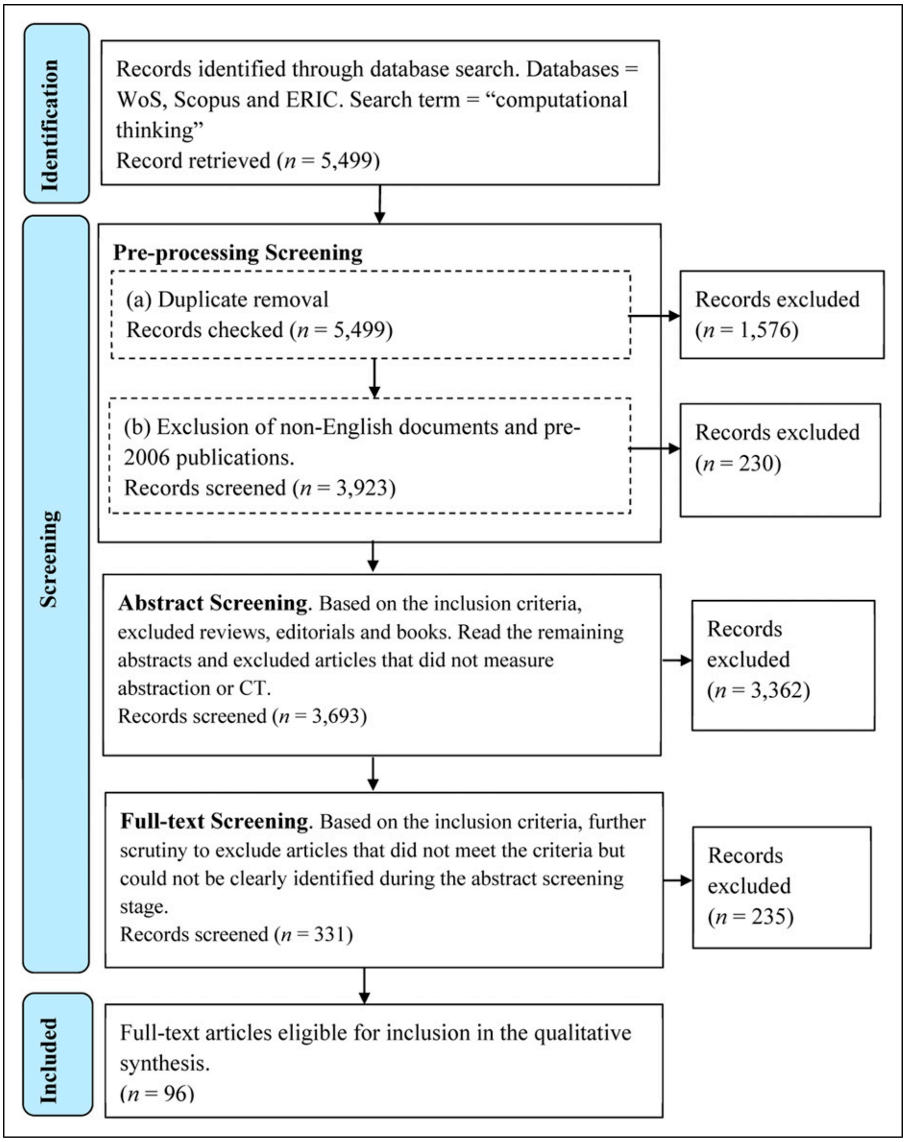
Identification
Records identified through database search. Databases = WoS, Scopus and ERIC. Search term = "computational thinking"
Record retrieved (n = 5,499)
Screening
Pre-processing Screening
(a) Duplicate removal
Records checked (n = 5,499)
Records excluded (n = 1,576)
(b) Exclusion of non-English documents and pre-2006 publications.
Records screened (n = 3,923)
Records excluded (n = 230)
Abstract Screening. Based on the inclusion criteria, excluded reviews, editorials and books. Read the remaining abstracts and excluded articles that did not measure abstraction or CT.
Records screened (n = 3,693)
Records excluded (n = 3,362)
Full-text Screening. Based on the inclusion criteria, further scrutiny to exclude articles that did not meet the criteria but could not be clearly identified during the abstract screening stage.
Records screened (n = 331)
Records excluded (n = 235)
Included
Full-text articles eligible for inclusion in the qualitative synthesis.
(n = 96)

Figure 1. Flow diagram of search strategies and screening for eligible studies for review (Inclusion criterion = CT empirical studies assessing learners' abstraction).

and Alexander's (2000) definitional coding scheme, which queried the degree of explicitness used by researchers to define constructs, Dinsmore et al. (2008) coded definitions into two categories – implicit or explicit. Implicit definition was further sub-categorised into *conceptual* definition (i.e. the construct definition was inferred from original words or phrases in the text), *referential* definition (i.e. the construct definition was derived from another work) or *measurement* definition (i.e. the construct definition was found by use of a measurement instrument).

Although the definitional framework initially appeared to be a promising model for deconstructing the meaning of abstraction, we encountered some ambiguity as to what constitutes an explicit or implicit definition in the charting process. Explicitness usually corresponds to a definition that is clearly stated in a given article. However, we discovered that some authors defined abstraction by referring to other works. For example, Basu et al. (2020) defined abstraction as 'a means of hiding complexity and building something large by putting together collections of smaller parts' (*p*. 914) by citing Brennan and Resnick's (2012) CT framework. If this definition had been coded as an explicit definition, the referential connection to Brennan and Resnick (2012) would be omitted.

To address this problem, we extended the definitional framework by adding dimensionality to the explicitness axis. Using a binary classification (yes/no), the *explicit* variable was coded 'yes' if and only if the authors of a given study advanced their own version of abstraction. Similarly, the binary variable *written* indicated whether meaning of abstraction was clearly stated in a given article (whether referenced from another work or devised by the authors). To illustrate the coding scheme – in a study that detailed the development of CT in a puzzle game, Rowe et al. (2021) defined abstraction as 'the ability to make generalizations from observed patterns and making general rules or classifications about objects, tasks, or information' (*p*. 4). As in this case the authors put forward their own version of abstraction, this study was coded as *explicit* (yes), *written* (yes) and *referential* (no). In a similar way, although Basu et al. (2020) stated the meaning of abstraction in written form, their statement of meaning depended on a citation of Brennan and Resnick (2012), and this study was therefore coded as *explicit* (no), *written* (yes) and *referential* (yes).

