A Framework for Decomposition in Computational Thinking

Peter J. Rich
Brigham Young University
Provo, Utah, United States
peter_rich@byu.edu

Garrett Egan
Brigham Young University
Provo, Utah, United States
egan.garrett@gmail.com

Jordan Ellsworth
Brigham Young University
Provo, Utah, United States
jcellswo@gmail.com

ABSTRACT

Computational Thinking has become an important cognitive skill to develop in all areas of education. Despite its increasing popularity, the construct itself is only partially understood. There are few measures currently in place that advance our understanding of computational thinking and its sub-constructs. In this article, we analyze existing measures of computational thinking (CT), looking specifically at their measures of decomposition. Decomposition is defined as the process of breaking down a problem into its sub-components. Even though most definitions of computational thinking include decomposition, few break down the decompositional process beyond a basic definition. As one of the first steps in the computational thinking process, it is important to better understand the various manners in which decomposition occurs, which methods are most effective, and under what conditions. To better understand the decompositional process, we analyze evidence of decompositional process in a variety of disciplines. We then present a framework for decomposition in computational thinking. We demonstrate how this framework may help educators to better prepare students to break down complex problems, as well as provide guidance for how decompositional ability might be measured.

CCS CONCEPTS

• **Social and professional topics → Computational thinking**; **Student assessment**; **K-12 education**.

KEYWORDS

Computational Thinking, Decomposition, Framework, Assessment

ACM Reference Format:
Peter J. Rich, Garrett Egan, and Jordan Ellsworth. 2019. A Framework for Decomposition in Computational Thinking. In *Innovation and Technology in Computer Science Education (ITiCSE '19), July 15–17, 2019, Aberdeen, Scotland Uk.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3304221.3319793

1 INTRODUCTION

Wing [33] suggested that "Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent." CT is not only the process of solving problems, but also of defining them. As educators have seen the value of the thought processes involved in CT being learned at a young age, they have sought to measure and teach CT during students' formative education worldwide [7]. Though scholars are working to improve the teaching and measurement of CT, there remains a lack of consistency in the definitions used to describe CT [22, 24, 27] . These inconsistent definitions cause measurement and teaching of CT to remain a complex challenge. The K–12 Computer Science Standards promoted by the Computer Science Teachers Association (CSTA) acknowledged this difficulty: "[d]eveloping an approach to computational thinking that is suitable for K-12 students is especially challenging in light of the fact that there is, as yet, no widely agreed upon definition of computational thinking" [11].

Despite the lack of a consistent definition of CT, there are some components that researchers agree are important aspects of CT, such as abstraction, decomposition, pattern finding, algorithm building, and debugging [11, 12].

Grover and Pea [16] asked,"What . . . can we expect children to know or do better once they‘ve been participating in a curriculum designed to develop CT and how can this be evaluated?" (p. 42). Efforts have been made to measure CT through traditional multiple-choice tests, lesson plans, analysis of computational artifacts, game design, and mobile app development. A robust measure of computational thinking ability would measure each of its sub-constructs without being conflated or confused with other sub-constructs.

One of the most consistently mentioned, but unexplored components of CT is decomposition [15, 24, 27]. Most definitions of computational thinking include decomposition as part of the process, but few provide specific details as to how decomposition is performed, ways of identifying more or less effective means of decomposing a problem, or common decompositional challenges. Considering that problem decomposition is one of the first steps of computational thinking, doing it well can potentially improve or hinder the entire CT process.

The purpose of this article is to define problem decomposition beyond a generic, "break down the problem" description [5, p. 430]. We analyze existing measures of CT, looking specifically for their measures of decomposition. We then explore examples of problem decomposition across multiple disciplines. This resulted in an emergent framework to better understand decomposition as a distinct CT skill.

2 LITERATURE REVIEW

A variety of measures, evaluations, and assessments have been developed to aid in the recognition of CT ability. These have been tested on children from four to sixteen years of age. Some measures require students to take a multiple-choice style test agnostic of a specific programming language or platform, while others involve the analysis of computational artifacts created by students. Using Google scholar and the ACM database, we searched for "measure" or “assess” and “computational thinking.” We were specifically interested in measures created since Wing‘s revival of computational thinking in 2006. This resulted in 9 measures whose specific aim was to measure evidence of CT in students or their computational products. Table 1 presents these measures, the constructs they each purport to measure, and each measure‘s specific explanation for how it measured decomposition.

