COMPUTATIONAL THINKING IN ENGINEERING

Osvaldo Luiz de Oliveira[1]

Abstract — *"Computational thinking" is an expression that has been used to describe the human and machine reasoning. Research results suggest that, like reading, writing and calculating, which are fundamental skills every child learns, computational thinking is a skill needed for every person in today´s world. Computational thinking involves, among other things, problem solving, systems development, abstraction, and is the core of all modern science and technology. When an engineer calculates, develops a project or a work implementation schedule, establishing a logical and interdependent relationship between different activities, he is thinking computationally. In this paper, we present the principles of computational thinking, give examples, and discuss the challenges of providing a computational thinking discipline to Engineering.*

Index Terms — Computing Education in Engineering, Computational Thinking.

INTRODUCTION

The roots of the term Computing refer to calculate, count, determine anything by calculation. The advent of computing machines has established the need for a Computer Science to organize and develop a body of knowledge to a wide variety of phenomena and topics such as problem solving, development of algorithms, design, computer programming, theoretical models for the notion of Computing, among many others.

In the 1960s, the increasing use of computers led many universities to introduce concepts of computer programming in Engineering, Physics and Mathematics, to name only a few careers. The idea was that these professionals acquired the ability to develop algorithms as a way to program the computer to solve problems, such as of calculus, statistic, mechanics, machine design and structures. Computing disciplines taught the development of algorithms in languages such as FORTRAN, C and Pascal.

The popularization of personal computers, during the beginning of the 1980s, was accompanied by software for the automation of activities such as text editing, elaboration of spreadsheets, presentations, and specific applications for technical and artistic drawing, Engineering projects, sound processing, music, etc. Many universities have developed disciplines to study the use of specialized software, such as computer aided design, virtual electronics lab, geographic information systems, and programs for office automation and communication online. Although useful, these disciplines are focused on operations, in the functionality and use of the interface language required to interact with the software. Fundamental Computing concepts are not discussed in this approach.

These conceptions do not regard Computing as a basic discipline, which concepts are widely employed as those of Mathematics, for example. Computational Thinking is more than to use the computer to solve problems, or execute a routine mechanical as do computers. Computational thinking is a fundamental human ability, related to how we solve problems. To think computationally is more than programming a computer, because it involves several levels of abstraction [7] and, thus, the approaches that are focused on programming are specialized and limited.

A Computing discipline useful for Engineering is the one that deals with models, methods to solve problems, development of materials, structures, machines, devices, systems and processes that work in a coherent, logical, intelligent, secure and robust way. In this sense, a Computing discipline deals with ways to think like a computer scientist. In this paper, we argue that computational thinking is a fundamental skill for engineers, not just for people interested in the science and practice of Computing. In the next section we show how things of everyday life can be seen from the computational perspective and discuss some key concepts and methods of Computing, illustrating how they are implicitly present in Engineering activities. Next, we discuss some current challenges for designing and implementing a computing discipline for Engineering and present our conclusions.

OBSERVING THE WORLD FROM A COMPUTATIONAL PERSPECTIVE

Computer Science is the study of what can be computed and how it can be computed. Computational thinking includes a variety of scientific and technological artifacts, "mental tools" that reflect the development of Computing, and provide the answer to many questions and the accurate solution to problems.

Algorithms

The notion of Computing requires an algorithm. On a broad sense, any set of instructions that properly specify a behavior can be considered an algorithm. An algorithm is a description of a behavioral process. It consists of a finite set of instructions that govern the behavior step by step. Although the algorithm notion is central to Computing, it is not necessarily related to computers. In order to see the

1 Osvaldo Luiz de Oliveira, Faculty of Campo Limpo Paulista - Faccamp, Rua Guatemala, 167, Jardim América, 13231-230, Campo Limpo Paulista, SP, Brazil, osvaldo@faccamp.br.

world from the Computing perspective, it is necessary to understand as algorithms the behavior of things in the world: Objects, machines, people, plants, static and dynamic structures. To understand a behavior as an algorithm it is necessarily trying to imagine a set of actions that can be executed by the object whose event is being observed.

