COMPUTING IN SCHOOLS

Michal Armoni

Spiral Thinking: K–12 Computer Science Education as Part of Holistic Computing Education

INCREASINGLY, MORE COUNTRIES have come to recognize the importance of pre-college computing education, and K–12 computing curricula are currently being developed throughout the world. It is really encouraging, even heart-warming, to read reports from the best computer science (CS) educators from various countries, who are involved in K–12 computing curricular development. International conferences (e.g., the Workshop in Primary and Secondary Computing Education, WiPSCE, and Informatics in Schools: Situation, Evolution and Perspective, ISSEP) focus on K–12 computing education, and offer opportunities for CS educators to engage in lively discussions on various issues concerning K–12 computing education.

As part of these ongoing discussions taking place through various media, I heard several arguments about the knowledge components of K–12 CS curricula. For example, some argue that K–12 CS curricula should not contain specific components of undergraduate CS curricula, since otherwise, the students who have studied CS in high school will be bored when they take undergraduate introductory courses. Or, in an example having a more concrete nature, others argue that a K–12 CS curriculum should only deal with very basic data structures, such as built-in types and one-dimensional arrays. Two-dimensional arrays, and certainly lists or stacks are beyond the understanding of high-school students. And, another argument offered is that there is no need to teach more than one programming paradigm in high school, a second paradigm is too much.

These three statements, given here as examples, are of different natures and stem from different justifications. The second and the third statements indicate a belief in the limited capabilities of high-school students—students can only handle so much, and we should not overload them with too many concepts or concepts that are too difficult. The first statement calls for variability—avoiding repetition or overlap when moving from a K–12 CS curriculum to an undergraduate CS curriculum. The first argument is concerned about the potential negative effects of K–12 CS education on undergraduate CS education. The other two arguments are concerned about the potential negative effects of CS undergraduate education on K–12 CS education, specifically, the effects of transferring knowledge units from the undergraduate curriculum.

However, these three (and similar) arguments can be addressed by adopting the ideas of the world-renowned psychologist Jerome Bruner, whose educational theories had, and still have, enormous effect on educational practice, specifically on curricular development. Bruner thought that nothing is too complex for a child, as long it is taught at the appropriate level, "We begin with the hypothesis that any subject can be taught effectively in some intellectually honest form to any child at any stage of development" [3: 33]. Some of you may remember one of my previous columns (December 2012) that cited Piaget, another important educational psychologist. In that column, I asked whether starting the CS pipeline as early as kindergarten is feasible and effective, considering Piaget's theory of cognitive development, according to which young children cannot handle abstractions and abstract thinking. Indeed, Bruner disagreed with Piaget regarding this point (and I will return later to this contradiction of views). He believed that not only can young children understand complex concepts, at some intellectual level, but that early treatment of complex concepts can even foster deeper understanding of these concepts at a later stage. This is the basis for Bruner's spiral curriculum, in which concepts are introduced early, and are revisited again and again throughout the curriculum, each time treated in a deeper manner, at an intellectual level appropriate for the learner.

The learning of a certain concept does not occur at a specific point; rather, it is a process in which understanding is built and developed based on previous knowledge. The more solid the basis is, the deeper and more meaningful is the learning constructed on top of it. Take, for example, the mathematical concept of a function. A learner encounters this concept many times throughout the mathematical curriculum, each time at the appropriate level. The learner's acquaintance with the concept can start as early as kindergarten, for example, with a list of products and corresponding prices in a kindergarten play-grocery, and it gradually evolves over the years until a function can be perceived as a theoretical mathematical object. Previous exposure does not come at the expense of later exposure. On the contrary, it supports it.

