Computational Thinking: A Blueprint for Modern Education

A discourse on the general agreement of the modern scientific community on what Computational Thinking is, as well as how it can be generalized to the K-12 level curriculum.

Abstract

Computational thinking is the thought process of applying the fundamentals of computer science across all disciplines. In computational thinking, a student would approach any problem the way she would approach coding and programming problems, then expand the solution to the field of her choice, including the natural sciences, humanities, and even everyday situations such as waiting in lines or packing bags. However, despite the increasing awareness regarding the importance of computational thinking, the scientific community has yet to agree on a clear definition of what this abstract concept is, much less how it can take on a more concrete form in the shape of actual programs and curricula.

This paper will distill existing resources and current research to delve into computational thinking and its applications. In addition to delineating what computational thinking has meant over time, this paper will provide an insight into its current direction, and research efforts to materialize the concept. Furthermore, by exploring different methods that have been undertaken by various computer scientists and educators, this paper will explain how computational thinking can be taught to students as a fundamental concept during their developmentally critical periods and enable it to become truly ubiquitous for all.

History

Seymour Papert

Jeannette Wing's 2006 paper Computational Thinking is widely regarded to be the starting point of modern research with regards to the subject of the paper's namesake. However, it must be noted that Seymour Papert, the co-inventor of the programming language Logo, was an initial pioneer of this field way back in the 1960s (Professor Seymour Papert), when personal computers had yet to be invented, much less distributed to the general population.

Papert's contributions are noteworthy in that he was the first person who foresaw the vast potential that computational thinking, as opposed to computers, would have on the field of education. For him, computers were means to a greater goal, a platform on which students would be able to absorb thought processes, mathematics, and computational thinking. As such, he argued that children must be steered towards the active use of computers from a young age, for pre-teen exposure to computers would be critical in computer familiarization, just as how early age exposure was crucial in fully mastering foreign languages.

Unfortunately, because personal computers had not been widely distributed until the late 90's, a lot of Papert's work had to be diverted to advocating the widespread use of computers in classrooms – a concept now known as universal computation (Papert, The Children's Machine: Rethinking School in the Age of the Computer). Whereas computers in their various shapes and sizes are so indispensable to the everyday lives of people today, it was a luxury that very few could afford back when Papert had conducted his research. And while computational thinking is a concept that is certainly distinct from – almost independent of – treating computers as physical machines, it is undeniable that preaching active education of computational thinking is quite challenging when students are not familiar with computers to begin with.

More important than Papert's efforts towards the dissemination and early exposure to computers is the fact that as a pioneer in the field of education, Papert was convinced that computers could be used as an effective medium to teach what he considered to be the most important concepts – learning and thinking. The language that he co-developed, Logo, was the result of his belief that computers could be used to visualize geometric concepts and facilitate the understanding of algorithms – a process that he referred to as "body-syntonic reasoning". (Papert, Mindstorms). With simple commands on a GUI (graphic user interface), the students would be able to move a small robot called the turtle, through which they would have an enhanced understanding of mathematics by envisioning themselves as the turtles on screen. Using a mix of turtle commands (ex: forward, back, left, right), math operations (ex: sum, difference, product, quotient, remainder, trigonometric functions), boolean operations (ex: less, greater, equal, and, or, not), and control structures (ex: if, repeat), the students were able to create a wide range of geometric shapes, ranging from the most rudimentary ones such as triangles and circles to complex ones including cubes, floral patterns, and infinitely branching-out trees (DiSessa and Abelson, Turtle Geometry).

Jeannette Wing

While early researchers such as Papert or Alan Perlis are credited with pioneering the field of computational thinking, it is Jeannette Wing who invigorated the research community and gave new life to an otherwise outdated – and somewhat forgotten – concept. Her paper, Computational Thinking, was a wake-up call to not only the scientific community but to the general population with regards to the importance of computer science, especially in the wakes of the dot-com bubble bust of the early 2000s. A computer scientist herself, Wing considered computer science to be a key component of modern education, not because of its contributions to software development and engineering, but because of its contributions to the thought processes of children. Just as how reading, writing, and mathematics composed of the three R's of modern education, she vouched for the inclusion of Algorithms ('Rithms) as the fourth R that would supplement the existing pillars of reading, writing ('Riting), and arithmetic ('Rithmetic).