To understand the concrete and measurable implementation of abstraction in a given article, we identified the type of instrument used for measurement (e.g. observation, interview, self-report and performance rating). Following on Kramer (2007), the nature of abstraction was coded (i.e. decomposition, generalisation, none or both). Finally, we identified the level/granularity of abstraction represented in the operationalisation; this categorisation was based on the four-tier abstraction theorised in Perrenet et al. (2005) – execution level, program level, object level and problem level.

Results and Discussion

Table 1 summarises the features of the charted studies. Only about 23% of the studies ($n = 22$) included both conceptual and operational definitions. In our view, for complex and amorphous constructs such as abstraction, which tend to be open to interpretation, the probability of the construct being misinterpreted by research audiences will be significantly reduced when both operational and conceptual definitions are presented in a given empirical study. However, among the studies that presented conceptual

Table 1. Summary of the Profiles of the Charted Studies.

Study Characteristic	Conceptualisation, *n* (%)	Operationalisation, *n* (%)
Charted studies	56 (58.33)	62 (64.58)
Explicit	19 (33.93)	4 (6.45)
Written	50 (89.29)	4 (6.45)
Referential	30 (53.57)	38 (61.29)
Nature	28 (50.00)	3 (4.84)
Decomposition	11 (19.64)	4 (6.45)
Generalisation	9 (16.07)	26 (41.94)
Both	8 (14.29)	29 (46.77)
None/Not clear		
Level	–	0 (0)
Execution	–	23 (37.10)
Program	–	0 (0)
Project	–	35 (56.45)
Problem	–	4 (6.45)
None/Not clear		
Measurement types	–	55 (88.71)
Performance rating	–	4 (6.45)
Interviews	–	2 (3.23
Think-aloud	–	3 (4.84)
Self-report		

definitions ($n = 56$), an impressive 89% ($n = 50$) were conspicuously *written*. Recall that the *written* class refers to studies in which definitions were clearly stated in the article and was a superset of the explicit classification (i.e. *written* definitions advanced by study authors themselves). The authors' efforts to state the conceptual meanings of abstraction as clearly as possible were desirable, so that audiences are not subjected to infer the meanings themselves. In the case of operational definitions, it is self-evident that these had to be inferred from instances of the tasks in the measurement instrument.

The nature of abstraction found in Kramer (2007) showed that researchers are twice as likely to conceptualise abstraction from the decomposition than from the generalisation aspect. In contrast, the operational definition (i.e. the measurable implementation) showed that the two aspects – decomposition and generalisation – were often intertwined in tasks that measured abstraction. A certain proportion of the operational definitions could not be clearly associated with either decomposition or generalisation (29/62; 46.7%). This observation was in line with the initial aims of the review and showed that the notion of abstraction remained obscure to many researchers. For example, studies (e.g. Garneli & Chorianopoulos, 2018; Munoz et al., 2018) that measured abstraction as a dimension of CT with Dr Scratch (Moreno-León et al., 2015), an automatic analytical instrument for Scratch projects, could not be clearly classified as either decomposition or generalisation. In Dr Scratch, the grading of abstraction ranked from *basic* (1 point for projects with more than one sprite and script), developing (2 points for definitional blocks – similar to creating user-defined functions) and proficiency (3 points for the use of clones). It was not clear how these features could be mapped to abstraction. In fact, the developers of Dr Scratch have acknowledged these limitations and suggested that the tool was intended to provide formative feedback to improve a user's coding skills and not as a substitute for the formal evaluation of coding projects (Moreno-León & Robles, 2015). Although the centrality of abstraction has been well established in CT theory, its operationalisation seems to be lagging in studies. This was evident from the fact that several studies were excluded from the eligible pool due to abstraction being absent from their CT assessment frameworks; especially the case for studies that measured CT with the self-reporting CT scale of Korkmaz et al. (2017) and the CT test of Román-González et al. (2017). Another plausible explanation for certain cases of failure to classify operational definitions relates to the view of abstraction as being inextricably embedded in learners' CT practices, such as those of programming, and therefore not explicitly measurable.

As shown in Table 1, it was not feasible to classify the conceptualised definitions by level as described in Perrenet et al. (2005) because these definitions were broad and did not limit abstraction to specific platforms, programming languages or applications. For example, in a study that examined the relationship between mental rotation and CT in primary school students, Città et al. (2019) broadly defined abstraction as the removal of irrelevant parts of a problem to focus on its core aspects.

For operational definitions, none of the studies implemented abstraction at the low-execution level. Understandably, any desire for algorithmic solutions that run on specific machines (i.e. the execution level) seem to be out of sync with recent technological developments in internetworking and cross-platform operation of applications. In addition, during data charting, we discovered that the boundary between object and problem levels was blurry. However, we relied on their commonality of requiring no programming and classified such studies as being at the problem level. This explains the absence of any study at the object level.

Four types of measurement emerged from our coding: performance rating, interviews, think-aloud protocols, and self-reporting. From the pool of eligible articles, performance rating (55/62; 88.7%) was the predominant measurement type and represented the cluster of assessments such as test, Bebras task, rubric/artefact analysis, automatic artefact analysis and multiple choice questions. Examples included tests of students' ability to draw electronic circuits of specific designs (Yin et al., 2019), a Bebras challenge that investigated the effectiveness of robotics laboratory training in the development of CT (Chiazzese et al., 2019), a rubric to evaluate whether abstraction was present in the written solutions submitted by in-service teachers in a CT teacher development programme (Kong & Lao, 2019), the automatic analysis of CT with Dr Scratch in a CT-based outreach programme for children aged 9 to 14 (Panskyi et al., 2019) and selecting an option that had the fewest statements while identifying a given picture in a multiple choice question (Basu et al., 2020).

All the measures, except performance ratings, were rarely used. We put forward two plausible explanations for the predominance of performance-based measures over interviews, think-aloud protocols, and self-reporting. First, researchers believe that abstraction is most appropriately measured by the output of processes that require applications of abstraction. Second, the use of interviews and think-aloud protocols may not have been scalable for empirical studies with large numbers of participants. Note that this explanation should be interpreted in the context of this review alone, which excluded from the final pool, studies that assessed CT but omitted abstraction as a factor.