This analysis demonstrates that there are many different constructs being measured by the different measures of CT. Regarding decomposition, the most common approach was not to measure it at all. When decomposition was measured, it was often measured by counting the ways in which students ‘modularized” their code. This is likely due to the fact that modularizing demonstrates a certain “breaking down” of the code into manageable “blocks.” While this may be one result of decomposition, this narrow definition reveals little about how an individual went about modularizing or decided how to modularize a block of code, and completely masks the decision-making process inherent in decomposition.

Despite the value these measures bring to the educational community in measuring CT ability, there are two areas for improvement in existing measures of CT: (a) conflation of multiple aspects of CT and (b) dependence on expert review for detection of successful decomposition. Conflation occurs when a measure combines multiple aspects of CT in a single measurement. In many cases, decomposition was paired with “abstraction” in a measure‘s rubric. This often occurred with descriptive measures that seek to identify aspects of CT through artifact analysis. In the case of objectively-measured tests, a student only receives credit for demonstrating computational thinking by providing a correct answer. The problem with this approach is that multiple CT sub-constructs may be involved in the process of coming to a correct solution. Students could use some form of cognitive shortcut or ad hoc trial and error method in order to arrive at the correct answer, which could cause false positives for CT ability.

Some measures‘ attempts to assess decompositional abilities are vague and dependent on the “you know it when you see it” notion that requires expert analysis in order to detect successful CT [8, 18]. This is not to dismiss expert review. Rather, we note that the rubrics provided in the literature do little more than state, “breaking down the problem.” This leaves a lot of interpretation up to the expert. In addition, it suggests that any one “breaking down” process is the same as any other. Yet, those same experts, when teaching computational thinking or programming, can identify more and less successful approaches by their students.

In order to have an effective and holistic measure of CT, we need an empirical means of measuring decomposition [8]. In order to develop an effective assessment of decomposition as a part of CT, we need to understand how decomposition is used as a problem solving tool. Due to the paucity of guidance provided by existing measures, we sought to discover the use of decomposition as a problem-solving strategy in various fields of interest.

3 DECOMPOSITION AS USED IN VARIOUS FIELDS

Decomposition is a common step in many problem-solving processes and is employed in a variety of fields [1, 13]. In order to

Table 1: Table 1: CT Measures with measured sub-constructs

Measure	CT Constructs Measured	Decomposition Definition
Computational Thinking test [25]	Sequences, loops, conditionals, functions, logical operations, testing and debugging„reusing and remixing, abstracting and modularizing	“Modularizing”(p. 680)
Dr. Scratch [6]	Abstraction & problem decomposition, parallelism, logical thinking, synchronization, flow control, user interactivity, data representation	“Definition of blocks” (p. 46)
Computational Thinking Patterns Test [4]	Computational concepts: sequences, loops, events, parallelism, conditionals, operators and data; Computational practices: experimenting and iterating, testing and debugging, reusing and remixing, abstracting and modularizing, Computational perspectives: expressing, connecting, and questioning	“Modularizing”(p. 7)
The Fairy Performance Assessment, [3]	Comprehension, design, complex problem solving, debugging	Not measured specifically
Computational Thinking Pattern Analysis (CTPA)	Push, pull, transportation, collision, absorption, generations, cursor control, hill climbing, diffusion.	Not measured specifically
CS4Impact [5]	Data collection, data analysis, data representation, problem decomposition, abstraction, algorithms and procedures, automation, parallelization, simulation, connection to other fields	“Students are required to break the problem down on their own.” (p. 430)
Progression of Early Computational Thinking (PECT) [28]	Procedures and algorithms, problem decomposition, parallelization and synchronization, abstraction, data representation	“Modularize…”(p. 63)
Evaluation of Computer Games [2]	Sequence, iteration, variables, conditional statements, lists (arrays), event handling, threads, coordination and synchronisation, keyboard input, random numbers, boolean logic, dynamic interaction, user interface design	Not measured specifically
Assessment of Mobile Computational Thinking, [26]	Naming, procedural abstraction, variables, loops, conditionals, lists, screen interface, events, component abstraction, data persistence, data sharing, public web services, accelerometer & orientation sensors, location awareness	Not measured specifically

better understand decomposition, we analyzed case studies from various fields of research to determine how decomposition is used, understood, and measured.