Spiral teaching is relevant for all subjects, and computer science is no exception. Once there is a K–12 CS curriculum, it is part of the CS teaching and learning process; a part that precedes undergraduate CS learning, supports and enriches it. These phases should not be considered as separate phases that might interfere with one another (as suggested by the arguments mentioned above), but rather as complementing phases that together result in a holistic learning process.

tion of data abstraction, the use of these data structures, as abstract types, can be emphasized, rather than their implementation. Implementation issues probably should not be ignored altogether, since they assist in discussing important aspects of abstraction, such as distinguishing the interior of a black box from its exterior, and being able to move between these two levels of abstraction as necessary. But for this purpose, any simple implementation can suffice (e.g., implementing a list using an array and implementing a stack using a list etc.), and there is no need to deal with all the specific details of this implementation. Later, at more advanced phases of the curriculum, this issue will be revisited with more sophisticated and efficient implementations.

available everywhere, and where it is available, it is not always mandatory. As long as this is the case, there will always be undergraduate CS students who will join the spiral at a relatively late stage, without experiencing early parts of it. However, this does not mean that we should not let those students that do start early benefit from a spiral curriculum and exploit its advantages. A properly designed K–12 curriculum, built around a spiral treatment of fundamental ideas, will result in a better learning process that supports undergraduate studies, also in terms of affective aspects.

> The objective is for students to understand that there is more than one way to approach and think about a problem …

According to Bruner, spiral teaching is especially effective and beneficial for learning the fundamental ideas, the big concepts. Fundamental ideas satisfy four criteria [5], the second of which is the vertical criterion, which states that a fundamental idea can be taught at almost every level of understanding. That is, fundamental ideas allow for spiral treatment, but meaningful learning of them also necessitates spiral treatment. Fundamental ideas are soft concepts [4], and as such are indeed complex and challenging to teach and learn. Their generality—being big—is what makes them so challenging to teach, since it is never enough to teach them in some specific contexts. Meaningful, non-specific transfer is necessary for attaining a deep and real understanding of such ideas. Spiral teaching promotes meaningful learning and specifically promotes non-specific transfer. In other words, we should not be afraid of including complex concepts in a K–12 CS curriculum. Rather, we should aspire to do exactly that, and look for ways to introduce these ideas at appropriate intellectual levels and in the appropriate manner.

For example, and referring to the second statement above, it is preferable to include the idea of abstraction in general, and data abstraction in particular, in a CS program for secondary schools. Non-basic data structures can serve as an excellent tool for this purpose. There is no need to avoid abstract data types such as lists, stacks, or trees. If the objective is a spiral introduc-

Similarly, spiral treatment of the soft concept of a paradigm calls for introducing a second programming paradigm in secondary education. The objective is for students to understand that there is more than one way to approach and think about a problem, and this objective does not necessitate taking a complete, professional course in another programming language of another paradigm, as one would perhaps expect in the context of an undergraduate curriculum. It is enough to exemplify this idea with a gentle, not-too-deep introduction of a relatively simple language. More important is revisiting the idea, in every place that lends itself to such revisits. When approaching a problem, the class can discuss the possible approaches, or the different lenses through which the problem can be viewed, before choosing one to follow.

My first two columns (2010 March [1] and 2010 December [2]), which described how theoretical topics of CS can be taught in high school, can serve as another example of a spiral treatment, teaching complex ideas at a simplified level, at a level appropriate for 11th and 12th graders.

Implementing spiral teaching for CS is not straightforward, and is challenging, since even today K–12 CS education is not

I can testify to that from my own past experience as a freshman, though in the context of calculus and not CS. I had been fortunate enough to study in a high school that participated in an experimental calculus program. This was discovery-based learning that handled complex ideas of calculus in a simplified manner that maintained their conceptual essence. When I started my first undergraduate calculus course and was introduced to the idea of limits, with its very symbolic "epsilon-delta" definition, it was for me like meeting an old friend, whereas other students were intimidated by this introduction, which was so much different from the formula-based calculus that they had encountered in high school. Not only did this make the in-depth learning easier for me, it also affected my confidence and motivation. It is not that the high-school course on calculus that I took was more difficult than the regular one, but it was different. And these differences were exactly those that enabled a continuous and evolving learning process, rather than relearning the same subject again from a different perspective when it is even better to forget what you have learned before, so that it will not interfere with the current learning process.