The production of many artifacts requires the development of algorithms not only for computers but also for the use of people. Computing skills, whether conscious or not, for developing algorithms, are necessary for many Engineering activities. A project is an algorithm that describes how a work or machine must be built. Other examples that expose algorithms include: (1) planning the logical relationship between the activities that make up a work schedule, (2) the development of technical standards to define how a test or experiment should be conducted, or rules and features of a product or service, (3) the development of a strategic plan that outlines goals and objectives, (4) the writing of administrative procedures to set up actions and responsibilities that need to be complied, in order to make the quotation and purchase materials for a work.

Abstraction

The concept of abstraction is fundamental to imagine, create, understand and improve algorithms. Abstraction is the process or the result of reducing the information content of a concept or phenomenon, generally in order to retain only the information that is relevant for a particular purpose. For example, a person can be seen only by their name and official identification number (data abstraction) and also among the intricate set of their actions in the world, like eating, walking (abstraction process). We can have a hierarchy of abstractions that mainly specify the main ideas without the intricacy of the details, which will be described in separate modules. Abstracting is separate different concepts, it is choosing representations for a problem or modeling relevant aspects of a problem to make it tractable. It also provides confidence so that we can modify large and complex systems in a safe way, without understanding all details. In Civil Engineering, designs of Architectural, structural, electrical and plumbing of a building are different abstractions of the same work. Different levels of details of an architectural project create, for example, a hierarchy of abstractions. Computational techniques can be used to retain only relevant information for a certain purpose, modeling techniques and discovery of knowledge can aid the engineer in developing more concise and understandable projects, in addition to help in solving problems.

Problem solving

The theoretical knowledge of Computer Science enables us to evaluate the difficulty of solving a problem and the best way to solve it. We can examine if an approximate solution is good enough and if false positives or negatives are allowed. Currently, we have a theoretical framework that lists many of the strategies for solving problems. A group of widely referenced general strategies includes inductive approaches such as “divide and conquer” and “decrease and conquer”, reductionist approaches, such as “transform and conquer” [3], and other less general approaches as “dynamic programming”, “brute force” and “greedy” [5]. Complex problems of Logistic Engineering as “Vehicle Routing” or “Traveling Salesman”, of Packaging Engineering as “Knapsack”, problems of planning and control processes, present in all areas of Engineering, may have their solutions or approximate solutions found with the aid of these strategies.

Solutions may be limited in complexity or may not exist

Different solutions can be proposed to solve the same problem. The solutions may differ in the amount of steps required to solve the problem (time complexity), and also at the consumption of other resources such as memory (space complexity). Complexity is inherent to algorithms and does not depend on the speed of a processor, whether it is a human being, a fast or slow computer. There are feasible solutions to certain problems, while it would take hundreds of thousand years to solve others, even if they are processed by a very fast processor and good amount of memory. For example, a solution to the classic problem of Transportation Engineering which establishes the shortest path between two cities (A and B) would be to generate all possible sequences of cities starting on A and ending on B, discarding those sequences that do not have roads connecting the cities and, finally, choose the sequence of the shortest path. This solution is not feasible in practice because it takes hundreds of years to be achieved by a modern computer, even for maps containing about 50 cities. However, we know that there is a solution that can be achieved in a few seconds under these conditions. There are also problems that are believed to have no solution.

CHALLENGES IN CREATING A COMPUTATIONAL THINKING DISCIPLINE FOR ENGINEERING

Today, Computing is widely researched and applied, taking into account that the processor is a computer. There is little research on the uses of Computing by people in their various activities. Researches must be developed to understand how people use algorithms and what algorithms are suitable for humans. What language, or dialect, and linguistic structures do people use in the production of artifacts including algorithms? Architectural projects, for example, are developed in a predominantly graphic language. How to write precise and unambiguous architectural projects considering such languages? Does this language require additional linguistic structures for this purpose or a particular project? In order to design and implement a process to engage stakeholders from multiple sectors and identify the computational tools and problem solving skills,

[6] has by objective to define how these skills can be integrated into Engineering curricula.