The fields we selected came primarily from their inherent connection to CT, such as STEM fields like Computer Science and Engineering. We searched Google Scholar, ACM, EBSCO, and Scopus for "problem decomposition," "problem setup," "breaking down the problem," "modularization," and related terms in articles that provided information on the decompositional process in these respective fields. We found little direct research on problem decomposition within these STEM fields. This could have occurred because of differences in term usage, such as the term "decomposition" when used in the context of Biology, and the broad usage of other terms, such as "problem-solving." We then broadened our search to include case studies that evaluated problem solving within these fields. In these case studies, we analyzed how the authors described their process for breaking down complex problems, and we identified principles of decomposition within their work.

After an initial investigation, we recognized connections between what we had found in these STEM fields with principles of foundational theoretical fields, such as Philosophy, Anthropology, and Design. In order to add to our theoretical understanding of decomposition as a problem-solving strategy, we searched for articles in these theoretical fields for information that would improve our understanding of decompositional processes in non-STEM fields.

In the following section, we present research from STEM and non-STEM fields that demonstrates how decomposition may be approached and understood from various perspectives. The goal of this section is to demonstrate the variety of ways individuals approach decomposition, as well as to extract common decompositional principles.

3.1 Decomposition in STEM

Common ways of decomposing the computational problems we analyzed in computer science were to break down the problem by its structure, function, sequence, and dependence [10, 19, 20, 31]. Structural decomposition consists of breaking down the problem into the sub-pieces and resolving each one separately. Functional decomposition breaks the problem down by which functions are performed by which pieces of the system, separating them so that each can be analyzed separately. Sequential decomposition determines the order of operations in a process or other sequence of steps, and dependence decomposition breaks down the problem by which parts are dependent on which other parts. Each of these ways of decomposing a problem appears to be more effective in specific situations, depending on what information is needed in order to resolve the problem.

Another important finding from the Computer Science literature is the concept of black box programming [14]. A black box is a device, process, or system whose inputs, outputs, and relationships to other processes or systems may be known, but whose internal structure is unknown. Most problems, prior to successful decomposition, are like black boxes, whose inner structures and relationships are unknown to the problem solver.

In engineering, math, science, and technology, we found the decomposition process simply defined as "the act of breaking something down into simpler constituents" [21]. Mattson Sorenson maintain that the two most common forms of breaking down problems and products in engineering are structural decomposition and functional decomposition. Those observations come from a product development perspective, focusing on how to creatively begin the product design process. Structural decomposition involves dividing the problem up into its physical components. For example, a bicycle could be broken down into the seat, wheels, brakes, gears, etc. Functional decomposition involves splitting the problem into sections based on what they perform. Sorensen and Mattson (ibid) suggest this strategy to be especially useful when "there is no established concept or preconceived structure" (p. 228). For example, if we were to need something to get us to the store and back, we could break it down into sections such as: time to arrive at store, top speed, steering method, etc. This example illustrates how an unsolved problem acts as a black box until it is broken down into a more understandable set of structures and functions that can be used to solve the larger problem.

3.2 Decomposition in Non-STEM Fields

In Ontology (i.e., the study of being) philosopher researchers have tried for centuries to categorize the types of beings, as well as the characteristics that define existence. Two common categories spoken of are substance and relation, the matter and how that matter compares and relates to other matter (Heidegger, 1927). Many sub-categories are also identified based on their relevance to a specific issue or field, such as place, time, state, sequence, and dependence. Place, as a form of categorizing types of beings, states that some objects are above other objects, or some objects are located on the ground. Some commonly used categories are sequence, relation, and substance or physical characteristics [23, 32**?**].

