Computational Thinking for Youth in Practice

Irene Lee ▪ Fred Martin ▪ Jill Denner ▪ Bob Coulter ▪ Walter Allan
Jeri Erickson ▪ Joyce Malyn-Smith ▪ Linda Werner

Computational thinking (CT) has been described as the use of abstraction, automation, and analysis in problem-solving [3]. We examine how these ways of thinking take shape for middle and high school youth in a set of NSF-supported programs. We discuss opportunities and challenges in both in-school and after-school contexts. Based on these observations, we present a "use-modify-create" framework, representing three phases of students' cognitive and practical activity in computational thinking. We recommend continued investment in the development of CT-rich learning environments, in educators who can facilitate their use, and in research on the broader value of computational thinking.

1 INTRODUCTION

Computational thinking (CT) is a term coined by Jeannette Wing [11] to describe a set of thinking skills, habits and approaches that are integral to solving complex problems using a computer and widely applicable in the information society. Thinking computationally draws on the concepts that are fundamental to computer science, and involves systematically and efficiently processing information and tasks. CT involves defining, understanding, and solving problems, reasoning at multiple levels of abstraction, understanding and applying automation, and analyzing the appropriateness of the abstractions made. CT shares elements with various other types of thinking such as algorithmic thinking, engineering thinking, design thinking, and mathematical thinking. As such, CT draws on a rich legacy of related frameworks as it extends previous thinking skills.

This paper aims to help computing and STEM (science, technology, engineering and mathematics) educators understand computational thinking (what it looks like "in practice", how it connects with their existing curriculum, and how to nurture computational thinking in today's youth) by sharing rich examples from National Science Foundation funded Innovative Technology Experiences for Students and Teachers (ITEST), Academies for Young Scientists (AYS) and Research and Evaluation on Education in Science and Engineering (REESE) programs. The examples provide a lens through which one can consider the implications for learning and teaching computational thinking in grades K through 12.

Key questions include:

- *What does computational thinking for youth look like in practice?*
- *How can we support growth in computational thinking, both in and out of school?*

The examples and recommendations presented within this paper were collected by the ITEST working group on Computational Thinking. All of the authors are members of this community by virtue of their involvement with current or previous ITEST programs. This work is intended to complement The National Academies "Computational Thinking for Everyone" workshop series and the work currently being carried out by the Compuer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE) as part of the Computational Thinking Thought Leaders project, and to further the discussion by presenting examples of computational thinking in action within programs for youth in both formal and informal settings.

2 COMPUTATION THINKING FOR YOUTH IN PRACTICE

In this paper, we respond to several recent calls to describe CT among youth and to identify strategies for integrating CT into K-12 settings [4][5][7]. We apply and build on existing descriptions of CT, which have been based on thinking like a computer scientist in college and beyond. Specifically, we offer examples of what computational thinking looks like among youth from a range of cultural and socioeconomic backgrounds, both in and out of school. Examples are drawn from three domains: modeling and simulation, robotics, and game design and development. Across these domains, we have identified commonalities in the nature of youth's computational thinking.

We found the terms of abstraction, automation, and analysis [3] to be useful for understanding how youth can use CT to approach novel problems. Abstraction is "the process of generalizing from specific instances." In problem solving, 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 a labor saving process in which a computer is instructed to execute a set of repetitive tasks quickly and efficiently compared to the processing power of a human. In this light, computer programs are "automations of abstractions." Analysis is a reflective practice that 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?", "Were important factors left out?" or "Was the implementation of the abstraction or automation faulty?" Table 1 provides a summary of these domains

TABLE1: EXAMPLES OF CT IN THREE DOMAINS

	Abstraction	Automation	Analysis
Modeling & Simulation	Selecting features of real-world to incorporate in a model	Time stepping using a model as an experimental testbed	Were the correct abstractions made? Does the model reflect reality?
Robotics	Design robot to react to a set of conditions	Program checks sensors to monitor conditions	Are there situations that were not taken into account?
Game Design & Development	Games are abstracted into a set of scenes containing characters	Game responds to user actions	Do the elements incorporated make the game fun to play?