Disentangling the Conceptual Definitions

Abstraction as a Form of Decomposition. Based on Kramer's (2007) dichotomous classification of abstraction into decomposition and generalisation, about 50% of the conceptual definitions considered abstraction solely as a form of decomposition activity. For example, in their investigation of the influence of educational robotics on the development of CT, Atmatzidou and Demetriadis (2016) described abstraction as a process of simplifying complex entities by focusing on their relevant features. Taken together, these studies tended to identify the main activities of decomposition as (a) identifying the relevant and irrelevant details (attributes, characteristics, features, aspects, parts) of an entity (e.g. Kert et al., 2020; Wang et al., 2014), (b) obscuring or removing an entity's irrelevant details (e.g. Angeli & Valanides, 2019) and (c) focusing on only the relevant or important details of an entity (e.g. Wu & Su, 2021). The resulting solutions have often been depicted through new representations such as creation of models and simulations (Jaipal-Jamani & Angeli, 2017; Mendoza Diaz et al., 2020). Judging by this literature, the overarching purpose of abstraction as a decomposition process converges in the simplification (reduction of complexity) of a system. This is consistent with the description of Selby and Woollard (2013) in a review of the most frequently occurring terms of CT.

Abstraction as a Form of Generalisation. Unlike the cases of decomposition, abstraction in the form of generalisation involved (a) identifying the underlying rules or patterns that controlled the behaviour of entities/processes (e.g. Basu et al., 2018; Wu et al., 2019) and (b) developing general rules for entities/processes rooted in the discovered patterns (e.g. Rowe et al., 2021). Although the generalisation approach tended to focus on creating general rules from concrete observations (Barr & Stephenson, 2011), the core purpose converges to simplification, which was also the case for decomposition.

The Intersection of Decomposition and Generalisation. Although our investigation used Kramer's (2007) dichotomous classification of abstraction into decomposition and generalisation, researchers' definitions of these two categories have often overlapped. For example, in validating a CT instrument for grades 4–6, Basu et al. (2020) defined abstraction as obscuring complexity and creating a generic representation by identifying patterns within entities. Similarly, Asbell-Clarke et al. (2021) explicitly defined abstraction as ‘generalizing from observed patterns and making general rules or classifications about objects, tasks, or information by discerning relevant from irrelevant information’ (*p*. 2). These definitions show how abstraction has often depended on both decomposition and generalisation spaces. Hiding the complexity of a system and distinguishing between the core and peripheral components of a problem are elements of decomposition activities. Meanwhile, the generalisation wing conceived abstraction as a process of observing groups of objects, identifying patterns and creating classifications or rules.

In a similar vein, in Bebras challenges (Dagienė & Sentance, 2016), each task is classified as measuring one or more CT skills, including abstraction, as conceptualised in Selby and Woollard (2013). According to Dagienė and Sentance (2016), Bebras tasks are classified as abstraction if the learners distinguish between their important and trivial components and make sense of similarities between structures in completing them. The realization of abstraction as a combination of decomposition and generalisation also featured in a study that compared the development of abstraction in objected-oriented and robotics programming activities (Çınar & Tüzün, 2020). The meaning of abstraction in this study was determined according to both formal and descriptive classes of abstraction (Hill et al., 2008). Formal abstraction referred to the ability to simplify and create symbolic representations (identical to the decomposition approach) while descriptive abstraction referred to the ability to construct a generic model by identifying the similarities and differences between groups of objects (similar to the generalisation approach).

Contrasting Interpretations. As noted in our statement of objectives, this review revealed certain intricacies in the conceptualisation of abstraction that researchers in this area ought to know about. First, the core purpose of abstraction was to simplify processes and outputs, especially in cases where the resulting entity was large. This objective was aptly captured in the description of abstraction as ‘a strategy to make problems or systems easier to think about’ (Chen & Chi, 2020, *p*. 8). However, simplifying complex entities does not necessarily mean creating small replicas of given entities. In fact, Basu et al. (2020) referred to abstraction as ‘building something large’ (*p*. 8) by hiding the object’s complex parts and aggregating its smaller parts.

Second, the implied narrative of abstraction often seemed to suggest that ‘more abstraction is better’. However, this has not always been the case. In Price and Price-Mohr (2018), decreasing abstraction was the preferred learning goal in an exploratory study that evaluated the differences between novice and expert processes of Java programming. There, abstraction was described as the level of polymorphism used in overloading a method. In other words, adding more arguments to the declaration of a method meant ending up with less abstraction. In this scenario, abstraction was depicted as vagueness that needed to be reduced by providing more information as the input of a process. Of course, the overall goal of method overloading is to simplify programs by providing alternative methods for objects with different input parameters.

Although these different views of abstraction were conceptually aligned to simplifying the design of systems, this observation reinforced the need to understand the contest of abstraction.

Therefore, in our view, abstraction should not be limited to a specific set of activities but rather should be understood as the summation of cognitive activities whose common goal is the simplification of systems, which may include decomposition and generalisation.

Fitting Other Definitions of Abstraction. Some of the definitions of abstraction in our review did not fit the decomposition-versus-generalisation abstraction mould. In this section, we highlight other definitions that the authors presented to support their studies.

Problem *formulation* – Newton et al. (2020) described abstraction as the sum of activities that learners engage in to formulate solutions to problems before coding. According to Newton et al. (2020), students are abstracting when they make decisions about the behaviour of agents in game design before the actual coding. These prior-to-coding activities could take diverse forms, including the decomposition and generalisation of behaviour. In this conceptualisation, abstraction stood out as the sum of cognitive planning and was distinct from writing computer code.

Data storage and manipulation – In Grover et al. (2015), abstraction was defined as the transformation of real-world data into variables and the use of functions to find manageable solutions to problems. This view was similar to the notion of abstraction as a process of storing and manipulating data (Looi et al., 2018).

Program testing and verification – Deng et al. (2020) defined abstraction as the ability to test and verify computer programs with programming tools, and Sulistiyo and Wijaya (2020) described abstraction as the design of correct mathematical representations of problems.

The Operationalisation of Abstraction

In this section, we highlight six ways in which abstraction was operationalised in the selected articles. Noticeably, studies that implemented abstraction via Bebras tasks generally did not fit in any one of these categories. Although the underlying considerations for abstraction tasks in Bebras are well defined, the variability of tasks across studies (e.g. Baek et al., 2019; del Olmo-Muñoz et al., 2020) limited their classification into a particular theme.

The Sophistication of Programming Concepts. Some authors operationalised abstraction as a measure of the presence and frequency of certain programming concepts. For example, in using Scratch and Alice to assess the CT skills of students, Allsop (2019) designed a rubric whereby abstraction could be measured on a scale of 0–2 for the presence of methods/functions and event handlers in students' artefacts. The use of built-in methods/functions and simple event handlers received a score of 1, whilst artefacts that contained user-defined methods/functions and sophisticated event handlers received a score of 2. When these programming concepts were entirely absent from a student's artefact, a score of 0 was assigned for abstraction.