It is also important to note that philosophical understanding of problem decomposition tends to focus on breaking down a problem as a strategy for understanding the problem better [**?**]. This is similar to using dissection as an educational activity for better understanding animal anatomy and biology. This also relates strongly with the goal of opening up the black box and better understanding the inner workings of any given system, process, or device. The task of decomposition is not used as the goal, but rather the means of gaining some important new information that will inform the next steps of the problem solving strategy. James Spradley [30], a well-known ethnographer, outlined a series of semantic relationships that can be used to better understand how individuals interact within and relate to their cultures. Namely:

- Strict Inclusion - X is a kind of Y
- Spatial - X is a place in Y; X is a part of Y
- Cause-Effect - X is a cause/result of Y
- Rationale - X is a reason for doing Y
- Location for - X is a place for doing Y action
- Function - X is used for Y
- Means-end - X is a way to do Y
- Sequence - X is a step/stage in Y
- Attribution - X is an attribute/characteristic of Y

These semantic relationships are used as a way to meaningfully decompose and categorize parts of a culture into its relational sub-parts. For example, the “Location for” relationships help to describe where tasks are typically performed in a specific culture. This decompositional strategy is intended as an iterative process of successively approximating a solution, using problem decomposition as a means of identifying ways of reaching the next successful iterative step [29].

4 A DECOMPOSITIONAL FRAMEWORK

Before successful decomposition is performed, a problem or sub-problem is known only as a black box. The inputs, outputs, and other relationships between problems or sub-problems may be known, but the inner workings of the problem at hand are unknown. Decomposition helps the problem solver to better understand the inner function of the problem by unpacking the problem and separating it into multiple sub-problems. Often, when breaking down a problem into sub-problems, the sub-problems all begin as black boxes as well, with their inner structures unknown to the problem solver.

For example, in this article we have broken down CT into the component parts decomposition, abstraction, pattern-finding, algorithm-building, and debugging. By breaking down CT in this way, we better understand how it works, even if we do not yet have a complete understanding of the inner workings of each sub-construct. The next step is to break down each sub-construct to unpack those black boxes as well.

4.1 Decompositional Categorization

One of the key features of decomposition is the differentiating and categorizing of parts of a problem or sub-problem [12]. The goal of this activity is to divide the problem or sub-problem into more manageable pieces or to gain better understanding of the problem by breaking it down into its component parts.

Two general forms of decomposition emerged from our examination of the theoretical and practical examples found in the previously explored fields. Those two forms of decomposition are substantive decomposition and relational decomposition [17].

4.1.1 Substantive Decomposition. Substantive decomposition involves the breaking down of a problem or artifact by its componential characteristics, such as breaking down a jigsaw puzzle by its various pieces, a painting by the colors used, or an article by its paragraphs. These are attributes or parts of the object or idea being decomposed. When decomposing a problem, a person identifies a meaningful axis on which they can categorize parts of the problem, and then they divide and cluster those parts into the identified categories along the specified axis [15].

Axis selection is dependent on the problem statement and context. One such axis could be to break down a problem artifact by its component parts, separating connected pieces at their joints in order to understand each part individually. This is commonly used in the analysis of physical objects, such as in automobiles or human anatomy [9], but can also be used to better understand the component parts of an abstract concept, such as the breaking down of an argument into its premises, assumptions, and conclusions.

Figure 1 **??** shows the general case of substantive decomposition, where some component A is broken down into two sub-components B and C. These two sub-components are then considered separately and distinct of each other, and they provide additional information while separate that was not provided while combined.

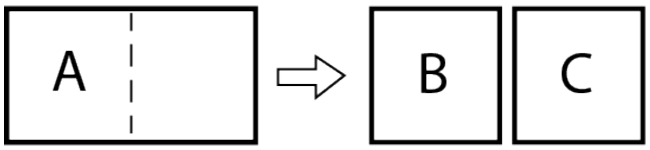

Figure 1: Substantive Decomposition Process

4.1.2 Relational Decomposition. Each component or sub-component of the problem may relate to other parts and sub-parts. Those relations could be based on time, sequence, location, dependence, function, or any of the other previously mentioned categories of relational decomposition.

Relational decomposition is performed by assigning a relation between two components or sub-components of a problem. This can be done, if desired, similar to substantive decomposition, by determining a type of relation, such as dependence, function, or sequence, and assigning all relations of that type to the various components and sub-components identified.

Figure 2 **??** shows the general case of relational decomposition, where some components A and B are assigned a relation. Prior to this relational decomposition, A and B were not considered to be related in this way.