In the next sections, we use examples from three out-of-school time (OST) youth programs to illustrate what the three aspects of CT look like in practice, in each of the three domains. Each of these programs offers opportunities for middle and high school students to engage in computational thinking. The students come with a range of computer experience and confidence, including students with limited English and no computer at home, as well as students who have grown up tinkering with technology. The hands-on and student-driven nature of the programs is designed to allow students at all levels to engage in CT.

2.1 Modeling and Simulation

The first domain we consider is modeling and simulation. Dave Moursund [6] suggests "the underlying idea in computational thinking is developing models and simulations of problems that one is trying to study and solve." In Project GUTS (Growing up Thinking Scientifically) middle school students actively engage in computational thinking as they design and implement models of local relevance and then use the models to run simulations. Students used the process of abstraction to narrow the problem down to something that could be implemented on the computer using StarLogo TNG, an agent based modeling tool. Restrictions imposed by the modeling environment include an upper bound on the number of agents (4076) and a limit on the size of the environment (101 by 101 cells). Within these parameters students designed and created models as testbeds to answer questions about real-world concerns. For example, as part of the Project GUTS unit on Epidemiology, a group of students wanted to know if a disease would spread throughout their school population given the layout of the school, the number of students, the movement of the students, the virulence of the disease, and the number of students initially infected. See Figure 1.

Figure 1: GUTS club members creating ecosystem models in Chicago.

Mapping this question and scenario onto an agent based model, agents were used as abstractions or simplified representations of students and the number of agents matched the number of students in their school. Agents were given movement behaviors that were abstractions of moving from classroom to classroom, and decisions were made about which features of the school were important to take into consideration before a 3-D virtual model of the school building was created. For instance, students decided that recreating the number and location of passages and doors at the school was important. Additionally students modeled the characteristics of the contagion being spread: how often contact between students spread the disease from one to the other and how many students were initially infected. To make the model a testbed capable of running experiments, it was equipped with interface sliders to control individual variables. One slider controlled the number of initially infected agents and another controlled the virulence of the contagious element. See Figure 2.

Automation was used in a number of ways. The "program" itself automated "stepping through" or advancing the simulation through the use of a run loop that updated each agent's state, location, and color (representing sick or healthy) at each time step. Because agent-based models involve randomness, for example, the initial location of infected individuals is chosen randomly, they tell us the probabilities of certain outcomes rather than predictions. Automation was used to execute multiple "runs" of the experiment with

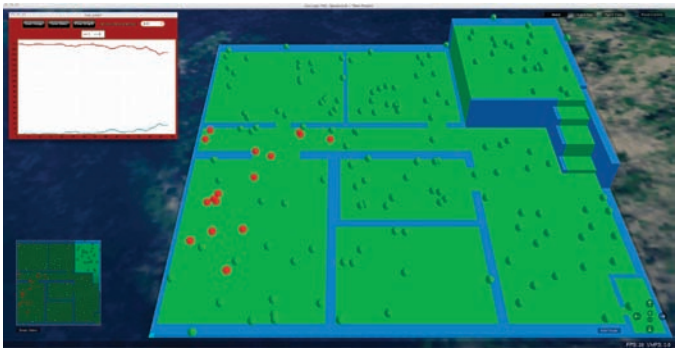

Figure 2: Students' customization of contagion model to reflect school layout.

the same parameter settings in order to attain the probabilities of certain outcomes. Once the simulations were run and data on the number of infected individuals after a fixed number of time steps were collected, students reflected on the outcomes. In some cases, the students were able to analyze their models, the assumptions and abstractions made, by comparing the model generated data with data collected within their schools. For instance, one group compared the data generated using their model that simulated the spread of swine flu with attendance/absenteeism records collected during a period of time when swine flu was known to be circulating within their school. Analysis of this sort may lead to reconsideration of what factors to include in a model and cycle back to the beginning of the process described above.