An important issue raised by these debates is to understand how people specify processes using natural language. For example, specifying a process to "find a list of employees with the job of photographer and exercising leadership role, giving a file containing the records on paper". Surprising results have pointed out that people rarely use flow control structures (if, while, etc.) common to the formal language for describing computer algorithms [2]. How do people without programming knowledge specify an algorithm for a task? In a recent paper, Lewandowski [4] explore the task of selling tickets to a theatre play given the following scenario: Multiple sellers will work at the same time, (2) the customer will ask for "n" seats and the seller should get the best seats available, (3) the seller must deal with different payment options (debit card, credit card and check). Lewandowski noted the correctness of solutions, especially to what, in Computing jargon, we call concurrency control, i.e., preventing the same seat from being sold more than once. The results of Lewandowski suggest that students think "naturally" in concurrency control and are able to develop solutions to the task given.

The teaching of Computing for engineers should require different methods from those used for the Computing area students. The development of methods that work for Engineering students will be required to answer difficult questions such as: What do these students know about Computing? (2) What is difficult and challenging for them? (3) What kind of tools can make the computer thinking more accessible to them? These issues are addressed both by researchers in the area of Education in Engineering and Computer Science.

CONCLUSION

The computational models and methods enable us to solve problems and develop the design of engineering systems, what none of us could do without such knowledge, whether we use it consciously or not. The ability to think computationally is important for engineers and not only for people interested in Computing. The Game Theory [1] has modified and expanded the boundaries of Engineering Economics. Combinatorial Optimization is an alternative to the solution of many complex problems of Engineering, the "Nano Computing" and "Quantum Computing" have changed the thinking and theories of Chemistry and Physics, to name a few areas of knowledge. Computational thinking is a great vision that can help people in everyday life, students, researchers and engineers in their activities. Today, after nearly sixty years of the advent of digital computers, we have a strong body of knowledge on Computing for these computing machines. Often we do not realize that Computing is something that is beyond computers, that the world and its beautiful phenomena can be interpreted on a computational perspective. However, we need research on how this can be done and we need to reorganize the educational processes in order to understand Computing as a basic discipline, focusing on methods, artifacts and ways of thinking present in Computer Science, intended for Engineering students.

REFERENCES

[1] Davis, M., D., *Game Theory: A Nontechnical Introduction*. Mineola, NY, USA, Dover Publication, 1997.

[2] Guzdial, M., Paving the Way for Computational Thinking. *Communications of the ACM*, Vol. 51, Nº 8, 2008, pp. 25-27.

[3] Levitin, A., Papalaskari, M., Using puzzles in teaching algorithms, *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, Cincinnati, Kentucky, USA, 2002, pp. 292-296.

[4] Lewandowski, G., Bouvier, D., McCartney, R., Sanders, K., Simon, B., Commonsense Computing (episode 3): Concurrency and Concert Tickets, *Proceedings of the third international workshop on Computing Education Research*, Atlanta, Georgia, USA, 2007, pp. 133-144.

[5] Sternberg, R. J., Frensch, P. A., eds., *Complex problem solving: Principles and mechanisms*, Hillsdale, NJ, USA, Lawrence Erlbaum Associates, 1991.

[6] Vergara, C. E., Urban-Lurian, M., Dresen, C., Coxen, T., MacFarlane, T., Aligning Computing Education with Engineering Workforce Computational Needs: New Curricular Directions to Improve Computational Thinking in Engineering Graduates, *Proceedings of the 39th ASEE/IEEE Frontiers in Education Conference*, San Antonio, TX, USA, 2009, pp. W2H 1- W2H 6.

[7] Wing, J. M., Computational Thinking. Communications of the ACM, *Communications of the ACM*, Vol. 49, Nº 3, 2006, pp. 33-35.