THINKING ABOUT COMPUTATIONAL THINKING

J.M.Mastonov
Karshi city 10th school's Information technology teacher
mastonovjahongir@gmail.com

ABSTRACT: Jeannette Wing's call for teaching Computational Thinking (CT) as a formative skill on par with reading, writing, and arithmetic places computer science in the category of basic knowledge. Just as proficiency in basic language arts helps us to effectively communicate and in basic math helps us to successfully quantitate, proficiency in computational thinking helps us to systematically and efficiently process information and tasks. But while teaching everyone to think computationally is a noble goal, there are pedagogical challenges. Perhaps the most confounding issue is the role of programming, and whether we can separate it from teaching basic computer science.

I think that to successfully broaden participation in computer science(CS), efforts must be made to lay the foundations of CT long before students experience their first programming language. I posit that programming is to Computer Science what proof construction is to mathematics, and what literary analysis is to English. Hence by analogy, programming should be the entrance into higher CS, and not the student's first encounter in CS. We argue that in the absence of programming, teaching CT should focus on establishing vocabularies and symbols that can be used to annotate and describe computation and abstraction, suggest information and execution, and provide notation around which mental models of processes can be built. Lastly, we conjecture that students with sustained exposure to CT in their formative education will be better prepared for programming and the CS curriculum, and, furthermore, that they might choose to major in CS not only for career opportunities, but also for its intellectual content.

Keywords: IT, computer, computer science, computational thinking, computer-like thinking, computational-informatic thinking

INTRODUCTION

Since the dot-com bubble, the conundrum we face in computer science is how such a useful discipline can have such difficulties attracting students, despite continuing growth of the IT industry. We blame student disinterest on career instability, but similar and even stronger arguments have long existed for other disciplines with little impact on enrollment. Recent data from the National Center for Education Statistics show that computer and information sciences conferred fewer degrees than either the visual and performing arts or the social sciences and history – hardly the stuff that ironclad career guarantees are made of. Not surprisingly, the number of students majoring in CS lags far behind those majoring in other practically-perceived disciplines such as education or business. Through the years, despite our best efforts to articulate that CS is more than "just programming," the misconception that the two are equivalent remains.

PROGRAMMING: DESCRIBING COMPUTATIONAL PROCESSES

For those students interested in pursuing higher-level English and mathematics, there exist milestone courses to help shift the focus from the development of useful skills to the academic study of these subjects. In English, courses in literary analysis pave the way for students to read texts critically and to argue theoretically. In mathematics, a course on proof

understanding and construction is the gateway into higher mathematics. These courses make critical intellectual leaps. And while being educated implies proficiency in basic reading, writing, and quantitative skills, it does not imply knowledge of or the ability to understand and carry out scholarly English and mathematics. Analogously, we believe the same dichotomy exists between computational thinking, as a skill, and computer science as an academic subject. Our thesis is this.

> "Programming is to CS what proof construction is to mathematics and what literary analysis is to English."

Example (Introduction to Multiplication).
Current curricula introduce multiplication in Grade 3. Two common concepts are "multiplication is repeated addition" and"the result of multiplication is the same no matter which number you write first."

The use of repeated addition as a definition for multiplication is an opportunity to introduce two computational concepts: iteration and efficiency. We may explain that each application of the symbol + is an iteration, and that while the operation is commutative, the efficiency of the two forms of expression may be different. Some useful exercises may be the following.

1. For each multiplication, write it as repeated additionand then the answer. Also write down the number of iterations that are required.
2. Write the multiplication by switching the two numbers, and compare the number of iterations required. Which one is more efficient?

Example

expression	numbers switched	which is more efficient?
3 × 6	6 × 3	6×3 needs 3 iterations, 3 × 6 needs 6, so 6 × 3 is more efficient.

DISCUSSION

Through practice and repeated encounters, thinking and communicating in the CTL will become second nature by the time students reach their final year in high school. A culminating AP course or equivalent introductory college courses, such as the excellent "Great Theoretical Ideas In Computer Science" by Steven Rudich and colleagues at CMU, will help to integrate students' experiences and prepare them for exposure to programming. Such courses may formally address computational properties, such as convergence, efficiency and limits of computation, again without necessarily referring to a specific computing agent. College-level service courses may be offered on domain-specific computational thinking (e.g., bioinformatics, chemical informatics).

To truly integrate computational thinking into current primary and secondary curricula undoubtably presents significant challenges. It will necessarily be a gradual and evolutionary process, and requires concerted efforts and coordination among many constituents of the wider education community. We see concrete efforts towards achieving broader CT literacy as some of the most exciting, challenging, and necessary next steps in the maturation of our discipline.

REFERENCES:

1. J. M. Wing. Computational thinking. CACM 49(3):33-35, 2006.

2. A. Cohen and B. Haberman. Computer science: a language of technology. SIGCSE inroads 39(4):65-69, 2007.
3. P. J. Denning and A. McGettrick. Recentering computer science. CACM 48(11):15-19, 2005.
4. www.digitalpromise.org
5. www.unite.ai