Now that universal computation, which Papert had so strongly advocated, was something that modern educators could take for granted, Wing sought to build on that concept and make the fundamentals of computer science something that even non-majors could learn from. Computational thinking does not aim to make humans think like computers; rather, it is a way of problem solving (Wing, Computational Thinking). The very concepts that are used to solve software design problems, such as simulation, parallel processing, abstraction, and decomposition, could be used to determine the fastest way to rotate restaurant tables or optimize the number of supermarket check-out lines. Even better, the same concepts could be used for even the seemingly most mundane, everyday acts such as planning groceries, because that is how ubiquitous computational thinking is. To put it in her own words, computational thinking is for everyone, everywhere (Wing, Computational Thinking).

10 years after her groundbreaking article was published, much progress had been made, most notably with respect to the skyrocketing interest in computer science – the highest level since the dot-com bubble burst (Wing, Microsoft Research Blog). Due to increased funding in computer science education, many countries have started to include computer science in their basic K-12 curricula. Even outside of normal school settings, many non-profit organizations – most notably code.org – took off with the initiative of increasing awareness and interest in computer science, especially within populations that had previously been considered as minorities in the STEM field, such as women and underrepresented ethnicities (Partovi).

While Wing has had a huge impact on laying out the definition and framework for what computational thinking is, she has not been an active pioneer when it comes to implementing concrete versions of CT curricula. Nevertheless, she has presented important research questions that have yet to be answered, such as how to best harness the advent of ubiquitous computation, and when/where certain concepts should be taught – for example, when is the best time to introduce recursion to students? (Wing, Microsoft Research Blog) And as academia, industry, and government work towards the common goal of making computational thinking a truly universal education concept, the blueprints laid out by the very person who revitalized the concept will be key to future endeavors.

Definition of CT

Scientific Community

Grover and Pea put it best when they summed up computational thinking in a single phrase: *CT's essence is thinking like a computer scientist when confronted with a problem* (Grover and Pea). The beauty of this definition is that it embodies the very objective of computational thinking, which is to make the fundamentals of computer science available for all disciplines and even everyday life – thus the generic term "problem". The details to this approach have been further clarified by Alfred Aho, who stated that computational thinking is the thought process involved in formulating problems so that their solutions can be represented as computational steps and algorithms (Aho) (Grover and Pea).

Many others, such as the Royal Society, National Science Foundation(NSF)/College Board, Computer Science Teachers Association (CSTA), and the International Society for Technology in Education (ISTE), have offered their perspectives on how computational thinking should be defined (Grover and Pea). These include explanations that cover the seven big ideas of computing (Snyder), the operational definitions of computational thinking (Barr and Stephenson), and the importance of recognizing aspects of computation in the world that surround us, as well as applying tools and techniques from computer science to understand natural and artificial systems/processes (The Royal Society).

Personal Opinion

While multitudes of different definitions exist, the one put forth by Grover and Pea captures the essence of what computational thinking is and should be, in that it emphasizes the problem-solving process, which is the single most important aspect of CT. Perlis, Papert, and Wing have all argued that computational thinking is a modern literacy that is as important as reading, writing, and arithmetic. The fact that CT is considered as the fourth pillar of education shows how CT is not just an end goal by itself; rather, it is the means to facilitate everyday decisions, improve students' learning capabilities, and advance research across all disciplines, including the non-STEM fields.

While people may not be familiar with terms like abstraction, modularizing, backtracking, or concurrency, they act upon such concepts in everyday life so naturally that they can almost be perceived as a part of subconscious human behavior. As such, computational thinking is also a process of theorizing and formalizing an already widespread concept, one that has been around for decades, if not centuries. Wing argues that a student packing her bag for school is prefetching and caching, that a child looking for her lost items is backtracking, and that standing in supermarket lines is multi-server system performance

(Wing, Computational Thinking). And while she may be taking a huge and rather generalized logical step, her point that such ideas are already ingrained into people's behaviors is a very valid one. Despite the majority of the population not having even heard of what it is, computational thinking is already very much a part of everyday life.

