# Computational Thinking in Computer Science Classrooms: Viewpoints from CS Educators

Jon Good
Aman Yadav
*Michigan State University*
*United States*
goodjona@msu.edu
ayadav@msu.edu

Punya Mishra
*Arizona State University*
*United States*
punya.mishra@asu.edu

**Abstract:** Computational thinking (CT) has been described as a mental activity, a problem solving approach, and a skill fundamental to most disciplines. For teachers, the varied definitions of CT make it difficult to integrate into the curriculum. The purpose of this study was to examine how secondary computer science teachers perceive computational thinking practices and concepts in their own introductory computer science classes. Using in-depth qualitative interviews with CS teachers, we investigated how their existing curriculum was structured, their impressions of computational thinking concepts, and whether they identified computational thinking concepts within their curriculum. The results from the study suggested that computer science teachers are generally not familiar with computational thinking concepts, but when made aware of them, they find them relevant to their curricula. The findings inform CS teacher education, and refine both the theory and practice of CT in K-12 classrooms.
**Keywords:** K-12 Education, Teacher Education, Computer Science, Computational Thinking

**Keywords**
K-12 education, Teacher Education, Computer science, Computational thinking

## 1. LITERATURE REVIEW

### 1.1 Computational Thinking

#### *1.1.1 Definitions.*

With her article "Computational Thinking," Wing (2006) called upon computer science educators to examine how principles, skills, and practices from computer science can be added to every child's analytical ability. Wing emphasized that the ubiquitous presence of computing in students' lives, along with the applicability of computing to multiple fields, has established a need for every child to receive instruction in computational thinking. Wing (2008, 2010) further expanded her description of CT to include algorithms, abstraction, and automation as central components of CT, which can be found in many contexts and disciplines. Cuny, Snyder, and Wing (2010) summarized computational thinking as follows:

> 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. (as cited in Wing, 2010, p. 1)

Denning further argued that computational thinking was an expansion of earlier notions of "algorithmic thinking" from the 1950's expanded to "include thinking with many levels of abstractions, use of mathematics to develop

algorithms, and examining how well a solution scales across different sizes of problems" (2009, p. 28). Denning did not see these abstraction skills as unique to computer science, yet something with which all scientists were familiar.

Similarly, Hemmendinger (2010) argued the characterization of these skills as CS-specific to be a possible overreach, yet agreed with Wing that computing now makes it possible to "make precise various notions of complexity" of algorithms (p. 6). Computing hardware and software capabilities now allow for the automation of algorithms. Knowledge of the capabilities of technology used to enact this automation is critical in deciding whether a particular solution to a problem is feasible. This awareness and balancing of resources is part of the process of problem solving that makes CT distinct from other approaches. Yadav, Mayfield, Zhou, Hambrusch, and Korb (2014) supported Wing's view of abstraction and automation as being central to CT, with automation as a way to amplify the power of an abstraction.

*1.1.2 Components.*

Recent literature on computational thinking has shifted the focus from an overall definition to identifying concepts that can be tied to practices in elementary and secondary classrooms. Barr and Stephenson (2011) described nine core concepts and capabilities of CT, including (a) data collection, (b) data analysis, (c) data representation, (d) problem decomposition, (e) abstraction, (f) algorithms and procedures, (g) automation, (h) parallelization, and (i) simulation. They offered examples of how these concepts and capabilities could be instantiated within multiple content areas in a K-12 classroom. Grover and Pea (2013) also provided a list of nine key constructs that are central to computational thinking: (a) abstractions and pattern generalizations, (b) systematic processing of information, (c) symbol systems and representations, (d) algorithmic notions of flow control, (e) structured problem decomposition, (f) iterative, recursive, and parallel thinking, (g) conditional logic, (h) efficiency and performance constraints, and (i) debugging and systematic error detection.

The Advanced Placement Computer Science Principles course frames computational thinking as a set of six practices that complement the content knowledge of computer scientists (The College Board, 2014). These practices capture how computer scientists work within their domain, while the seven "big ideas" the course is organized around relate to fundamental concepts of computing. The CT practices are: connecting computing, creating computational artifacts, abstracting, analyzing problems and artifacts, communicating, and collaborating. Each of these practices is seen as being equally important in the practice of computer scientists. The big ideas of computer science are: creativity, abstraction, data and information, algorithms, programming, the internet, and global impact.