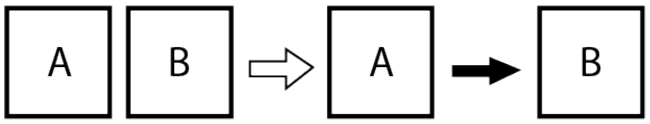

Figure 2: Relational Decomposition Process

4.1.3 Combining Substantive and Relational. Using Spradley‘s [30] set of semantic relationships listed earlier, we can better visualize how substantive and relational decompositions work together.

Substantive decomposition creates the “X” and “Y” of the semantic relationships. Relational decomposition creates the connection between the “X” and the “Y.” For example, the semantic relationship “X is a cause/result of Y” relates two substantive parts or sub-parts of a problem using a cause-effect relationship. Each substantive part or sub-part could be understood separate from the specific relational situation, but given the current circumstance, they are related in the specified manner.

4.1.4 Resolving Functional Decomposition. In Engineering, Computer Science, Design, and other fields, we observe problems frequently broken down by their functions. We believe that functional decomposition is the result of substantive and relational processes. For example, in Computer Science, a program is often broken down into a series of objects related by the functions that they perform, and how that relates those objects to the other objects in the system. Functional decomposition is understood through this framework as a combination of a substantive and relational decomposition combined to form a decompositional semantic relationship. This includes two or more components or sub-components related by a functional relationship. An example of this is shown in Figure 3 **??**, which depicts a class "Person", a class "Ball", and a relationship between them called "Kick." A function includes both substantive components and the relationship between them. This is similar to the identification of "propositions" in cognitive science, wherein memories are identified as nodes that are related to each other semantically (Anderson, 2005). Decomposing by function, then, is a parallel substantive and relational decomposition, which separates objects into their functional sub-groups and labels them with the appropriate functional relationships.

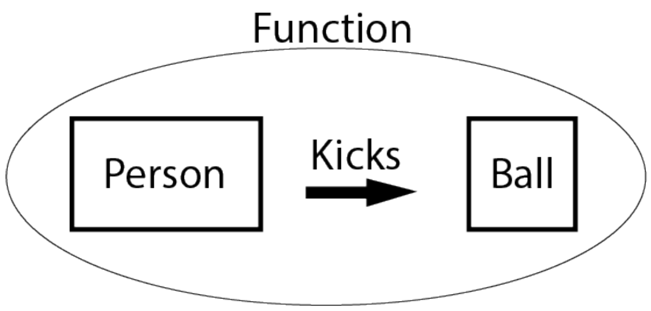

Figure 3: Functional Decomposition Process

With these two general categories of decomposition— substantive and relational—we can better understand the categorical process of decomposing a problem, and how that decompositional process helps to gain new meaningful information for solving the overall problem.

4.2 Decomposition as an Iterative Process

The act of decomposing a problem into its sub-parts appears to be an iterative process that involves a series of steps or events that contribute to the overall act of problem decomposition [29]. Similar to many design processes, such as the Engineering Design Process (EDP), instructional design processes such as the Successive Approximation Model (SAM), and others, this process includes multiple steps that can be repeated any number of times until sufficient information has been gathered to solve the problem. The generic steps we've identified are described below.

(1) *Identify an Axis*: One or more axes are identified as potential candidates to be used as part of the problem decomposition, based on their perceived potential value added toward solving the problem.
(2) *Proactively evaluate*: Each axis is evaluated for its potential to add new meaning or understanding to the problem prior to carrying at the decomposition.
(3) *Accept/Reject Axis*: Each axis is then either accepted or rejected, based on the previous evaluation. If rejected, a new axis is identified and evaluated. If accepted, the process continues.
(4) *Execute Decomposition*: The selected axis is used to categorize and label the parts and sub-parts of the problem.
(5) *Retroactively Evaluate*: The resulting decomposed version of the problem is evaluated for its ability to help solve the problem.

This decompositional process can be repeated as required in order to gain sufficient meaning and begin reconstructing the solution to the problem through patterns and algorithms. These events do not necessarily occur linearly, as various axes could be identified and decomposed in parallel to identify their related or differentiated value or meaning. This iterative approach to problem decomposition suggests that decomposition may occur at almost any point during CT. This contrasts with the belief that CT is a linear process, and that problem decomposition only occurs at one point of the process.

4.3 Decompositional Strategies

Problem decomposition is a tool that can be used as a part of a larger problem-solving strategy. We can learn many of the uses of problem decomposition as a part of CT by viewing problem decomposition through a lens of various problem-solving strategies. Below is a list of a few examples, for illustration.