But as general as CT is, the fact that modern schools face a severe lack of qualified instructors and curricula is still an outstanding issue. As such, researchers in recent years have now been shifting their focus from defining what computational thinking is to implementing computational thinking through various forms of curricula, both in and out of classroom settings. Due to how widespread the underlying concepts of computational thinking are in our society, the issue is less about formalizing what computational thinking is, and more about the methodologies that are used to educate students on CT-related concepts.

When it comes to training students to adapt to computational thinking processes, Andrea DiSessa's prescient observation of computational literacy plays an important role in providing the guidelines for future research (Grover and Pea). DiSessa separates computational thinking into two main aspects: the material side, which includes the programming tools and environments that provide hands-on experience to students, and the social side, which covers the thought processes that are behind computational thinking (DiSessa). This division is important, as the material aspect of CT is very dependent on external factors, such as the availability of computers or the internet, while the social aspect places more emphasis on the content that is taught through various channels – sometimes even forgoing the use of computers altogether, most notably by the pioneering efforts of CS Unplugged (CS Education Research Group, University of Canterbury).

The following sections will introduce currently ongoing research in tools and curricula used to introduce CT to K-12 students, as well as future directions that CT research could take in the coming years.

Current Applications

Tools

Building on Alan Perlis' belief that everyone should learn to program as part of a liberal education (A. Perlis), Mark Guzdial explored research projects that are being conducted to make computational thinking a 21st century literacy for all (Guzdial). Most notably, he examined the relationship between modern programming languages and students' understanding of basic computer science concepts by introducing a research carried out by Lance A. Miller.

Miller had discovered that object-oriented programming (OOP), the backbone of most modern programming languages such as Java, Python, and C++, was not something that non-computing students could intuitively understand (Miller). OOP has many advantages, including code reuse/recycling, encapsulation, design benefits, and software maintenance (Zoski and Salvage). However, the tradeoff of having such features is that intuition is somewhat sacrificed to make room for additional functionality. Many non-computing students have a hard time grasping the idea of classes and instances, much less advanced concepts such as inheritance and polymorphism, which are key components of object-oriented languages.

Moreover, Miller learned that control structures like the if-else statement were another point of difference between the major and non-major students, as non-computing students often entirely ignored the else- part of the control flow. For them, in the situation where the if-condition was not satisfied, the next obvious step of action would be to just move on to the next block of code. He notes that it is "easier for the novices trying to read those programs if the conditions for each clause's execution are explicit" (Guzdial), which is unfortunately not the case in most modern languages.

These findings came to play significant roles in the revamping of modern computer science education. Scratch and Alice, two of the most popular tools/languages that are currently being used for K-12 computer science, are constructed to be event-based languages as opposed to object-oriented. And while these tools are inherently devoid of some of the more advanced features that are found in more conventional programming languages, they are surprisingly flexible, as is evident in the vast array of different user-created projects that can be found in the Scratch online community, ranging from an obstacle-based game with three different levels to a program in which the main character moves in synchronized motion with the background music (Brennan and Resnick).

The holy grail of these entry-level programming tools and languages is to have a low floor, high ceiling, and wide walls. Because abstraction is one of the harder concepts to grasp in computer science, many of these languages feature characters and items that anyone can recognize, as opposed to amorphous classes and instances. By constructing commands for these characters to execute, such as traversing a certain path or interacting with other characters, students are introduced to the concept of algorithms, which is just a step-by-step instruction that leads to the desired result.

One of the goals of CT is to help students understand how to think programmatically without burdening them with esoteric computer code syntax. For example, a layperson would have a hard time understanding the for-loop syntax of Java or C++ code; this is why many modern Initial Learning Environments (ILE's) have graphic drag-and-drop functionalities that are further supported by text-box entries that users can use to type in variables. This is a much easier way for students to learn the basics of programming, especially considering how many of these students are still in their developmental stages of learning and therefore respond better to visual stimuli. Moreover, these tools are designed so that even the most inexperienced students can understand how the program works almost intuitively; this means replacing formal control structures such as the for-loop with simple statements, like *repeat-N-times*. Since each completed command is visualized on the screen in a user-friendly format, the student receives immediate feedback and reinforces her understanding of the underlying structures and algorithms, just as Papert's Logo was designed to be.