While there has been a move towards identifying key computational thinking concepts, the challenge for the field is to tie them to specific classroom practices as well as conduct empirical studies on how CT ideas manifest in the classroom, especially in computer science classrooms.

**1.2 Computer Science Classes as Instantiations of Computational Thinking**

With the conceptual connections and overlap between computer science and computational thinking concepts, computer science classrooms are the most likely setting where teachers and their students would encounter CT concepts. With the premise that computational thinking concepts naturally arise out of the practice and study of computer science (Hambrusch, Hoffmann, Korb, Haugan, & Hosking, 2009; Hemmendinger, 2010; Wing, 2006), it stands to reason that CS classrooms are the most likely context in which CT concepts would be observed. To discover how these concepts are used in planning for CS curricula and how CS teachers embed them, we examined practices of experienced computer science teachers teaching an introductory programming course.

We investigated (1) how computational thinking concepts were instantiated within teachers' existing introductory programming curriculum, and (2) whether they saw value of CT ideas in their curriculum and teaching after being introduced to them. The goal of this study was to discover which CT concepts a typical high school CS teacher was familiar with, intentionally embedded them within their curriculum, and the value attributed to including CT concepts in their teaching. The exploratory nature of this study lays the groundwork on the use of computational thinking concepts within teacher education efforts to train computer science teachers. Specifically, we addressed the following research question in this study.

1.3 **Research Question**

How do secondary computer science teachers view computational thinking practices and concepts being implemented in their own introductory computer science classes?

## 2. METHOD

### 2.1 Design

We used in-depth qualitative interviews with current computer science teachers to explore their views about computational thinking. Prior to the interviews, a review of the computational thinking literature was conducted to identify key components and practices of CT. The literature search through online library databases, such as ACM digital library, Google Scholar, etc. yielded a number of articles related to computational thinking. These articles were then narrowed down to prominent articles in consultation with top computer science education researchers in the field and subjected to a content analysis for common terms. The most frequently referenced terms used to describe and define CT from the content analysis were organized into a framework, seen in Table 1, which was shared with the CS teachers during the interview.

| **Computational Thinking Skills:** |
|---|
| Abstraction |
| Algorithms, heuristics, procedures |
| Automation |
| Balance limits and benefits of computing tools |
| Communication |
| Create Artifacts |
| Data Analysis |
| Data Representation |
| Group work, collaboration |
| Problem Decomposition |
| Problem Solving |
| Simulations, Modeling, Visualization |
| |
| **Computational Thinking Descriptors** |
| Complex problems, complex solutions |
| Computer Science is the source of skills |
| Computer Science education is the goal |
| Cross-disciplinary |
| Everyone can use CT |
| Mental Tools |
| Mathematical Thinking / Engineering Thinking |
| Programming |

**Table 1. CT Framework Shared in Interviews**

### 2.2 Participants

Participants were five high-school level CS teachers (3 males, 2 females) with experience teaching programming courses. Three of the five teachers taught within traditional public high schools, with students taking their programming courses as part of their regular classes. One of the teachers taught at a technical career center that students attend for half of each school day and at their regular high school for the remainder of each day. Finally, one teacher taught at a math and science magnet school with a half-day schedule similar to the career center's arrangement. All of the teachers taught an introductory programming course in their current or previous years. The teachers usually chose to use multiple programming languages/environments in their introductory courses, with choices including Scratch (4), Python (3), Java (2), C++ (2), JavaScript (1), and Alice (1).

### 2.3 Procedure

Teachers were recruited by contacting the local computer science and technology educator groups, local school districts, and colleagues. We planned to conduct interviews as long as they offered new insights into computer

science teachers' views about computational thinking and how they instantiated in their classrooms. This technique, known as data saturation helps identify whether there is sufficient data to draw inferences in qualitative research (Mason, 2010). Based on the in-depth interviews we conducted, it became clear that all of the five CS teachers conceptualized computational thinking in similar ways and saturation had been reached.