- *Means-end decomposition*: Starting from the desired end solution and working backwards to break the problem into its component parts. In this approach, the last pieces decomposed become the first pieces developed.
- *Bottom-Up decomposition*: Using a specific sub-piece to identify the related axis. This most commonly occurs when a specific sub-part of the problem is known, but its relationship to the whole is unclear and relevant to resolving the whole problem.
- *Multivariate decomposition*: Using multiple axes to identify parts or sub-parts that exist in a cross-sectional category or to prove dissociation between specific parts or sub-parts.
- *Multi-level decomposition*: Decomposing the problem multiple times, categorizing sub-parts from a specific category into sub-categories.
- *Comparative decomposition*: Using multiple axes to identify similarities or differences in the clustering of parts or sub-parts based on the different axes. This can be a useful method for finding common threads in multiple arguments, definitions, etc.

5 DISCUSSION

To provide clarity for this decompositional framework, we now consider a few examples below to illustrate the practical application of this framework to computational problems.

5.1 Example 1: Coding a Queue

Our first example is of a computer science assignment to create a simple queue class.

A student might initially identify that there are component pieces required in a queue class, such as some form of data storage (array, variables, etc.) and functions to add and remove items from the queue. This is an example of substantive decomposition, identifying the parts needed in order to solve the problem.

The student might then identify that a queue adds items to the back of the line and retrieves them from the front of the line. This means that there is a sequence to the objects in the queue, as well as functional relations between the abstract queue class and the data storage object accessed by that class. This is an example of relational decomposition, identifying the ways in which the substantive parts of the problem interact and relate.

In order to organize their thoughts, the student might draw a concept map of the different objects involved in solving this problem, as well as the ways in which they relate.

A teacher could then view the students' design and assist the student to identify all of the necessary objects and relationships involved in the problem. The teacher may also assess the order in which objects/relationships were identified, grouped, and labeled, giving a more concrete way to measure successful decomposition.

5.2 Example 2: Predicting the Tides

Our second example is of an Earth Science assignment to predict the height of the ocean tides in a specific area using pre-generated data available online. A student might first identify the substantial variables that might be used to predict tide height, such as time of day, temperature, moon phase, etc. The student might then identify or test the ways these variables relate to tide height and to each other in order to identify patterns. Once the student has identified any noticeable patterns in the data, they might then mentally note the relationships between variables and remove variables that have no impact on tide height while focusing on variables that have the strongest impact. By performing this decomposition successfully, the student will be set up well to create an algorithm for accurately predicting the height of the tides. The algorithm, after all, is a series of relationships between variables that is used to predict an outcome.

Notice that some instances of decomposition occurs before pattern-finding or abstraction, while others occur after. Decomposition is also a potential part of the debugging process as well. This illustrates the iterative nature of CT, with decomposition occurring throughout the process, and not only initially in a linear fashion.

6 CONCLUSION

While there are a variety of existing measures of Computational Thinking, none of these adequately assess decomposition. By analyzing decompositional practices in a diversity of domains, we have more fully described what's inside the black box of decomposition. Namely, decomposition can be viewed of as a process of:

(1) categorizing potential elements
   (a) identifying substantive elements
   (b) identifying relationships between elements
(2) employing different strategies to execute a chosen decomposition (e.g., means-end, bottom-up, multivariate)
(3) iteratively evaluating the utility of a specific decomposition

This is a nascent framework and needs further exploration. Through highlighting these process can now ask questions about what types of categorizations or strategies do experts vs. novices utilize, or how can we more purposefully operationalize decomposition. In answering such questions, we hope to be able to formulate a more holistic measure of CT.