However, the programming concepts associated with abstraction were not consistent in other implementations. By default, studies that measured CT with Dr Scratch (Moreno-León et al., 2015) relied on the presence of certain programming concepts and were different from the features in Allsop (2019). In Dr Scratch, abstraction was operationalised on a scale from 1 to 3 for the presence of two or more scripts and sprites, the use of definition blocks (equivalent to user-defined functions) and the use of clones, respectively. If these features were entirely absent, a score of 0 was assigned for abstraction.

The sophistication of programming concepts was used as a measure of abstraction in studies that used platforms other than Scratch and Alice. Using AgentCubes, a three-dimensional simulation authoring system from the scalable game design project (Repenning et al., 2015), Leonard et al. (2016) investigated the influence of robotics and game design on middle school students' CT skills. They associated abstraction with the creation of unique games containing novel actors/agents. In ascending order, abstraction was graded as emerging (1 point) if the agents and backgrounds resembled the lesson demo. A moderate grade (2 points) was assigned if a student only created agents or backgrounds. A student received a substantial grade (3 points) if both agents and backgrounds were new.

A similar operationalisation was adopted in a study that examined how students develop CT in informal learning environments (Newton et al., 2020). However, Newton et al. (2020) extended the implementation of abstraction as a measure of the usage levels of conditionals (i.e. whether a conditional such as an 'if statement' was unclear, a single-condition comparison or a multiple-conditions comparison). Corral et al. (2019) operationalised abstraction as the number of stored procedures (equivalent to user-defined functions/methods) in a project. The authors associated abstraction with the software engineering metric. Whether in Scratch, Alice, AgentCubes and even the software metric, the underlying measured concepts associated with abstraction were not consistent.

Matching Patterns. Extracting patterns that existed both within a process and between objects was a common implementation of abstraction in the reviewed studies. In Atmatzidou and Demetriadis (2016), students were asked to find examples of behaviour common to two robotics programs as a measure of abstraction. Calderon et al. (2020) also examined students' abstraction ability by challenging them to find commonalities between objects in opposite stacks of the Tower of London test (Shallice et al., 1982) – a test of executive functioning in which six boxes are drawn on the left and another six boxes on the right. Another instance of pattern matching assessed abstraction as students' ability to select the fewest statements that distinguished a specific picture from others (Basu et al. (2020).

For students aged 6 to 8, Kanaki and Kalogiannakis (2019) measured abstraction as their ability to identify animals that live in their geographical region by selecting the correct images on a worksheet. The task's design was informed by the belief that abstraction is linked to the ability to identify common patterns in sets of entities. Similarly, Rijke et al. (2018) paired up students in a card guessing game to investigate the interaction of age in the development of abstraction in primary school students. The first student sketched the concrete nouns in a deck of cards. Without revealing the cards, the second student was tasked to guess the word from the sketches. According to Rijke et al. (2018), high-quality sketches tended to focus on the essential features of the common noun, and abstraction was determined according to the number of correct guesses that students achieved with their sketches. Another illustration of abstraction as the ability to match patterns was found in a study that explored the relationship between programming environments and CT in students aged 10 to 12 (Wu & Su, 2021). These students were asked to distinguish important components from irrelevant details by selecting the peripheral item in multiple choice questions.

In summary, the focus on pattern revealed its association with abstraction, which was conceptually found to be a key practice in generalisation.

Creating Alternative Representations – Modelling and Simulation. Abstraction was also operationalised as the ability of students to create alternate representations of a problem, which is consistent with the definition of abstraction proposed by Jaipal-Jamani and Angeli (2017) as the modelling and simulation of problems. Kong and Lao (2019) demonstrated this aspect of abstraction in assessing in-service teachers' CT skills in a teacher development programme. The participants were presented with a pictorial representation of counters in a supermarket. For each counter, the average queueing time and number of customers were provided. Abstraction was assessed as the ability to capture the number of customers queueing at each counter, design mathematical representations to calculate their total queueing times and develop a generic algorithm for comparing the counters and finding the fastest of them.

Alternative representations take different forms and are not limited to mathematical formulas. In a physics and engineering class, Yin et al. (2019) framed abstraction as the ability to draw electronic circuit diagrams for a given design. In their exploration of ways to develop and assess integrated CT skills, they reasoned that representing the design through electronic circuit diagrams elicited students' abstraction ability irrespective of the maker activities, whether e-textile, breadboard, Makey-Makey or Arduino.

Transfer of Problem Solutions. Abstraction was operationalised as the ability to use solutions from past experience to solve new sets of problems with similar structures (Lee et al., 2014). Relying on children's natural interest in game playing and the affordances of puzzle games in developing CT, Lee et al. (2014) demonstrated that children can conceptualise their game play into formal algorithmic rules (i.e. explaining moves and ties in the game). Hence, the students demonstrated abstraction if they could rely on the explicit rules acquired on the game platform for their future game play.

Measure of Learner Activity. Abstraction was assessed as the percentage of game space that a learner completed when playing a puzzle game in a large-scale study involving 45 classes that examined the influence of game play on the development of CT (Asbell-Clarke et al., 2021; Rowe et al., 2021). The metric was informed by the view that identifying the puzzle game's underlying rules and the ability to extend the pattern to the rest of the game were causally related to the amount of space a player covered.

Identifying and Reading Program Code. Abstraction was also associated with the ability of students to explain the code that controlled the actors in a study that explored the impact of teaching CT using Agile software engineering methods (Fronza et al., 2017). At the end of the programme, the students were interviewed about their choice of actors, behaviour and code organisation. They received maximum points when their descriptions were specific. An average number of points was assigned for more general descriptions. Students who were unable to provide explanations received a low number of points. A similar approach was applied when abstraction was linked to the students' ability to determine the behaviour of a program from a screenshot in Scratch (Gillott et al., 2020).

Levels of Abstraction

Perrenet et al. (2005) identified four levels of abstraction in designing algorithmic solutions. However, as reported in the section Profiles of Charted Studies, none of the reviewed studies implemented abstraction at the low-execution level. In addition, apart from the shared characteristic of requiring no programming, the boundary between object and problem levels was blurry. Therefore, the implementation of abstraction in the reviewed articles was best classified into two categories – program and problem levels of abstraction.

Program-Level Abstraction. Abstraction at this level of granularity required learners to use a specific programming language to demonstrate their solutions. Examples included the investigation of students' algorithmic solution design in Scratch and Alice (Allsop, 2019) and the use of the Lego Mindstorms programming software in solving programming problems (Atmatzidou and Demetriadis (2016). Although 37.1% ($n = 23$) of the operational definitions of abstraction were classified at the program level, implementations based on text programming were visibly absent. The number of studies in this category was significant and diminished the differences between CT and programming. However, in comparison to the number of studies that implemented abstraction at the problem level (35/62; 56.45%), the result revealed a positive trend of dissociation of abstraction in CT from the strictly technical perspective of programming.