*2.3.1 Interviews.*

The interviews were conducted either face-to-face in high-school classrooms or via online audio/video conferences. The interviews ranged from 23 to 77 minutes with a mean length of 42:21 minutes (sd = 20:55 minutes). The interviews were audio recorded with the participants' consent and transcribed for data analysis. All interviews were conducted by the same researcher, who did not have a prior association with the participants. The researcher made it clear in the beginning of the interview that the purpose of the interview was to gain their insight into the current state of their curriculum, not evaluate the quality of their practice. The interviewer also identified his previous experience with computer science and familiarity with programming terms they may use.

The interview protocol had two major components. During the first half of the interview, the teachers were prompted to describe the major units, subjects, and themes of their CS curriculum. In the second half of the interview, the teachers were given a brief summary of key aspects of computational thinking developed during the literature review (see Table 1), and then were prompted to share their perspectives about CT as computer science educators. The interview protocol at this point attempted to discover any connections they saw to CT in their own curriculum, new ideas this framework presented to them, and whether they saw value in the use of these concepts in the development of CS expertise.

*2.3.2 Data analysis*

The analysis of the interview transcripts was divided into two portions. First, we coded the teachers' descriptions of their CS curriculum prior to exposure to the CT framework for how CT concepts instantiated in their classroom. Second, we coded the data for how teachers valued computational thinking in their classroom after being introduced to the CT framework.

Each of these two portions of each interview was coded for occurrences of each of the CT concepts included in the framework developed from the literature. The coding included not only explicit use of a term such as "problem solving", but also phrases that described problem solving, or agreement with interview questions that mentioned CT concepts. Table 2 includes a list of the CT concepts and a sample quote that corresponds to each concept.

Frequency analysis on the interview transcripts was conducted, using the terms from the CT framework, to identify concepts the teachers viewed as occurring within their curriculum. Some examples of the more common components and corresponding quotes are as follows: problem solving ("it's really designed to look at computer science from the perspective of how it can be used to solve problems in society"), create artifacts ("they have to make a game and it has to have an objective, keeps score, or some number of lives"), programming ("the principles like, they start in Scratch, that's their week one assignment. Scratch. And then they transition into things like Python"), and mental tools ("I want their critical thinking skills, no matter field they go in to whether it's medicine, or law, or engineering, or whatever, I want those critical thinking skills to be strong").

The teachers' reactions to the CT framework were also coded for value judgments (negative / positive / positive:high-value) regarding components of the framework and their relation to developing CS skills. This was done to discover whether certain components strongly resonated with teachers, beyond simple agreement with their applicability to CS coursework, and lastly whether some components were being ignored or not found to be applicable.

For example, a negative statement regarding programming by Teacher 2 was "very few of them want to do just heads down programming". A positive statement by Teacher 1 regarding the creation of artifacts was "they have to make a game and it has to have an objective, keeps score, or some number of lives, things like that". A statement of high-value for multiple concepts by Teacher 3 was "Some of the areas, like collaboration, problem solving, problem decomposition is [sic] a huge thing.".

| CT Concept | Sample Quotes |
|---|---|
| Abstraction | "where they can see things without actually seeing them with their eyes, they see them in their minds"; "make some connections between what they're programming, what the outcome actually is" |
| Algorithms | "algorithms"; "flow chart"; "procedural"; "step by step" |
| Automation | "Automation, of course. That's the whole point of writing code."; "you make it turn one way or the other, or you can set sequence for it to go through" |
| Balance Limits and Benefits of Computing Tools | "to see the benefits of what technology can do for them, for society"; "what can I make this do different than the last software that we were working with?" |
| Communication | "transmit data"; "communicating", "talking" |
| Complex Problems, Complex Solutions | "special projects"; "as problems get more complex" |
| Computer Science education is the goal | "getting them interested in computer science ideas"; "you could get more students in computer science" |
| Computer Science is the source of skills | "concepts that should be focused on and it sounds like a good base for the computer science principles" |
| Create Artifacts | "ending up with a product at the end, like they have a quick game or something, like a pong type game"; "app development" |
| Cross-disciplinary | "integrated more into all the classrooms" |
| Data Analysis | "data analysis" |
| Data Representation | "graphics"; "icons"; "see on the screen" |
| Everyone can use | "I want to make it and do this. I want to try that. How can I figure this out?", |
| Group Work, Collaboration | "teams"; "collaborate"; "group work" |
| Mathematical Thinking and Engineering Thinking | "algebra", "mathematical thinking", "lots of math concepts", "engineering" |
| Mental Tools | "critical thinking", "train their brains" |
| Problem Decomposition | "see a problem and break it down", "decomposed into more specific pieces" |
| Problem Solving | "problem solving", "solving a problem", "puzzle-solving" |
| Programming | "programming", "coding", "Java", "if statements" |
| Simulations, Modeling, Visualization | "simulation of bugs", "build a model", "model", "visualization" |