2.2 Robotics

A second domain that promotes computational thinking with precollege students is robotics. In a robotics project, student programmers design and program robots and other physical devices with embedded code. They need to think about how the robotic agent will interact within its world, based on factors such as its sensor values and the effects of its actuators. As they do this, the student makes choices of how their programming will connect these processes together to achieve the desired results.

Figure 3: Two iCODE students display their Sumo robot.

In the iCODE project (Internet Community of Design Engineers), middle and high school students 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. See Figure 3.

Abstraction takes place as students design robots to react to a limited set of conditions that may be encountered in the real world. Students think about how to sense the world, and how those stimuli will be abstracted as numerical or true-false values inside the control program. Automation occurs as the students' programs are executed by the embedded computing device. Students perform analysis when they decide whether or not the robot operated as expected in the real-world environment. If the robot "misbehaves," it may either mean that their implementation of their control idea is faulty, or that conditions were encountered that were not taken into account during the abstraction phase.

2.3 Game Design and Development

A third domain in which computational thinking takes place is computer game design and development. In the iGame afterschool program, middle school students engage in computational thinking by designing, programming, and testing computer games of their choosing using Storytelling Alice (SA). SA, as with many other programming languages, allows students to create their own abstractions. Because SA is a programming environment that allows the creation of 3D animations, students can then test highly complex abstractions quickly and precisely. To create their game, students build a group of scenes, where each scene contains characters, and each character has behaviors. Students choose from an array of character attributes and behaviors selecting only those details appropriate for the virtual world they are creating.

Students can define new methods representing behaviors not built into SA. Into these methods, students can place a combination of existing behaviors and changes to character attributes. In iGame, many student-created methods were simple combinations of sequential commands, but creating methods often requires an understanding of conditionals, iteration, and sequential and parallel execution. For example, in the Labyrinth of the Turtle game, a student programmed a character to dance using a series of parallel movements and vocalizations.

Games often require multiple similar characters to perform the same action. Students can "scale up" their abstraction creating a list data structure containing these similar characters. Then they can program similar behaviors for these characters by using special instructions that iterate through a list data structure. For example, in the Zombie Invasion game, a student programmed a group of zombies to wait and then start moving at the same time. As the player clicks on each one, it is programmed to disappear. However, most game programmers in iGame choose to repeat commands they understand, rather than learn to use lists.

Students in iGame engage in analysis when they judge whether or not their abstractions were correct and efficient. Analysis of correctness focuses on whether they produced the game they intended, i.e., whether the game plays the way they want it to or whether the game that was designed to be fun was, in fact, fun. Analysis of efficiency involves creating the simplest code to achieve the desired behavior. Analysis takes place during the process of testing and debugging their game, and students often need an external motivator to focus on efficiency because when their game is working, they see little reason to edit the code. Students also play-test their peers'

games, and analyze them in terms of playability, and whether they have created the best (most efficient, most believable, or most fun) abstractions.

2.4 Other Domains

The domains mentioned are by no means a comprehensive list. There are many other domains such as designing and programming webpages, cell phone apps, etc. that have potential to develop CT in youth. Common among these examples is the active use of key computational thinking concepts: abstraction, automation and, to various extents, analysis, by youth within middle school programs. Through these examples, we posit that not only is CT possible at the middle school level, it can easily be embedded within activities that encourage youth to be creators, innovators, and problem-solvers. Computational thinking projects like these support an iterative cycle of refinement that enables increasing a sense of agency, where learners are empowered to imagine, create, play, share, and reflect on what they are learning [9]. In all of these projects, the end result is a unique product created by the students.

3 SUPPORTING GROWTH IN COMPUTATIONAL THINKING

Based on our experiences with youth learning CT both during the school day and out- of- school contexts, we suggest concrete steps that can be taken to support the development of computational thinking.