Problem-Level Abstraction. Unlike the program level of abstraction that ran algorithms in a specific programming language, the problem level was independent of any platform or language. Asbell-Clarke et al. (2021) regarded abstraction as the percentage of space a player covered in a puzzle game. This represented the problem level of abstraction, as the algorithmic pattern of covering more space in the game existed as a black box. Similarly, the studies that operationalised abstraction through Bebras tasks, such as the investigation of the effects of an after-school robotics programme on the development of CT in primary school students (Baek et al., 2019), did not require knowledge of any programming language.

Conclusion

We embarked on this review with the goal of disentangling the conceptual and operational definitions of abstraction – the core component of CT. How have researchers defined abstraction in empirical studies? To highlight the educational implications of this review, it is important to acknowledge its limitations. Our focus on studies that measured abstraction or presented a conceptual definition of abstraction when measuring CT invariably excluded reviews and theoretical studies. Although our decision to impose these limitations was guided by the goal of finding operational definitions of abstraction, we acknowledge that it is possible that some important indicators of abstraction might have been omitted. In addition, interventions in CT empirical studies measuring learning outcomes other than CT, such as learners' disposition, were potential sources for further understanding abstraction. Therefore, identifying operational definitions of abstraction solely in studies that assessed learners' abstraction may have affected the validity of our conclusions.

Despite these limitations and restrictions, we believe that we unearthed valuable information for our research programme and dispelled some of the conceptual haze around abstraction, which other researchers both within and outside the CT/CS domains may find useful. Figure 2 presents the main aspects of abstraction from the reviewed articles. The ability to identify and break down entities or processes (whether concrete vs. abstract or top-down vs. bottom-up) into their constituent components was found to be the root of abstraction. Instances of abstraction often did not fit neatly into decomposition or generalisation categories. Instead, extracting relevant features, discovering underlying patterns, creating classification rules and assembling the parts together are the core actions of learners in abstraction. At the very top of Figure 2 is simplification, which has been identified as the main purpose of abstraction.

The heterogeneity of words is a feature of human interaction. People describe phenomena from various perspectives; in turn, the multiplicity of descriptions leads to a deeper understanding. From the results of this review, the complexity of abstraction and the pros of the wide range meanings of abstraction are obvious. Moreover, the diverse ways in which abstraction has been modelled in the reviewed literature revealed that it cannot be described by any narrow set of activities. When conceiving this study, one of our initial objectives was to probe whether authors' definitions of abstraction were

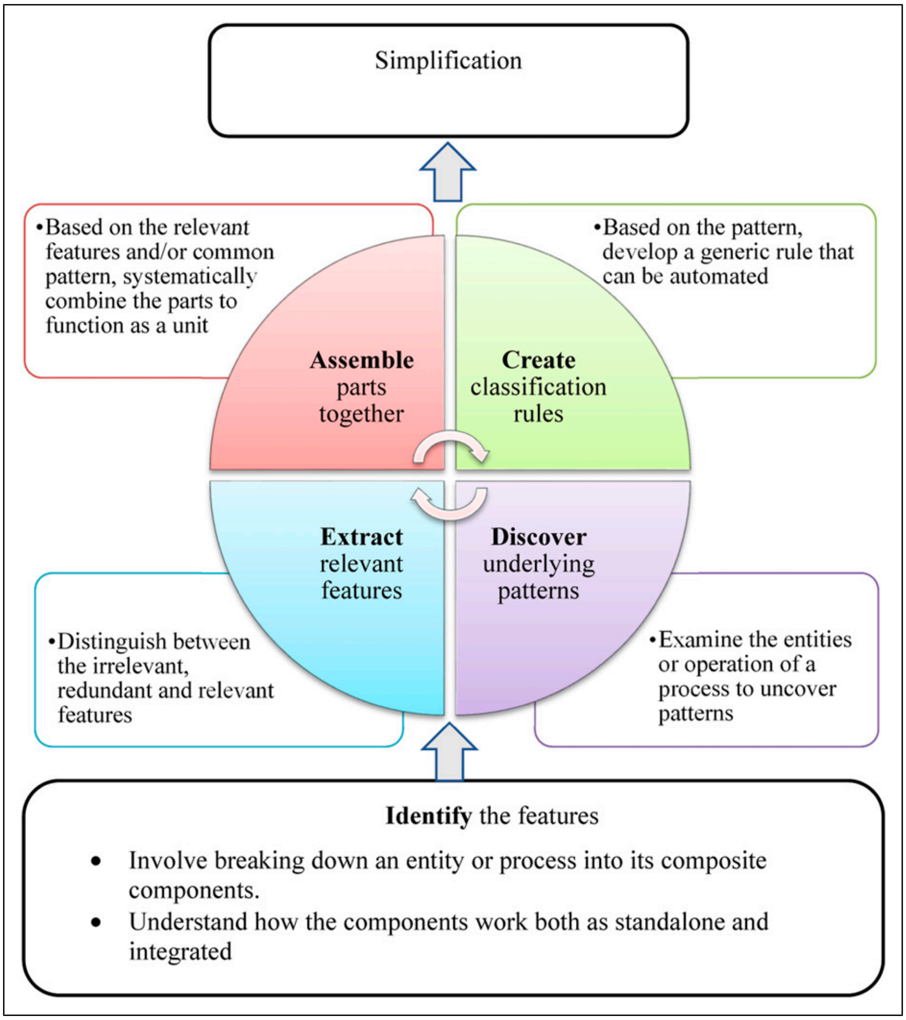
Simplification
•Based on the relevant features and/or common pattern, systematically combine the parts to function as a unit
•Based on the pattern, develop a generic rule that can be automated
Assemble parts together
Create classification rules
Extract relevant features
Discover underlying patterns
•Distinguish between the irrelevant, redundant and relevant features
•Examine the entities or operation of a process to uncover patterns
Identify the features
• Involve breaking down an entity or process into its composite components.
• Understand how the components work both as standalone and integrated

Figure 2. Summary of the core activities and purposes of abstraction.

aligned with their operationalisation. We abandoned that pursuit once it became clear that this alignment was hazy and difficult to establish. Future studies should conscientiously attempt to explicitly align both the conceptual and operational definitions of abstraction (and other CT constructs). Irrespective of how the researchers conceived abstraction, we believe that the definitions authors projected should be aligned to the ways it was operationalised. The importance of this alignment applies not only to abstraction but also to other constructs. For example, it was a similar concern about self-regulation and metacognition that led Dinsmore et al. (2008) to investigate the alignment of the constructs in their systematic review.