**Table 2. Alphabetical List of CT Skills/Descriptors with Sample Quotes from Qualitative Coding**

## 3. RESULTS

The results of the interviews and analysis are divided into three categories: pre-exposure to the CT framework, post-exposure to the CT framework, and statements of value throughout the interviews. Pre-exposure to CT includes teachers' views about aspects of computational thinking that they saw in their own curriculum before they were given the CT framework. Post-exposure, on the other hand, reflects how they were able to conceptualize their curriculum with a new perspective once teachers were introduced to computational thinking constructs. This was done to gain a picture of how teachers conceived of their curriculum without exposure to CT concepts and how their views could adjust once exposed to CT. This analysis also allowed us to gain an understanding of which CT concepts were emphasized most, and how those perceptions of value shift upon exposure to CT framework.

### 3.1 Pre-exposure to CT Framework

Two of the five teachers recognized the term of 'computational thinking', yet none were sure if they knew what it meant. All of the teachers in this study described the importance of learning to program as a key concept that they emphasized in their classroom. The teachers often represented their course in terms of which programming language they used. For example, Teacher 1 stated that ".. AP Computer Science is all Java programming. The Intro to Programming is… …the programming languages we kind of focus on are Python, and a little bit of Java.". Teacher 2 stated that "…we go into Python. So the same kinds of concepts. Always referring back to Scratch…". At least

one participant directly described the structure of their curriculum in terms of programming commands and concepts. The following comment highlighted this view as one teacher stated:

> We start "Hello World!" and then user input, variables, if statements and then loops, while, do while and for loops. From there, I think we do functions and methods. Well, actually we do conditional, so if statements and switch statements. From there I think we do arrays, then file input/output, classes and objects. Then I think sorting and searching is part of that at some point. Recursion is part of that. I think those are the main general topics for first year students. (Teacher 3)

Table 3 contains the frequencies for CT concepts mentioned by teachers prior to being introduced to the computational thinking framework. The focus on programming is reinforced given that it was the most commonly referenced concept within the CT framework. It is clear that these teachers conceptualized the teaching of CS as primarily the teaching of programming and second as the creation of artifacts. Note that the drop in frequencies after programming was substantial, going from 64 for programming to 19 for creating artifacts as the second most common concept.

| CT Concept | Pre-Framework Frequency | Post-Framework Frequency |
|---|---|---|
| Programming | 64 | 18 |
| Create Artifacts | 19 | 9 |
| Problem Solving | 13 | 27 |
| Computer Science education is the goal | 12 | 10 |
| Group Work, Collaboration | 12 | 12 |
| Mental Tools | 9 | 13 |
| Data Representation | 8 | 9 |
| Cross-disciplinary | 5 | 17 |
| Abstraction | 5 | 10 |
| Mathematical Thinking and Engineering Thinking | 4 | 12 |
| Algorithms, heuristics, procedures | 4 | 21 |
| Complex Problems, Complex Solutions | 3 | 5 |
| Communication | 3 | 10 |
| Problem Decomposition | 3 | 9 |
| Simulations, Modeling, Visualization | 3 | 7 |
| Everyone can use | 2 | 5 |
| Balance Limits and Benefits of Computing Tools | 1 | 13 |
| Data Analysis | 1 | 10 |
| Computer Science is the source of skills | 0 | 5 |
| Automation | 0 | 12 |

**Table 3. Pre-Post Exposure Frequency of CT Concepts in Interviews**

**3.2 Post-exposure to CT Framework**

After the teachers had reviewed the computational thinking definition and its key concepts, there was a shift in what teachers viewed as key CT constructs that were represented in their CS curriculum. This took the form of concepts they had already recognized in their curriculum as well as drawing connections with other CT concepts that were previously not identified. Teacher 2 immediately tied the CT framework to the Advanced Placement CS Principles course that they were planning to add to their offerings the following fall semester:

> In this course [CS principles], it's really designed to look at computer science from the perspective of how it can be used to solve problems in society and listing it up not so much from a heads down programming level, but instead, from -- so many of these things that I read on this list here. And so, more about the big picture of

> computational thinking, more about the big picture of computer science, much less emphasis on a specific programming language. And I like that.