3.1 Rich Computational Environments

The first is the use of rich computational environments. Rich computational environments are ones in which the underlying abstractions and mechanisms can be inspected, manipulated and customized. For example, consider the differences between SimCity and a model in StarLogo TNG. In SimCity, a user may add buildings to a city and see correlations between adding buildings and CO2 production but the underlying formulae and model are hidden from view. Contrast that with a StarLogo TNG environment in which the user can "look under the hood" and inspect the causal relationships and abstractions that are embedded in a model. The rich computational environment is one in which the user can develop CT skills and transform from user to creator. See Figure 4.

Figure 4: Inspecting the mechanism for infection in a basic contagion model in StarLogo TNG.

3.2 Three-stage Progression "Use-Modify-Create"

Second, we propose using a three-stage progression for engaging youth in CT within these rich computational environments. This progression, called Use-Modify-Create, describes a pattern of engagement (see Figure 5) that was seen to support and deepen youth's acquisition of CT in the authors' NSF projects. It is based on the premise that scaffolding increasingly deep interactions will promote the acquisition and development of CT. In the use stage, students are consumers of someone else's creation. For example, they run experiments using pre-existing computer models, run a program that controls a robot, or play a ready-made computer game. Over time they begin to modify the model, game or program with increasing levels of sophistication. For example, a student may initially want to change the color of a character or some other purely visual attribute. Later the student may want to change the character's behavior in a

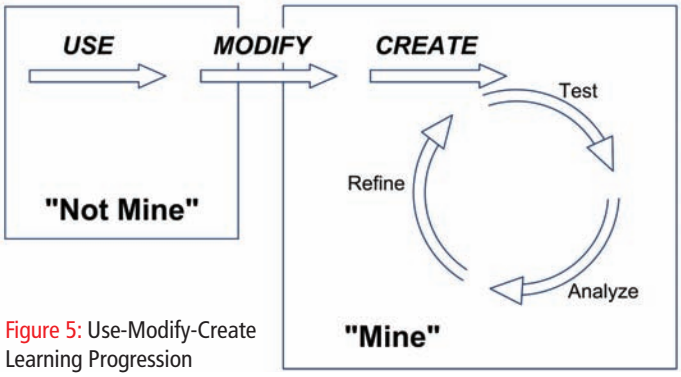

Figure 5: Use-Modify-Create Learning Progression

way that entails developing new pieces of code. Modification of this kind necessitates an understanding of at least a subset of the abstraction and automation contained within a program, model or game. Through a series of modifications and iterative refinements, new skills and understandings are developed as what was once someone else's becomes one's own. As youth gain skills and confidence, they can be encouraged to develop ideas for new computational projects of their own design that address issues of their choosing. Within this "create" stage, all three key aspects of computational thinking: abstraction, automation and analysis, come into play.

Moving through this progression, it is important to maintain a level of challenge that supports growth while limiting anxiety. As Repenning [8] notes, students can maintain their sense of cognitive flow [1] as they progress iteratively through a series of projects. In this work, students tackle progressively higher design challenges as their skills and capacities increase. Activities that were once "too hard" and were anxiety-inducing become possible with appropriate, incrementally challenging experiences. Conversely, boredom will set in if challenges don't keep pace with growing skills [8]. While we are advocating use of this three-stage progression to foster growth over time and with increasing capacity, we also raise a caution about taking it too literally. Just as an early teenage youth is moving from childhood to adolescence in fits and starts, there are no clean break points from using to modifying to creating. Youth may transition back and forth from users to modifiers to creators.

3.3 Other Domains

The examples of CT in youth programs described thus far took place after school, either during weekends, holiday breaks, and/or over the summer. EcoScienceWorks (ESW) is a program in Maine that leverages the State's one-to-one laptop initiative to engage students with environmental simulations as part of the school day science curriculum. This project also exposes students to simple programming challenges as a way of introducing them to the computational thinking that underlies the simulations. Through guided experimentation, EcoScienceWorks deepens students' understanding of both ecology and computer modeling.