Finally, behind the research question that underpinned this systematic review was a humble need to know. The findings reflect what we discovered about abstraction in the empirical studies reviewed. Although we have our own theoretical orientation, which may or may not be consistent with some of our findings, the purpose of the study was never to determine the rightness and wrongness of the operationalisation of abstraction. Rather, these findings can serve as a reference for future studies, to help other researchers understand how abstraction has been implemented. Understandably, there could be some doubts as to whether certain forms of operationalisation, such as the use of programming concepts and learner activities, can actually model abstraction. Therefore, future research should investigate whether it is reasonable to consider these operational models as capable of measuring abstraction reliably.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

Ndudi O. Ezeamuzie https://orcid.org/0000-0001-8946-5709
Jessica S.C. Leung https://orcid.org/0000-0002-6299-8158

Supplemental Material

Supplemental material for this article is available online.

References

Aharoni, D. (2000). Cogito, ergo sum! Cognitive processes of students dealing with data structures. *ACM SIGCSE Bulletin*, *32*(1), 26–30. https://doi.org/10.1145/331795.331804

Alexander, P. A. (2020). Methodological guidance paper: The art and science of quality systematic reviews. *Review of Educational Research*, *90*(1), 6–23. https://doi.org/10.3102/0034654319854352

Allsop, Y. (2019). Assessing computational thinking process using a multiple evaluation approach. *International Journal of Child-Computer Interaction*, *19*, 30–55. https://doi.org/10.1016/j.ijcci.2018.10.004

Angeli, C., & Valanides, N. (2019). Developing young children's computational thinking with educational robotics: An interaction effect between gender and scaffolding strategy. *Computers in Human Behavior*, *105*(2), 105954. https://doi.org/10.1016/j.chb.2019.03.018

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Educational Technology & Society*, *19*(3), 47–57.

Asbell-Clarke, J., Rowe, E., Almeda, V., Edwards, T., Bardar, E., Gasca, S., Baker, R. S., & Scruggs, R. (2021). The development of students' computational thinking practices in elementary- and middle-school classes using the learning game, zoombinis. *Computers in Human Behavior*, *115*(25), 106587. https://doi.org/10.1016/j.chb.2020.106587

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, *75*(B), 661–670. https://doi.org/10.1016/j.robot.2015.10.008

Baek, Y., Wang, S., Yang, D., Ching, Y., Swanson, S., & Chittoori, B. (2019). Revisiting second graders' robotics with an understand/use-modify-create (U^{2}MC) strategy. *European Journal of STEM Education*, *4*(1). https://doi.org/10.20897/ejsteme/5772.

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

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12. *Acm Inroads*, *2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Basu, S., McElhaney, K. W., Grover, S., Harris, C. J., & Biswas, G. (2018). A principled approach to designing assessments that integrate science and computational thinking. In J. Kay, & R. Luckin (Eds), *Rethinking learning in the digital age: Making the learning sciences count*. International Society of the Learning Sciences. https://doi.org/10.22318/cscl2018.384

Basu, S., Rutstein, D., Xu, Y., & Shear, L. (2020). A principled approach to designing a computational thinking practices assessment for early grades. In J. Zhang & M. Sherriff (Eds.), *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 912–918). ACM. https://doi.org/10.1145/3328778.3366849

Bers, M. U. (2017). *Coding as a playground: Programming and computational thinking in the early childhood classroom*. Routledge.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Proceedings of the 2012 Annual Meeting of the American Educational Research Association. AERA, *1*, 1-25

Calderon, A. C., Skillicorn, D., Watt, A., & Perham, N. (2020). A double dissociative study into the effectiveness of computational thinking. *Education and Information Technologies*, *25*(2), 1181–1192. https://doi.org/10.1007/s10639-019-09991-3

Cetin, I., & Dubinsky, E. (2017). Reflective abstraction in computational thinking. *The Journal of Mathematical Behavior*, *47*, 70–80. https://doi.org/10.1016/j.jmathb.2017.06.004.

Chen, K.-Z., & Chi, H.-H. (2020). Novice young board-game players' experience about computational thinking. *Interactive Learning Environments*, 1–13. https://doi.org/10.1080/10494820.2020.1722712

Chiazzese, G., Arrigo, M., Chifari, A., Lonati, V., & Tosto, C. (2019). Educational robotics in primary school: Measuring the development of computational thinking skills with the bebras tasks. *Informatics*, *6*(4), 43. https://doi.org/10.3390/informatics6040043

Çınar, M., & Tüzün, H. (2020). Comparison of object-oriented and robot programming activities: The effects of programming modality on student achievement, abstraction, problem solving, and motivation. *Journal of Computer Assisted Learning*, *37*(2), 1–17. https://doi.org/10.1111/jcal.12495

Città, G., Gentile, M., Allegra, M., Arrigo, M., Conti, D., Ottaviano, S., Reale, F., & Sciortino, M. (2019). The effects of mental rotation on computational thinking. *Computers & Education*, *141*, 103613. https://doi.org/10.1016/j.compedu.2019.103613

Colburn, T., & Shute, G. (2007). Abstraction in computer science. *Minds and Machines*, *17*(2), 169–184. https://doi.org/10.1007/s11023-007-9061-7

Corral, L., & Fronza, I. (2019). A Strategy for Assessing the Acquisition of Computational Thinking Competences: A Software Engineering Approach. In I. Fronza & C. Pahl (Eds.), *Proceedings of the 2nd Systems of Assessments for Computational Thinking Learning workshop*. CEUR.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational thinking: A guide for teachers*. https://eprints.soton.ac.uk/424545/

Dagienė, V., & Sentance, S. (2016). It's computational thinking! Bebras tasks in the curriculum. In A. Brodnik, & F. Tort (Eds), *Informatics in schools: Improvement of informatics knowledge and perception* (pp. 28–39). Springer. https://doi.org/10.1007/978-3-319-46747-4_3

del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of Primary Education. *Computers & Education*, *150*(1), 103832. https://doi.org/https://doi.org/10.1016/j.compedu.2020.103832

Deng, W., Pi, Z., Lei, W., Zhou, Q., & Zhang, W. (2020). Pencil code improves learners' computational thinking and computer learning attitude. *Computer Applications in Engineering Education*, *28*(1), 90–104. https://doi.org/10.1002/cae.22177

Denner, J., Campe, S., & Werner, L. (2019). Does computer game design and programming benefit children? A meta-synthesis of research. *ACM Transactions on Computing Education*, *19*(3), 1–35. https://doi.org/10.1145/3277565