There were multiple statements regarding how problem solving was a core skill in learning computers science. For example, Teacher 4 stated that "Problem solving, all my classes are about problem solving.". This shift in focus from programming to a broader application of computer science was mirrored again in the frequency analysis of the interview post-exposure to CT theory (see Table 3). Programming was supplanted by problem solving as the most frequently referenced concept.

Note, though, that the difference in frequency post-framework (Table 3) between problem solving, the most common code, and "Algorithms, heuristics, procedures", the second most common code, was six occurrences. In the pre-exposure portion of the interview, the difference in frequency between programming and "Create artifacts" was 45 occurrences (Table 3). This suggests that not only was programming referred to less frequently, but that the frequency among the items is more balanced between multiple concepts, rather than centered around a single concept such as "programming". The largest changes in frequency were for programming (-46 occurrences), algorithms, heuristics, procedures (+17), problem solving (+14), automation (+12) and balance limits and benefits of computing tools (+12). While the focus has broadened, the differences in frequencies suggest teachers still had some concepts that resonated more than others.

**3.3 Statements of negative value and high-value.**

Mention of negative value for a concept was not as prevalent in both pre-framework and post-framework portions of the interview when compared to the frequency of mentions of positive value. The changes from pre to post interview were minor. Programming went from one negative-value occurrence pre-framework to three negative-value occurrences post-framework, which was interpreted as too small of a change to warrant interest. This lack of criticism of programming supports the premise that interviewees were not simply attempting to please the researcher. Those trying to please the researcher would likely have made statements that were more negative about the teaching of programming, as the framework made it clear that this was a shift away from that approach.

| CT Concept | Pre-Framework Frequency | Post-Framework Frequency |
|---|---|---|
| Problem Solving | 1 | 14 |
| Group Work, Collaboration | 0 | 9 |
| Algorithms, heuristics, procedures | 0 | 7 |
| Balance Limits and Benefits of Computing Tools | 0 | 6 |
| Communication | 0 | 6 |
| Problem Decomposition | 0 | 6 |
| Computer Science education is the goal | 0 | 5 |
| Abstraction | 0 | 5 |
| Automation | 0 | 5 |
| Create Artifacts | 2 | 5 |
| Data Analysis | 0 | 5 |
| Data Representation | 0 | 5 |
| Simulations, Modeling, Visualization | 0 | 5 |
| Mental Tools | 2 | 4 |
| Complex Problems, Complex Solutions | 0 | 3 |
| Computer Science is the source of skills | 0 | 3 |
| Cross-disciplinary | 0 | 3 |
| Everyone can use | 0 | 3 |
| Mathematical Thinking and Engineering Thinking | 0 | 3 |
| Programming | 0 | 3 |

**Table 4. Pre-Post Frequencies for High-Value Statements (Ranked by Post-Framework Frequency)**

Mentions of high value, though, saw a large increase from pre-framework to post-framework. Problem solving saw the largest increase in statements of high value (see Table 4) from one occurrence in pre-framework to 14 in post-framework. The remaining top concepts for high value, group work/collaboration (9), algorithms heuristics (7), balance limits and benefits of computing (6), and communication (6), while all had zero mentions in pre-framework.

