Examples of Computational Thinking in the K-12 Experience

Irene Lee, Santa Fe Institute and Joyce Malyn-Smith, ITEST Learning Resource Center at EDC

ITEST CT White Paper Working Group (in alphabetical order: Walt Allan, Foundation for Blood Research; Bob Coulter, Missouri Botanical Garden; Jill Denner, ETR Associates; Jeri Erickson, Foundation for Blood Research; Irene Lee, Santa Fe Institute; Joyce Malyn-Smith, ITEST LRC at EDC; Fred Martin, University of Massachusetts Lowell)

June 4, 2010

Several years ago a working group of ITEST Principal Investigators and evaluators posed the question: What do ITEST youth know and what do they do with technology? A short survey of ITEST projects indicated that ITEST youth were using a wide variety of technology tools and systems, from simple to highly sophisticated; and many to high degrees of skill. As we dug deeper we began to explore the impact this use of technology had on youths’ patterns of thinking, processing information and problem solving. We began to discuss some of the commonalities we were observing among the ways youth approached problems and used computational tools/systems to develop various solutions. We talked about this as a type of technologically enabled and enhanced thinking. About the same time Jeannette Wing’s article on Computational Thinking was published in Communications of the ACM. (Wing, 2008) We found the concept of Computational Thinking (CT) closely aligned to what we were observing in the behavior of participants in our projects, and began to discuss our observations in light of the CT framework. A new working group, focusing on Computational Thinking emerged within the ITEST community of practice. Over the past year this working group has explored Computational Thinking within ITEST and other NSF EHR programs and identified several examples of what Computational Thinking looks like in action. This paper shares those examples and some of our thinking on this topic.

Computational thinking (CT) promoted by Jeannette Wing (2006) describes a set of thinking patterns that emerge from computer science but that are useful in much broader contexts, as they involve systematically and efficiently processing information and tasks, with or without a computer. CT involves defining, understanding, and solving problems, reasoning at multiple levels of abstraction, understanding and applying automation, and analyzing the computational tools we develop to solve problems.

There are three main pillars of CT: abstraction, automation, and analysis.

Abstraction may take the form of stripping down a problem to what is believed to be its bare essentials. Abstraction is also commonly defined as the capturing of common characteristics or actions into one set that can be used to represent all other instances.
Automation is using the computer as a labor saving device in which processes are used to execute a set of repetitive tasks quickly and efficiently compared to the processing power.

Analysis, as described by Cuny, Snyder, and Wing (2010), is a reflective practice. It refers to the validation of whether the abstractions made were correct. One might ask “Were the right assumptions made when narrowing the problem to its bare essentials?” or “Were important factors left out?”

It is also important to note at the outset that CT shares elements with various other types of thinking such as algorithmic thinking, engineering thinking, and mathematical thinking. As such, CT draws on a rich legacy of related frameworks as it extends previous thinking skills to include concepts unique and specific to computational media.

In layman’s terms, computational thinking is an evolving construct that is intended to capture and define foundational ways of thinking that are increasingly relevant in the digital age, where ubiquitous use of technology continues to change the ways we live, learn and work. While some think computational thinking is only developed after years of progressively intense studies of computer science, others believe that today’s youth – many of whom are power users of technology – are developing computational thinking through their daily intensive use of technology over a long term. In either case, many consider computational thinking to be a set of basic skills, a type of analytic, procedural and algorithmic thinking, that will enable our students to harness the power of our cyber-infrastructure to become the idea makers and innovators of the future, enabling us as a nation to understand and address the daunting issues we face in the 21st century and compete and succeed in a global economy driven by technology. If computational thinking is, indeed, a key to developing the capacity to discover, create and innovate, then teachers and other youth leaders need to understand computational thinking, how it connects to their curriculum, and how to recognize, nurture and assess these talents in today’s youth. To that end, this paper seeks to address the following questions:

• What does computational thinking for youth look like in practice? And,

• How can we support growth in computational thinking, both in and out of school?

What does computational thinking for youth look like in practice?

Much of the existing literature on computational thinking focuses on formal computational thinking such as one might encounter in a college-level computer science course. In this paper, we take a different approach, considering how computational thinking appears to be evolving among pre-college youth in and out of school. A wide range of activities build CT skills. Distilling the rich and complex legacy of formal computational thinking, we base our understanding of computational thinking for youth as an approach to framing problems or issues
that relies on three main pillars: abstraction, automation, and analysis (Cuny, Snyder and Wing, 2010). Phrased more tangibly, Dave Moursund (2009) suggests that “the underlying idea in computational thinking is developing models and simulations of problems that one is trying to study and solve.” In addition to the model-based approach promoted by Moursund, we will consider computational thinking in two other domains: with robots, and with game design. Although we recognize that a wide range of activities build CT skills, we found several examples from these three domains in our projects involving innovative work in computing being done with middle- and high-school students.