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33–39. https://doi.org/10.1145/2998438

Denning, P. J., Tedre, M., & Yongpradit, P. (2017). Misconceptions about computer science. *Communications of the ACM*, *60*(3), 31–33. https://doi.org/10.1145/3041047

Dinsmore, D. L., Alexander, P. A., & Loughlin, S. M. (2008). Focusing the conceptual lens on metacognition, self-regulation, and self-regulated learning. *Educational Psychology Review*, *20*(4), 391–409. https://doi.org/10.1007/s10648-008-9083-6

Durak, H. Y., & Saritepeci, M. (2018). Analysis of the relation between computational thinking skills and various variables with the structural equation model. *Computers & Education*, *116*(C), 191–202. https://doi.org/10.1016/j.compedu.2017.09.004

Ezeamuzie, N. O., & Leung, J. S. C. (2021). Computational thinking through an empirical lens: A systematic review of literature. *Journal of Educational Computing Research*. Advance online publication. https://doi.org/10.1177/07356331211033158

Fronza, I., Ioini, N. E., & Corral, L. (2017). Teaching computational thinking using agile software engineering methods: A framework for middle schools. *ACM Transactions on Computing Education*, *17*(4), 1–28. https://doi.org/10.1145/3055258

Garneli, V., & Chorianopoulos, K. (2018). Programming video games and simulations in science education: Exploring computational thinking through code analysis. *Interactive Learning Environments*, *26*(3), 386–401. https://doi.org/10.1080/10494820.2017.1337036.

Gillott, L., Joyce-Gibbons, A., & Hidson, E. (2020). Exploring and comparing computational thinking skills in students who take GCSE computer science and those who do not. *International Journal of Computer Science Education in Schools*, *3*(4), 3–22. https://doi.org/10.21585/ijcses.v3i4.77

Grover, S., & Pea, R. (2013). Computational thinking in K-12. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Guzdial, M. (2008). EducationPaving the way for computational thinking. *Communications of the ACM*, *51*(8), 25–27. https://doi.org/10.1145/1378704.1378713

Harimurti, R., Ekohariadi, M, Munoto, B., & IGP Asto, B. (2019). The concept of computational thinking toward information and communication technology learning. *IOP Conference Series: Materials Science and Engineering*, *535*, 012004. https://doi.org/10.1088/1757-899x/535/1/012004

Hazzan, O. (1999). Reducing abstraction level when learning abstract algebra concepts. *Educational Studies in Mathematics*, *40*(1), 71–90. https://www.jstor.org/stable/3483306

Hill, J., Houle, B., Merritt, S., & Stix, A. (2008). Applying abstraction to master complexity. In O. Hazzan & J. Kramer (Eds.), *Proceedings of the 2nd International Workshop on The Role of Abstraction in Software Engineering* (pp. 15–21). ACM. https://doi.org/doi.org/10.1145/1370164.1370169

Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, *82*, 263–279. https://doi.org/10.1016/j.compedu.2014.11.022

Jaipal-Jamani, K., & Angeli, C. (2017). Effect of robotics on elementary preservice teachers' self-efficacy, science learning, and computational thinking. *Journal of Science Education and Technology*, *26*(2), 175–192. https://doi.org/10.1007/s10956-016-9663-z

Kanaki, K., & Kalogiannakis, M. (2019). Assessing computational thinking skills at first stages of schooling. In A. Pfennig & K.-C. Chen (Eds.), *Proceedings of the 2019 3rd International Conference on Education and E-Learning* (pp. 135–139). ACM. https://doi.org/10.1145/3371647.3371651

Kert, S. B., Erkoç, M. F., & Yeni, S. (2020). The effect of robotics on six graders' academic achievement, computational thinking skills and conceptual knowledge levels. *Thinking Skills and Creativity*, *38*(1), 100714. https://doi.org/10.1016/j.tsc.2020.100714

Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly*, *81*(4), 323–343. https://doi.org/10.1080/00029890.1974.11993556

Kong, S.-C., & Lao, A. C.-C. (2019). Assessing in-service teachers' development of computational thinking practices in teacher development courses. In E. K. Hawthorne, M. A. Pérez-Quiñones, S. Heckman, & J. Zhang (Eds), Proceedings of the 50th ACM Technical

Symposium on Computer Science Education, NY, New York, 22 Feb 2019 (pp. 976-982). https://doi.org/10.1145/3287324.3287470

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior*, *72*(C), 558–569. https://doi.org/10.1016/j.chb.2017.01.005

Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM*, *50*(4), 36–42. https://doi.org/10.1145/1232743.1232745

Lee, T. Y., Mauriello, M. L., Ahn, J., & Bederson, B. B. (2014). CTArcade: Computational thinking with games in school age children. *International Journal of Child-Computer Interaction*, *2*(1), 26–33. https://doi.org/10.1016/j.ijcci.2014.06.003

Leonard, J., Buss, A., Gamboa, R., Mitchell, M., Fashola, O. S., Hubert, T., & Almughyirah, S. (2016). Using robotics and game design to enhance children's self-efficacy, STEM attitudes, and computational thinking skills. *Journal of Science Education and Technology*, *25*(6), 860–876. https://doi.org/10.1007/s10956-016-9628-2

Looi, C.-K., How, M.-L., Longkai, W., Seow, P., & Liu, L. (2018). Analysis of linkages between an unplugged activity and the development of computational thinking. *Computer Science Education*, *28*(3), 255–279. https://doi.org/10.1080/08993408.2018.1533297

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: what is next for K-12? *Computers in Human Behavior*, *41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational thinking in K-9 education. In A. Clear & R. Lister (Eds.), *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*. (pp. 1–29). ACM. https://doi.org/10.1145/2713609.2713610

Mendoza Diaz, N. V., Meier, R., Trytten, D. A., & Yoon Yoon, S. (2020). Computational thinking growth during a first-year engineering course. In A. Pears & M. Daniels (Eds.), *2020 IEEE Frontiers in Education Conference* (pp. 1–7). IEEE. https://doi.org/10.1109/FIE44824.2020.9274250.

Moher, D., Liberati, A., Tetzlaff, J., Altman, D. G., & The PRISMA Group. (2009). Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement. *PLoS med*, *6*(7), e1000097. https://doi.org/10.1371/journal.pmed.1000097

Moreno-León, J., & Robles, G. (2015). *Analyze your Scratch projects with Dr. Scratch and assess your computational thinking skills*. Retrieved 03 June, 2021 from http://jemole.me/replication/2015scratch/InferCT.pdf

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *Revista de Educación a Distancia*, *46*, 1–23. https://doi.org/10.6018/red/46/10.