REFERENCES

[1] 2009. *Design science as nested problem solving.* ACM.
[2] 2012. *Evaluation of computer games developed by primary school children to gauge understanding of programming concepts.*
[3] 2012. *The fairy performance assessment: measuring computational thinking in middle school.* ACM.
[4] 2012. *New frameworks for studying and assessing the development of computational thinking.* Citeseer.
[5] 2013. *CS4Impact: measuring computational thinking concepts present in CS4HS participant lesson plans.* Vol. Proceeding of the 44th ACM technical symposium on Computer science education. ACM.
[6] 2015. *Automatic detection of bad programming habits in scratch: A preliminary study.* IEEE.
[7] 2017. *Assessing children's understanding of the work of computer scientists: the draw-a-computer-scientist test.* ACM.
[8] 2018. *Labeling Implicit Computational Thinking in Pizza Pass Gameplay.* ACM Press, New York, New York, USA.
[9] A Avolio. 1980. Multi-branched model of the human arterial system. *Medical & Biological Engineering & Computing* (1980), 709–718.
[10] P. J. Blanco, J. S. Leiva, R. A. Feijoo, and G. C. Buscaglia. 2010. Black-box decomposition approach for computational hemodynamics: One-dimensional models. *Computational Methods for Applied Mechanical Engineering* (2010), 1385–1405.
[11] CSTA. 2017. *K-12 Computer Science Standards, Revised 2017.* CSTeachers.org.
[12] J. Cuny, L. Snyder, and J. M. Wing. 2010. Demystifying computational thinking for non-computer scientists. *Unpublished manuscdript in progress* (2010).
[13] R. Davis and R. G. Smith. 1983. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence* 20, 1 (1983), 63–109.
[14] B. Du Boulay, T. O'Shea, and J. Monk. 1981. The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies* 14, 3 (1981), 237–249.
[15] D Fried, A. Legay, J. Ouaknine, and M. Y. Vardi. 2018. Sequential relational decomposition. *ACM/IEEE Symposium on Logic in Computer Science* (2018), 432–441.
[16] Shuchi Grover and Roy Pea. 2013. Computational Thinking in K—12: A Review of the State of the Field. *Educational Researcher* 42, 1 (2013), 38–43.
[17] M Heidegger. 1927. Neomarius Verlag. In *Sein und Zeit*, J. Macquarrie and E. Robinson (Eds.). Harper & Row, New York, 41–311.
[18] C. Ho. 2014. Some phenomena of problem decomposition strategy for design thinking. *Design Studies* 22, 1 (2014), 27–45.
[19] A. Leygue and E. Verron. 2010. A first step towards the use of proper general decomposition method for structural optimization. *Springer Link* 17, 4 (2010), 465–472.
[20] N. A. M. Maiden and A. G. Sutcliffe. 1996. A computational mechanism for parallel problem decomposition during requirements engineering. *IEEE* (1996), 159–163.
[21] C. A. Mattson and C. D. Sorenseon. 2017. *Fundamentals of Product Development.* Vol. 5th ed. Springer.
[22] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, Inc., New York, NY: USA.
[23] A Potesta. 2016. The fragility of the present and the task of thinking: Heidegger, thinker of the future. *Philosophy Today* 60, 4 (2016), 911–925.
[24] Noa Ragonis and Päivi Kinnunen (Eds.). 2015. *Concepts in K-9 Computer Science Education.* Vol. the 2015 ITiCSE. ACM Press, New York, New York, USA.
[25] Marcos Román-González, Juan-Carlos Pérez-González, and Carmen Jiménez-Fernández. 2017. Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior* 72 (2017), 678–691.
[26] Mark Sherman and Fred Martin. 2015. The assessment of mobile computational thinking. *Journal of Computing Sciences in Colleges* 30, 6 (2015), 53–59.
[27] Valerie J. Shute, Chen Sun, and Jodi Asbell-Clarke. 2017. Demystifying computational thinking. *Educational Research Review* (2017).
[28] Beth Simon, Alison Clear, and Quintin Cutts (Eds.). 2013. *Modeling the learning progressions of computational thinking of primary grade students.* Vol. the ninth annual international ACM conference. ACM Press, New York, New York, USA.
[29] James Spradley. 1979. *The ethnographic interview.* Harcourt Brace Jovanovich. College Publishers, Fort Worth, Texas: USA.
[30] James Spradley. 1980. *Participant observation.* Holt, Rinehart, and Winston, New York, NY: USA.
[31] A. Tarlecki, R. M. Burstall, and J. A. Goguen. 1991. Some fundamental algebraic tools for the semantics of computation: Part 3. Indexed Categories. *Theoretical Computer Science* (1991), 239–264.
[32] A. L. Thomasson. 2008. Existence questions. *Springer Science & Business Media* (2008), 63–78.
[33] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.