On the other end of the spectrum, these languages also need to let more experienced users be able to implement advanced features that are well beyond simple movements and commands. The Logo language was very intuitive and had a low entry bar in terms of difficulty, but it was lacking in that there was only so much that the user could do outside of drawing various geometric shapes. Modern tools enable the user to do so much more – for example, the Alice language creates a 3-D environment complete with a plot and multiple characters, so that the user is introduced to programming concepts while exploring a fictional storyboard (Werner, Campe and Denner, Children Learning Computer Science Concepts via Alice Game-Programming).

And finally, modern tools need to give students a wide range of choices when it comes to the type of program she wants to create. Just as conventional languages are used across all fields for different purposes, CT tools are expected to have the same range of possible implementations, which requires a great deal of flexibility as well as a multitude of different bells and whistles. Many platforms have various forms of media support, such as image, music, and video animations, so that students can use data structures and algorithms satiate their needs and imaginations. The advent of the internet has also encouraged the rise of multi-user platforms where students can share their projects and activities, most notably those in Scratch or Khan Academy.

Over the past few years, the rapid growth of mobile platforms has fostered the creation of puzzle-like games that are designed to introduce computational thinking to a younger audience. Still others such as Google Research have taken the concept of hands-on education to another level by creating hardware blocks that correspond to variables, mathematic notations, and control structures through experiments such as Project Bloks (Google) (Bilkstein, Sipitakiat and Goldstein). In tandem with computational thinking curricula, modern tools and languages are playing an indispensable role in this renaissance of computer science education.

Curricula

Teaching computational thinking to computer science majors is not too big of a challenge, considering how the students are already familiar with many of the concepts through their introductory courses in data structures and algorithms, as well as the tools and languages that are used throughout their classwork. After all, it is easy for those with even a small amount of computer science experience to see how long division, in which the numerator is constantly divided by the denominator (the numerator is then sequentially updated after each iteration of the division), is in fact an algorithm that students have been using ever since they first learned how to take a quotient of two numbers.

It is the non-computing students that struggle to grasp computational thinking, not just because the terms and concepts are rather alien, but also because most schools today do not have enough adequate human capital and structured curricula to introduce computer science, much less computational thinking. While the concepts themselves are not inherently difficult to understand, it takes a certain amount of time for non-computing students to see how simple algorithms like long division or Fibonacci sequences are not only shared with concepts in computer science, but in fact derive from its very foundations.

This is where important questions that CT researchers are currently facing arise. How is computational thinking different from other forms of thinking, such as mathematic thinking? At what point of students' academic progress do we teach them a given concept? Is computational thinking really necessary for everyone, even for students who are not pursuing STEM degrees? (Wing, Microsoft Research Blog) (Grover and Pea). With the rise in global recognition of the importance of CT, these are the questions that must be answered for CT to gain further momentum and truly become a modern literacy for all.

With regards to the difference between computational thinking and other similar forms of thinking, while it is true that CT has overlap with preexisting forms of thinking, such as arithmetic, it is unique in that it takes real-world constraints into account, just as an engineer would (Wing, Computational Thinking). Now that universal computation has been realized, it is important for people to keep in mind that as potent as they are, computers have their share of limits as well, such as their capabilities, computing power, and operating environment. Recent developments in computer graphics, simulations, and virtual reality have somewhat freed humankind from certain constraints, in that people are now able to build their own worlds. However, this does not mean in any way that we have also been freed from the constraints of natural laws and physics; rather, it grants us the ability to build systems that closely resemble physical conditions. Computational thinking reminds students that these constraints must be taken into account in their planning stages.