Mouza, C., Pan, Y.-C., Yang, H., & Pollock, L. (2020). A multiyear investigation of student computational thinking concepts, practices, and perspectives in an after-school computing program. *Journal of Educational Computing Research*, *58*(5), 1029–1056. https://doi.org/10.1177/0735633120905605

Munoz, R., Villarroel, R., Barcelos, T. S., Riquelme, F., Quezada, A., & Bustos-Valenzuela, P. (2018). Developing computational thinking skills in adolescents with autism spectrum disorder through digital game programming. *IEEE Access*, *6*, 63880-63889. https://doi.org/10.1109/access.2018.2877417

Murphy, P. K., & Alexander, P. A. (2000). A motivated exploration of motivation terminology. *Contemporary Educational Psychology*, *25*(1), 3–53. https://doi.org/10.1006/ceps.1999.1019

Nardelli, E. (2019). Do we really need computational thinking? *Communications of the ACM*, *62*(2), 32–35. https://doi.org/10.1145/3231587

Newton, K. J., Leonard, J., Buss, A., Wright, C. G., & Barnes-Johnson, J. (2020). Informal STEM: Learning with robotics and game design in an urban context. *Journal of Research on Technology in Education*, *52*(2), 129–147. https://doi.org/10.1080/15391523.2020.1713263

Panskyi, T., Rowinska, Z., & Biedron, S. (2019). Out-of-school assistance in the teaching of visual creative programming in the game-based environment - case study: Poland. *Thinking Skills and Creativity*, *34*, 100593. https://doi.org/https://doi.org/10.1016/j.tsc.2019.100593

Papavlasopoulou, S., Sharma, K., & Giannakos, M. N. (2020). Coding activities for children: Coupling eye-tracking with qualitative data to investigate gender differences. *Computers in Human Behavior*, *105*, 105939. https://doi.org/10.1016/j.chb.2019.03.003

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

Perrenet, J., Groote, J. F., & Kaasenbrood, E. (2005). Exploring students' understanding of the concept of algorithm: levels of abstraction. In J. Cunha & W. Fleischman (Eds.), *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* (pp. 64–68). ACM. https://doi.org/10.1145/1067445.1067467

Price, C. B., & Price-Mohr, R. M. (2018). An evaluation of primary school children coding using a text-based language (Java). *Computers in the Schools*, *35*(4), 284–301. https://doi.org/10.1080/07380569.2018.1531613

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In G. Lewandowski, S. Wolfman, T. J. Cortina, & E. L. Walker (Eds), Proceedings of the 41st ACM Technical Symposium on Computer Science Education, Milwaukee Wisconsin USA, 10–13 Mar 2010 (pp. 265–269). ACM. https://doi.org/10.1145/1734263.1734357

Repenning, A., Webb, D. C., Koh, K. H., Nickerson, H., Miller, S. B., Brand, C., Horses, I. H. M., Basawapatna, A., Gluck, F., Grover, R., Gutierrez, K., & Repenning, N. (2015). Scalable game design. *ACM Transactions on Computing Education*, *15*(2), 1–31. https://doi.org/10.1145/2700517

Rijke, W. J., Bollen, L., Eysink, T. H. S., & Tolboom, J. L. J. (2018). Computational thinking in primary school: An examination of abstraction and decomposition in different age groups. *Informatics in Education*, *17*(1), 77–92. https://doi.org/10.15388/infedu.2018.05

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior*, *72*(C), 678–691. https://doi.org/10.1016/j.chb.2016.08.047

Rothstein, H. R., & Hopewell, S. (2009). Grey literature. In H. Cooper, L. V. Hedges, & J. C. Valentine (Eds), *The handbook of research synthesis and meta-analysis* (2nd ed., pp. 10–125). Russell Sage Foundation.

Rowe, E., Almeda, M. V., Asbell-Clarke, J., Scruggs, R., Baker, R., Bardar, E., & Gasca, S. (2021). Assessing implicit computational thinking in zoombinis puzzle gameplay. *Computers in Human Behavior*, *120*, 106707. https://doi.org/https://doi.org/10.1016/j.chb.2021.106707

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2019). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, *111*(5), 764–792. https://doi.org/10.1037/edu0000314

Selby, C., & Woollard, J. (2013). *Computational thinking: The developing definition*. https://eprints.soton.ac.uk/356481/

Shallice, T, Broadbent, D. E., & Weiskrantz, L. (1982). Specific impairments of planning. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *298*(1089), 199–209. https://doi.org/doi:10.1098/rstb.1982.0082

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

Sulistiyo, M. A. S., & Wijaya, A. (2020). The effectiveness of inquiry-based learning on computational thinking skills and self-efficacy of high school students. *Journal of Physics: Conference Series*, *1581*(1), 12046. https://doi.org/10.1088/1742-6596/1581/1/012046

Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. In J. Sheard & C. S. Montero (Eds.), *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (pp. 120–129). ACM. https://doi.org/10.1145/2999541.2999542

Tissenbaum, M., Sheldon, J., & Abelson, H. (2019). From computational thinking to computational action. *Communications of the ACM*, *62*(3), 34–36. https://doi.org/10.1145/3265747

Wang, D., Wang, T., & Liu, Z. (2014). A tangible programming tool for children to cultivate computational thinking. *The Scientific World Journal*, 428080. https://doi.org/10.1155/2014/428080

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

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. https://doi.org/10.1098/rsta.2008.0118

Wu, B., Hu, Y., Ruis, A. R., & Wang, M. (2019). Analysing computational thinking in collaborative programming: A quantitative ethnography approach. *Journal of Computer Assisted Learning*, *35*(3), 421–434. https://doi.org/10.1111/jcal.12348

Wu, S.-Y., & Su, Y.-S. (2021). Visual programming environments and computational thinking performance of fifth- and sixth-grade students. *Journal of Educational Computing Research*. *59*(6), 1075–1092. https://doi.org/10.1177/0735633120988807

Yadav, A., Good, J., Voogt, J., & Fisser, P. (2017). Computational thinking as an emerging competence domain. In M. Mulder (Ed), *Competence-based vocational and professional education: Bridging the worlds of work and education* (pp. 1051–1067). Springer International Publishing. https://doi.org/10.1007/978-3-319-41713-4_49

Yin, Y., Hadad, R., Tang, X., & Lin, Q. (2019). Improving and assessing computational thinking in maker activities: The integration with physics and engineering learning. *Journal of Science Education and Technology*, *29*(13), 1–26. https://doi.org/10.1007/s10956-019-09794-8

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607. https://doi.org/10.1016/j.compedu.2019.103607