## 4. DISCUSSION

The results from the study suggested that computer science teachers, currently, conceived of their CS curriculum mainly in terms of programming and less so in terms of computational thinking ideas. Teachers in this study also mentioned that computational thinking concepts were either already present in the curriculum or that they would like to include those that were not present. Once teachers were introduced to computational thinking ideas, they were more inclined to view instantiations of CT ideas in their own computer science classrooms. Specifically, they shifted some of their conceptions of CS instruction away from programming toward more general concepts such as problem solving, algorithms, heuristics, procedures, and cross-disciplinary application

### 4.1 Pre-exposure: Programming is Central to CS Instruction

The results from the interview exhibited that the computer science teachers commonly described their curriculum in terms of programming, which suggests a focus on a domain-specific skillset. This focus on programming is understandable as the hierarchical nature of programming can map well onto a progression of topics for a teacher attempting to plan a CS curriculum. However, conceptualizing computer science curriculum around programming, rather than the underlying principles of the field, runs the risk of misrepresenting the nature of the discipline to the students. This has two potential implications; first, we miss the opportunity to teach students how to apply programming to solve problems within other contexts. Second, the conceptualizing computer science as programming might turn off traditionally underrepresented groups of students.

### 4.2 Post-exposure: Teachers Identify and Find Value in CT Components

The shift seen in positive high-value statements about computational thinking after exposure to the CT framework highlights that computer science teachers view overlap of CT in their classrooms. However, it seems that computer science teachers are not necessarily aware of what computational thinking entails.

There are clearly concepts that did not receive any statements of high value, while problem solving saw a large jump from one to 14 statements of high value. If this were merely an instance of experimenter bias, then the statements of high value would likely be more evenly distributed throughout the responses. In addition, the decline of positive statements regarding programming (-46) and creating artifacts (-10) (see Table 3) post-framework suggests a shift in focus, yet we did not see a large increase in negative statements regarding programming. The concept of problem solving clearly resonated with the teachers.

## 5. LIMITATIONS

The largest limitation of this study is in the sample size of five participants. While this number of participants is acceptable for qualitative research, it is important to not interpret these findings as making generalization claims for a larger population. Being a qualitative study at its core, intent of this study was to offer an example of how teachers may react to a CT framework and identify connections to their own curriculum, but not to make claims that the larger population of CS teachers will necessarily react in the same manner. In addition to the size of the sample, there are certainly additional research opportunities to be found in examining CS teachers at differing grade levels and types of courses taught.

## 6. IMPLICATIONS

For those in teacher development, these results are encouraging for many reasons. The existing focus on programming suggests that there is indeed a need for work to be done for those hoping to shift the focus of computer science to more of a CT-based conception of the field. The quick recognition of CT concepts and their possible value within the curriculum suggests that these concepts are welcomed by CS educators and are often already present, but teachers need to be made aware of CT and those ideas emphasized. The lack of consensus in the field about what comprises computational thinking might explain why computer science teachers might not be able to

easily identify overlap with their own course concepts. Lastly, the interest in cross-disciplinary work suggests an opportunity for exposing CS teachers to the instruction of CT concepts within non-CS subject areas.

The findings have implications for researchers examining computer science teachers' views about computational thinking in their classrooms on a large scale. Future research could also examine how to expose teachers to CT ideas and practices both within the context of a computer science classroom as well as across disciplines.

## 7. CONCLUSION

Results from this exploratory study provide evidence for how typical computer science educators conceive of CT concepts, whether they find value in relating CT concepts to CS, and whether exposure to computational thinking can shift teacher views about CS curricula. While we hope to see the study and implementation of CT beyond the CS discipline, CS classrooms are one environment where this shift can begin.

## 8. REFERENCES

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48–54.

Denning, P. J. (2009). The profession of ITBeyond computational thinking. *Communications of the ACM*, *52*(6), 28. https://doi.org/10.1145/1516046.1516054

Grover, S., & Pea, R. (2013). Computational Thinking in K–12 A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin*, *41*(1), 183–187.

Hemmendinger, D. (2010). A plea for modesty. *Acm Inroads*, *1*(2), 4–7.

Mason, M. (2010). Sample size and saturation in PhD studies using qualitative interviews. In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research* (Vol. 11). Retrieved from http://www.qualitative-research.net/index.php/fqs/article/viewArticle/1428

The College Board. (2014, June). AP Computer Science Principles Draft Curriculum Framework. Retrieved from http://media.collegeboard.com/digitalServices/pdf/ap/comp-sci-principles-draft-cf-final.pdf

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

Wing, J. M. (2010, November 17). Computational Thinking--What and Why? Retrieved August 21, 2014, from http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational Thinking in Elementary and Secondary Teacher Education. *ACM Transactions on Computing Education*, *14*(1), 1–16. https://doi.org/10.1145/2576872