Another question is finding the right age at which students can learn a given concept. While many countries have different opinions on what general computational thinking skills are and how they can be developed in students, the fundamental question of "How young is too young?" has yet to be answered definitively by researchers (Duncan and Bell, A Pilot Computer Science and Programming Course for Primary School Students). According to the Progression of Early Computational Thinking model, put forth by Seiter and Foreman, certain patterns are best suited for certain grades: students in early stages of elementary school could explore shape configuration, sprite interaction, and a small amount of animation/synchronization, while those with at least 4+ years of elementary education could start learning selection or controlled repetition involving variables and booleans (Seiter and Foreman) (Duncan, Bell and Tanimoto, Should your 8-Year-Old Learn Coding?). And while students have varying degrees of comprehension, there is a limit to the level of abstraction for many students in their primary stages of education (Werner, Campe and Denner).

Through trial and error that has lasted for hundreds, maybe even thousands of years, educators generally tend to have a good idea of when students should be taught addition, multiplication, functions, and calculus. However, since computer science is a much younger subject compared to its contemporary counterparts such as mathematics, reading, and writing, the scientific community has yet to agree on a structured timetable for computer science education. Fortunately, with increased computer supply in classrooms and the heightened global awareness on the importance of computational thinking, researchers in the near future will have more opportunities to arrive at an optimal conclusion.

One of the most important unanswered questions is on whether computational thinking is something that every student needs. Here, it must be noted that CT is not meant to teach students to become software engineers; rather, it is an effort to make the fundamental concepts of computer science available for all disciplines outside of computer science. Therefore, it is not just a matter of question, but absolutely imperative that even students who are vested in non-STEM subjects be exposed to CT. The widespread use of technology in the 21st century has created a rather odd phenomenon where students are "taught how to use software to write, but not how to write software." (Duncan, Bell and Tanimoto, Should your 8-Year-Old Learn Coding?) In the interdisciplinary world that we live in nowadays, the boundary between different subjects is becoming weaker and more meaningless with each passing of time. Just as elementary education – and the notion of liberal education in many higher-level institutions – values well-rounded students, computational thinking will be yet another tool that students will be able to use, regardless of what fields and professions they end up choosing in the long run.

To ask whether non-STEM students should also invest time in learning computational thinking would be to misunderstand the whole point of CT, because it is a concept that is especially important for non-STEM students who do not have exposure to the mathematic or engineering approaches that their

STEM counterparts have honed throughout their studies. In more recent decades, computer science has not only advanced science, but also the seemingly most irrelevant fields such as journalism, history, or philosophy, due to concepts like big data, automation, or artificial intelligence. As such, CT will be a key bridge that helps students understand the far-reaching influence that technology has on the modern world.

Conclusion

Almost 40 years ago, Papert argued that children of almost any age could learn how to program under good conditions, plenty of time, and powerful enough computers. However, he also noted that providing such conditions and environment would be a notable challenge in advancing the field of computational thinking (Papert, Mindstorms).

Decades have passed since his prescient vision, and many things have changed. Tech stars such as Mark Zuckerberg or Evan Spiegel receive the same accolades that a TV star or a Billboard musician would. Universal computation has helped the general population realize just how important technology – and its underlying building blocks – is to every aspect of this modern world, as evidenced by the rebirth of Silicon Valley that has well surpassed its pre-dotcom bubble days. Nontheless, the same problem that Papert had foreseen, back when personal computers had not even been developed yet, still continues to challenge the research and education communities.

Despite the increasing importance of – and interest in – computational thinking, reality is that educators are having a hard time teaching these valuable concepts to students. The hacker phenomenon, cultural factors, and the lack of qualified teachers are just some of the many reasons why CT still has a long way to go. (Duncan, Bell and Tanimoto, Should your 8-Year-Old Learn Coding?). Recent developments in tools and curricula that foster computational thinking are taking strides to make CT a 21st century literacy, but they are still in their baby steps, still waiting for a breakthrough moment.