The success of the project has been partly a result of addressing some of the challenges in introducing computational thinking into the classroom head on. For instance, because CT is not evaluated by standardized testing, it is difficult in the current educational climate for teachers to teach CT concepts directly. The ESW staff addressed this constraint by designing a simulation-based ecology curriculum in which the CT portions of the curriculum were what students had to do in order to fulfill explicit content learning goals. That is, an ecology curriculum that arguably required CT was designed *to replace* the existing curriculum that focused on content transfer. With this pedagogical design, the required core ecology concepts could be covered in much greater depth, and CT was fostered through the use and understanding of computational models.

While this work offers a promising example, it is important to recognize the resources that were necessary in order for it to be successful. Infrastructure was not a significant obstacle as each student had access to a laptop, district support had been established, and intensive support was provided by the project staff in the form of professional development and ongoing assistance. Transformative applications of CT can work in schools with all of these ingredients in place.

Implementing CT during the school day is a compelling vision, but there are substantial challenges to this, including existing curriculum standards, lack of opportunities for teachers to learn CT as part of their professional development, and lack of access to necessary infrastructure. Consequently, much of the work in CT with youth remains in out-of-school environments. As shown in this paper, new opportunities for fostering computational thinking are emerging, and NSF-funded programs are actively exploring ways in which computational thinking works in both in-school and out-of-school environments.

Implementing CT during the school day is a compelling vision, but there are substantial challenges to this, including existing curriculum standards, lack of opportunities for teachers to learn CT as part of their professional development, and lack of access to necessary infrastructure.

4 CONCLUSIONS

The call for integrating CT into K-12 settings has been growing increasingly louder, despite the lack of descriptions of what learning to think computationally actually looks like among youth. In this paper, we have contributed to efforts to define and support CT for youth by using examples from several youth projects to make two key points.

The first key point is that existing definitions of CT can be applied to K-12 settings. The examples show that youth can engage in key aspects of computational thinking within programs focusing on modeling and simulation, robotics, and game design and development. Students from a range of backgrounds are able to use abstraction, automation, and analysis to create original products when given access to rich learning environments that include skilled teachers, developmental considerations, and usually include new technology. However, the field requires systematic assessment procedures that build on existing research from the learning sciences in order to describe the developmental progression of these three CT constructs. Some of the authors are currently testing a variety of assessment approaches.

The second key point is that CT takes place on a continuum. The use-modify-create progression is offered as a framework for educators and researchers that are looking at how CT develops, and how that development can be supported. But research is needed to understand why students are thinking at different levels of abstraction, automation, and analysis. These differences may be a function of students working in different phases of the use-modify-create learning progression. For example, we suggest that moving from modifying to creating an original project requires increasing levels of abstract representation and understanding. Similarly, simple analysis includes testing and debugging a program, while a deeper level of analysis would involve trying to determine if a model can be validated against real-world data. As a foundation moving forward, the use-modify-create framework offers a helpful progression for developing CT over time. Its greatest benefit is in illustrating the benefits arising from engaging youth with progressively more complex tasks and giving them increasing ownership of their learning.

This paper aims to inform efforts to engage K-12 students in CT, and to assess the value of these efforts. We recommend that future efforts get more specific about the type and level of CT that will be addressed. The CS Principles project has moved this effort

forward in the context of high school CS classes, and a recent posting by Snyder [10] describes specific computational thinking practices. We are working on contributing to a similar effort in OST.

This paper builds on existing efforts to describe the scope and nature of CT [7] as well as the concepts involved in CT and how youth should be able to use those concepts [2]. We hope this paper will contribute to a national dialogue about the most important dimensions of CT in K-12, how different aspects of CT develop, the role of context and motivation in this development, and effective strategies for engaging youth in computational thinking. **Ir**

Acknowledgments

Portions of this paper were adapted from *Computational Thinking for Youth*, ITEST Working Group on Computational Thinking (2010) with permission from Education Development Center, Inc., Newton, Massachusetts.

