Capítulo 1

Pensamento computacional

Em um artigo seminal, no ano 2006, a professora da Universidade de Columbia, Jeannette Wing, formaliza a noção de "Pensamento Computational".

Entendido como a uma abordagem para resolução de problemas, este conceito se propõe a abranger um conjunto de habilidades e práticas não apenas relevante para programadores ou cientistas da computação, mas passível de compor até mesmo as faculdades analíticas de qualquer criança (Wing, 2006).

Em linhas gerais, conforme sintetiza em artigo de 2010, "[...] o pensamento computacional descreve a atividade mental na formulação de um problema para admitir uma solução computacional. A solução pode ser realizada por um humano ou máquina, ou mais geralmente, por combinações de seres humanos e máquinas"(Wing, 2010, tradução nossa).

O uso da palavra "computacional"busca também evidenciar a relação dessa abordagem com elementos da ciência da computação.

When I use the term computational thinking, my interpretation of the words "problem" and "solution" is broad; in particular, I mean not just mathematically well-defined problems whose solutions are completely analyzable, e.g., a proof, an algorithm, or a program, but also real-world problems whose solutions might be in the form of large, complex software systems. Thus, computational thinking overlaps with logical thinking and systems thinking. It includes algorithmic thinking and parallel thinking, which in turn engage other kinds of thought processes, e.g., compositional reasoning, pattern matching, procedural thinking, and recursive thinking. Computational thinking is used in the design and analysis of problems and their solutions, broadly interpreted. The most important and high-level thought process in computational thinking is the abstraction process. Abstraction is used in defining patterns, generalizing from instances, and parameterization. It is used to let one object stand for many. It is used to capture essential properties common to a set of objects while hiding irrelevant distinctions among them. For example, an algorithm is an abstraction of a process that takes inputs, executes a sequence of steps, and produces outputs to satisfy a desired goal. An abstract data type defines an abstract set of values and operations for manipulating those values, hiding the actual

representation of the values from the user of the abstract data type. Designing efficient algorithms inherently involves designing abstract data types. Abstraction gives us the power to scale and deal with complexity. Recursively applying abstraction gives us the ability to build larger and larger systems, with the base case (at least for computer science) being bits (0's and 1's). In computing, we routinely build systems in terms of layers of abstraction, allowing us to focus on one layer at a time and on the formal relations (e.g., "uses," "refines" or "implements", "simulates") between adjacent layers. When we write a program in a high-level language, we do not worry about the details of the underlying hardware, the operating system, the file system, or the network; furthermore, we rely on the compiler to be a correct implementation of the semantics of the language. As another example, the narrow waist architecture of the Internet, with TCP/IP at the middle, enabled a multitude of unforeseen applications to proliferate at the highest layer, and a multitude of unforeseen hardware platforms, communications media, and devices to proliferate at the lowest.

Computational thinking draws on both mathematical thinking and engineering thinking. Unlike in mathematics, however, our computing systems are constrained by the physics of the underlying information-processing agent and its operating environment. And so, we must worry about boundary conditions, failures, malicious agents, and the unpredictability of the real world. But unlike other engineering disciplines, because of software (our unique "secret weapon"), in computing we can build virtual worlds that are unconstrained by physical reality. And so, in cyberspace our creativity is limited only by our imagination.

Nos anos seguintes à publicação do artigo, um sem-número de debates em torno do tema tomaram os forums de educação na Europa e Estados Unidos.

Computer science, it argues, is neither programming nor computer literacy. Rather, it is "the study of computers and algorithmic processes including their principles, their hardware and software design, their applications, and their impact on society"

no widely agreed upon definition of computational thinking.

As the International Working Group on Computational Thinking [8] pointed out, however, computational thinking "shares elements with various other types of thinking such as algorith-mic thinking, engineering thinking, and mathematical thinking". Perkovic et al. [10] similarly focus on the intellectual skills neces-sary to "apply computational techniques or computer applications to ... problems and projects" in any discipline. Hemmendinger [6] notes that we must be aware of the risks of arrogance and over-reaching when discussing the role of computational thinking, es-pecially across disciplines. He argues that the elements of compu-tational thinking that computer scientists tend to claim for their own (constructing models, fi nding and correcting errors, creating representations, and analyzing) are shared across many disciplines and that the appearance of grand territorial claims risks provoking adverse reactions. Hemmendinger concludes that the ultimate goal should not be to teach everyone to think like a computer scientist, but rather to teach them to apply these common elements to solve problems and discover new questions that can be explored within and across all disciplines.

CT is an approach to solving problems in a way that can be s Employ diverse learning strategies. implemented with a computer. Students become not merely tool users but tool builders. They use a set of concepts, such as ab- The dispositions and pre-dispositions category arose from straction, recursion, and iteration, to process and analyze data, an attempt to capture the "areas of values, motivations, feelings, and to create real and virtual artifacts. CT is a problem solving stereotypes and attitudes" applicable to computational thinking. methodology that can be automated and transferred and applied These included: s across subjects.