In a Project GUTS (Growing up Thinking Scientifically) middle school students actively engage in computational thinking through the modeling and simulation of real-world issues within their communities. Within Project GUTS clubs, students investigate local issues in their community, create agent-based models in StarLogo TNG with which they investigate the issues further and test potential mitigation strategies virtually. For example, in an investigation of epidemics, Project GUTS club members collected data on student circulation within their schools, the physical layout of their school, and researched various contagious diseases. Using this data, they customized a computer model of a simple contagion to reflect local conditions and match a chosen virulent. Their computer models were used as experimental test beds with which they tested strategies to mitigate potential epidemics within their school community. Interestingly, club members showed great creativity – some chose gossip, bullying behavior, and the spread of fads and fashions as contagious elements.

A second key application area of computational thinking with pre-college student is designing and programming robots and other physical devices with embedded code. In iCODE (Internet Community of Design Engineers) middle and high school children complete a variety of microcontroller-based projects, beginning with a simple project with programmable flashing lamps, to a musical memory game, to fully autonomous (self-controlled) robots that enter a contest. In many respects, the type of CT that students engage in when developing these projects is similar to the thinking involved in creating agents in game programming, but with iCODE and other similar work, students will focus on one agent -- their project -- and the immediate world that surrounds it and provides input to its sensors. With game programming, students are more likely to think about interactions among a collection of game agents.

A third key application area of computational thinking is computer game design. In the iGame after school program, middle school children engage in computational thinking by programming computer games using Storytelling Alice. Building a computer game requires not only programming, but the ability to think at multiple levels of abstraction and in terms of scale. Salen (2007) says that “knowing how to put together a successful
game involves system-based thinking, iterative critical problem solving, art and aesthetics, writing and storytelling, interactive design, game logic and rules, and programming skills” (p. 305). When students program their games using Storytelling Alice, they engage in relevant CT concepts such as algorithmic thinking, as they solve problems related to game programming using conditionals, iteration, and sequential execution. Game programming also engages students in abstraction, because students must create a model of their world, and set up variables to define the state of the world. Finally, game programming engages students in an understanding of scale, when they create a list data structure so they don't have to program each object individually.

For example, Squire (2004) has shown how the game Civilization has used a mass-market simulation game to promote historical understanding. Students then use the game’s modification tools to create their own game scenarios. Likewise, the Community Science Investigators program engages youth in “augmented reality” games that provide an overlay of an environmental mystery game scenario within their neighborhood. As the players seek clues to solve the mystery, they are engaging in simulated science within a game context. Later in the program participating students build on their experience with simulations to design their own games.

**Use-Modify-Create Learning Progression**

Based on our observations in several youth projects across the US, we propose a three-part model or framework that illustrates a learning progression of how CT skills develop.

**USE:** The outcome of this initial phase is that youth learn how to use the technology, including the interface and tools, and the kinds of products that others have made. This is a first step that must happen before higher levels of engagement with CT.

**MODIFY:** As comfort is gained in using the tools, youth begin to experiment and explore, modifying existing programs or projects. The outcome of this phase is that students to begin to understand how they can control underlying mechanisms to bring about different results, a skill the they will later use in making original creations.

**CREATE:** In this phase, youth apply their growing computational thinking skills to create an original product. Implicit in the development, of course, is that the creation will be used and modified over time.

In the Use phase of iGame program, middle school students learn how to use the programming environment, in this case Storytelling Alice. To do this, they take the interactive tutorials and play games made by their peers. The outcome of this phase is that youth learn about the software interface, and the kinds of games they might make. In the Modify phase, they complete self-directed "challenges," which are step-by-step instructions for modifying and expanding on existing programs. The outcome of this phase is that students begin to understand the
mechanisms they will use to program their game. They learn to use tools, such as the clipboard to copy and paste code. The challenges get increasingly more difficult, with more complex and abstract concepts, and with fewer and fewer instructions. In the Create phase, they program original games, with varying degrees of complexity. There is a continuum of CT within this phase, with some students engaging in high-level abstraction (creating complex new methods or embedded loops) and others creating more linear code. For instance, many students apply the concept of conditionals using simple If/Else commands. Others use nexted If/Else commands, suggesting a high level of mastery of these concepts. As stated in the NAS report (2010), programming is learning a language that one can use to express new ways of thinking and to learn to express ideas in a precise way. Learning that language in iGame involves not only creating, but also analyzing, testing, and revising their games, as well as testing games made by their peers.

EcoScienceWorks is an in-school curriculum that features SimBiotic Software’s EcoBeaker™ agent-based ecology simulations re-designed for Maine’s one-to-one middle school laptop program. These simulations (Maine Explorer) and the accompanying teacher-designed field exercises and lesson plans were designed to replace ecology curriculum currently being taught in Maine thus were aligned with state and national learning standards and the topics (succession, species interactions, habitat fragmentation, eutrophication and invasive species) were influenced by the teacher teams that were part of the EcoScienceWorks ITEST project. Students learned the interface and how to perform directed experiments in the USE phase of the project. This involved using the simulation’s tools to discover important features of the habitat. For instance, the microscope tool is used by students to hover over an individual agent’s icon to discover its gut contents as they work out the habitat’s food web. The Use phase of the project also involved performing experiments using the simulation. In the eutrophication lab students discovered the impact of different levels of phosphorus pouring into a simplified lake ecosystem on population sizes for algae, zooplankton and trout and uncovered an explanation for the decline in trout population sizes by measuring the simulated lake’s dissolved oxygen content. Thus, this phase of the project was rich in the CT aspects of abstraction and concepts such as control of variables, replication of experiments and data analysis. In order to increase student interest and understanding of the underlying design of computer models, a separate programming challenge lab was included with Maine Explorer, called Program a Bunny. A series of challenges in StarLogo TNG-like CodeBlock programming are presented to students. In this Modify phase of the project students learn how to use conditional commands, randomization and recursion to program a single bunny to forage for carrots in a field. The challenges culminate in a competition between the student programmed bunny and a pre-programmed bunny.