The demand for computational thinking will only become more pronounced with time, as it is the very backbone of this ongoing tech boom that does not seem to be stopping any time soon. Computational thinking is not just the future; it is already very much an integral part of people's everyday lives, abrupt as it may be. How we harness this powerful tool will decide whether this is just yet another bubble, a wasted opportunity, or a stepping stone to a modern renaissance that will change the way we live – or better, the way we think.

Works Cited

Aho, Alfred. "Computation and Computational Thinking." *Computer Journal* 55 (2012): 832-835.

Barr, Valerie and Chris Stephenson. "Bringing Computational Thinking to K-12: What is Involved and What is the Role of Computer Science Education Community?" *ACM Inroads* (2011): 48-54.

Bilkstein, Paulo, et al. "Project Bloks: Designing a Development Platform for Tangible Programming for Children." *Research at Google* (2013).

Brennan, Karen and Mitchel Resnick. "New Frameworks for Studying and Assessing the Development of Computational Thinking." *AERA* (2012): 1-25.

CS Education Research Group, University of Canterbury. *Computer Science Unplugged*. n.d. 11 December 2016. <http://csunplugged.org/about/>.

DiSessa, Andrea and Hal Abelson. *Turtle Geometry*. MIT Press, 1981.

DiSessa, Andrea. *Changing Minds: Computers, Learning, and Literacy*. MIT Review, 2001.

Duncan, Caitlyn and Tim Bell. "A Pilot Computer Science and Programming Course for Primary School Student." *The 10th Workshop in Primary and Secondary Computing Education* (2015).

Duncan, Caitlyn, Tim Bell and Steve Tanimoto. "Should your 8-Year-Old Learn Coding?" *The 11th Workshop in Primary and Secondary Computing Education* (2016): 60-69.

Google. *Project Bloks*. 2013. 7 December 2016. <https://projectbloks.withgoogle.com/>.

Grover, Shuchi and Roy Pea. "Computational Thinking in K-12: A Review of the State of the Field." *Educational Researcher* (2013): 38-43.

Guzdial, Mark. "Paving the Way for Computational Thinking." *Association of Computing Machinery* (2008): 25-27.

Miller, L.A. "Natural Language Programming: Styles,Strategies, and Contrasts." *IBM Systems Journal* (1981): 184-215.

Papert, Seymour. *Mindstorms*. New York: Basic Books, 1980.

—. *The Children's Machine: Rethinking School in the Age of the Computer*. New York: Basic Books, 1993.

Partovi, Hadi. *Code.org*. January 2013. 5 December 2016. <https://code.org/>.

Perlis, A.J. "Programming of Digital Computers." *Communications of the ACM* (1964): 210-211.

Perlis, Alan. *Computers and the World of the Future*. March 1964. 5 December 2016. <https://mitpress.mit.edu/books/computers-and-world-future>.

Professor Seymour Papert. n.d. <http://www.papert.org/>.

Seiter, Linda and Brendan Foreman. "Modeling the Learning Progressions of Computational Thinking of Primary Grade Students." *Proceedings of the ninth annual international ACM conference on International computing education research* (2013): 59-66.

Snyder, Larry. *An Open Letter to the Computer Science Community*. 2010. 10 December 2016. <https://csprinciples.cs.washington.edu/sevenbigideas.html>.

The Royal Society. "Shut down or restart? The way forward for computing in UK schools." *Royal Academy of Engineering* (2012).

Werner, Linda, Shannon Campe and Jill Denner. "Children Learning Computer Science Concepts via Alice Game-Programming." *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (2012): 427-432.

—. "The Fairy Performance Assessment: Measuring Computational Thinking in Middle School." *Special Interest Group on Computer Science Education* (2012).

Wing, Jeannette. "Computational Thinking." *Communications of the ACM* (2006): 33-35.

—. "Microsoft Research Blog." 23 March 2016. *Microsoft Research Blog.* 5 December 2016. <https://www.microsoft.com/en-us/research/blog/computational-thinking-10-years-later/>.

Zoski, Paul and Jeff Salvage. *Object Oriented Programming: Advantages of OOP*. 2015. 7 12 2016. <https://www.cs.drexel.edu/~introcs/Fa15/notes/06.1_OOP/Advantages.html?CurrentSlide=3>.