We are still left with the contradiction between Bruner and Piaget concerning intellectual thresholds. In the terms of my December 2012 column, this contradic-

tion concerns the appropriate age at which the CS spiral can begin. At the extreme end, can kindergarten students handle CS concepts? Specifically, since abstraction is inherent in any CS concept, can the idea of abstraction be taught in a way appropriate for young children, while still maintaining its essence? A positive answer, following Bruner, induces an early-based spiral treatment of CS ideas, specifically of abstraction. A negative answer, following Piaget, indicates that a too-early introduction to CS ideas might not support future learning of these ideas, and might even interfere with it. For me, at least, this is still an open question. In order to address it, CS educators should devise an appropriate framework for introducing abstraction to very young children, and then empirically assess the effectiveness of this framework. However, leaving this extreme end open for now, and based on research and experience, I have no doubt that CS fundamental ideas can be learned in high school and even in junior high school, as part of a spiral curriculum.

These views are consistent with what I have mentioned often in these papers: my view of K–12 CS education as a platform for teaching CS ideas, a process that should be designed around fundamental CS ideas. For me, this is an educational objective that should precede the knowledge-driven objectives of conveying this knowledge unit or other. **Ir**

References

[1] Armoni, M. "Computing in Schools – On Teaching Topics in Computer Science Theory." *ACM Inroads*, 1 (1): 21-22.

[2] Armoni, M. "Computing in Schools – On Teaching Topics in Computer Science Theory – Part II: Making it Possible by Using the Prism of Fundamental Ideas." *ACM Inroads*, 1 (4): 18-19.

[3] Bruner, J. S. *The Process of Education*. (Cambridge, MA: Harvard University Press, 1960).

[4] Corder, C. *Teaching Hard Teaching Soft: A Structured Approach to Planning and Running Effective Training Courses*. (Aldershot, UK: Gower 1990).

[5] Schwill, A. "Fundamental ideas of computer science." *Bulletin of European Association for Theoretical Computer Science, 53* (1994): 274-295.

Michal Armoni
Department of Science Teaching
Weizmann Institute of Science,
Rehovot, 76100, Israel
michal.armoni@weizmann.ac.il

DOI: 10.1145/2614512.2614521

DISTANCE LEARNING

Yoav Yair

Did You Let a Robot Check My Homework?

IN EDUCATION AND LEARNING, one of the central pillars of teacher-student interaction is the feedback on assignments and tasks. While grades offer the obvious and tangible type of feedback, they are merely a "right-wrong" indication and have a rather limited benefit to students. A more profound and valuable type of feedback would be to pre-define clear rubrics, and after submission of the assignment to show where the errors are, analyze them and their sources, and point to the correct answer by explaining why it should be right. In a canonical paper, Sadler [9] identified three key conditions that enable students to benefit from feedback on their academic tasks: students need to know what "good performance" is, know what their own current performance is and how it relates to the good one, and identify means to close the gap between the two. The teacher, on the other hand, should pursue a "formative assessment" aiming to help students accelerate and improve their learning. Timing of feedback is clearly a crucial factor, because any delay can jeopardize the progress of a student's learning along the course.

Seven principles of good feedback practice can be identified [7], among which are

- clear definitions of what good performance is,
- a delivery of high-quality information to students about their learning - thus facilitating the development of reflection in learning and
- a way to enable students to close the knowledge and performance gaps.

Faculty members dedicate considerable time and effort on feedback, which research shows is of special importance to novices in introductory level courses. Students' capacity to effectively utilize the feedback they are receiving depends on various factors and is content dependent. Price et al. [7] noted that "*[i]n higher education, the likelihood of feedback providing unambiguous, categorical feedback to*

Faculty members dedicate considerable time and effort on feedback, which research shows is of special importance to novices in introductory level courses.