Learning CT in School and Out

As we explored NSF funded programs we found most of our examples of CT in the K-12 experience in out-of-school (OST) environments.
As noted above, the Use, Modify, Create progression is developmental. Computational thinking projects like those mentioned above support an iterative cycle that enables increasing sense of agency, where learners are empowered to imagine, create, play, share, and reflect on what they are learning (Resnick, 2007). As this iterative cycle progresses, it is important to maintain a level of challenge that supports growth. As Repenning (2008) notes, students can maintain their sense of cognitive flow (Csikszentmihalyi, 1990) as they progress iteratively through a series of projects. In this work, a student tackles progressively higher challenges as her skills and capacities increase. What was once “too hard” and anxiety-inducing becomes possible with appropriate, incrementally challenging experiences.

Conversely, Repenning argues, boredom will set in if challenges don’t keep pace with growing skills. In fact, most students relish this challenge in their out-of-school lives, seeking out challenges that help them to grow and to demonstrate increased mastery. As Seymour Papert (1998) noted, most young people willingly pursue “hard fun.” This process of increasing challenge and complexity—engagement with a long-term project—is not easily compatible with a curriculum packed with many topics. Curricular flexibility that allows for deep exploration is part of the culture change needed for computational thinking to take root in schools.

With few curricular constraints, the capacity to hire staff with the requisite technology skills, and the necessary technological infrastructure, it’s not surprising that many of the best examples of CT-rich learning happen outside of a traditional school day. Projects designed to support computational thinking can marshal the resources needed to overcome the limitations often found during the school day.

This is not to say that OST programs are the ideal environment. First, access to high quality out-of-school time learning spaces is far from evenly distributed. In particular, rural areas rarely have these spaces, which essentially keeps the school as the sole provider of educational opportunities. Until broadband penetration becomes more common in rural areas, virtual learning opportunities won’t provide meaningful programs either, further exacerbating the opportunity gap faced by rural communities.

Also, many of the most ambitious programs tend to be funded through expensive, time-limited grants from government and foundation sources that serve only a very small portion of the potential base of participants. Continuation past the grant cycle is often dependent on next grants, as is replication in other locations, or it requires a significant outlay of human and financial resources that favors communities with the economic wherewithal to take on such a responsibility. Relying on grants is also problematic in that the low funding percentage for most grant competitions makes it uncertain that a next grant is in the offing. Also, grant support to continue a successful program is usually much harder to procure than is funding to start something perceived as new or innovative.
Given the equity, access, and continuity limitations associated with specialized out of school environments, we need to continue looking at ways to make CT environments more universally accessible through the school environment.

As a community of practice, we recommend moving forward by leveraging what is possible in both formal and informal learning environments to advance our collective work.

To that end, in the next section we share lessons learned along the way and offer the potential next steps for practice and research.

Conclusions

In this paper, we have contributed to the dialogue about computational thinking for youth by using examples from several youth projects to describe what CT looks like, and to consider strategies for engaging youth in CT. Given the importance of CT, we need to deepen our collective understanding to guide our steps forward. We are not yet at the point where we have a set of best practices to recommend, but we do hope this paper will move us closer to that point by start a national dialogue about effective strategies for engaging youth in computational thinking.

At this point we are confident that existing, broad definitions have utility for understanding CT, but there are developmental considerations that need to be addressed. We know from the examples cited here and from other projects that youth can engage in abstraction and automation, but these processes need to be viewed in light of each child's age and prior experiences. More generally, attempts to list fundamental CT skills, such as those articulated by the National Academies of Science (2010), need to be interpreted accordingly. Computer and learning scientists need to collaborate with practicing educators in thinking through and articulating sets of foundational skills and developmental progressions. These can then be considered in light of how they might be used to guide computational thinking in different domains and learning environments. The work here focuses on models and simulations, robotics, and game design; these and other application areas will benefit from such a framework.

As a foundation moving forward, the Use-Modify-Create framework offers a helpful model for understanding how CT develops over time; and provides a useful trajectory as youth engage in progressively more complex tasks and increase ownership of their learning.

Discussion Questions:

• What is computational thinking?

• What does computational thinking look like in practice among youth participating in NSF projects?
• What role does informal learning play in the development of foundational competence in computational thinking?

• How can formal educators assess CT competence in their students and determine where they are in the CT learning progression of Use – Modify – Create?

• How can we support growth in computational thinking, both in and out of school?

References