References

[1] Csikszentmihalyi, M. (1990). *Flow: The psychology of optimal experience.* New York: Harper.
[2] Computational Thinking Thought Leaders Meeting. (2010). Computer Science Teachers Association & International Society for Technology in Education. April 18-19, 2010.
[3] Cuny, J., Snyder, L., and Wing, J. (2010). *Computational Thinking: A Definition.* (in press)
[4] Henderson, P.B. (2009). Ubiquitous computational thinking. Computer, 42(10), 100-102. Available online at *http://www.computer.org/portal/web/csdl/doi/10.1109/MC.2009.334.* Accessed June 29, 2010.
[5] Lu, J.J. & Fletcher, G.H.L. (2009). Thinking about computational thinking. ACM Special Interest Group on Computer Science Education Conference, (SIGCSE 2009), (Chattanooga, TN, USA), ACM Press. Available online at *http://portal.acm.org/citation.cfm?id=1508959&dl=ACM&coll=portal.* Accessed June 29, 2010.
[6] Moursund, D. (2009). Computational Thinking. IAE-pedia. Available online at *http://iae-pedia.org/Computational_Thinking.* Accesed August 8, 2010.
[7] National Academies of Science. (2010). *Report of a workshop on the scope and nature of computational thinking.* Washington DC: National Academies Press.
[8] Repenning, A. and Ioannidou, A. (2008). *Broadening participation through scalable game design,* ACM Special Interest Group on Computer Science Education Conference, (SIGCSE 2008), (Portland, Oregon USA), ACM Press. Available online at http://www.cs.colorado.edu/~ralex/papers/index.html accessed April 19, 2010.
[9] Resnick, M. (2007). *All I really need to know (about creative thinking) I learned (by studying how children learn) in kindergarten.* ACM Creativity & Cognition conference, Washington DC, June 2007. Available online at http://web.media.mit.edu/~mres/papers.html accessed April 19, 2010.
[10] Snyder, L. (2010). Seven Practices of Computational Thinking. Available online at http://csprinciples.cs.washington.edu/sevenpractices.html accessed August 2, 2010.
[11] Wing, J. (2006). Computational thinking. *Communications of the ACM 49*(3), 33-35.

IRENE LEE
Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, New Mexico 87505 USA
lee@santafe.edu

FRED MARTIN
University of Massachusetts Lowell, Olsen Hall Rm 208, Lowell, Massachusetts 01854 USA
fredm@cs.uml.edu

JILL DENNER
ETR Associates, 4 Carbonero Way, Scotts Valley, California 95066 USA
jilld@etr.org

BOB COULTER
Missouri Botanical Garden, PO Box 299, St. Louis, Missouri 63166-0299 USA
bob.coulter@mobot.org

WALTER ALLAN
Foundation for Blood Research
ScienceWorks for ME
8 Science Park Road, Scarborough, Maine 04070 USA
allan@fbr.org

JERI ERICKSON
Foundation for Blood Research, P.O. Box 190
8 Science Park Road, Scarborough, Maine 04070-0190 USA
jerickso@maine.rr.com

JOYCE MALYN-SMITH
ITEST Learning Resource Center, Education Development Center
55 Chapel Street, Newton Massachusetts 02458-1060 USA
jmsmith@edc.org

LINDA WERNER
University of California, Santa Cruz, Baskin Engineering
1156 High Street, Santa Cruz, California 95064 USA
linda@soe.ucsc.edu

Categories and Subject Descriptors: K.3.2 [Computers and Education]: Computer and Information Science Education – *Computer science education, curricula, literacy.*
General Terms: Human Factors, Performance, Design, Experimentation.
Keywords: Computer science education, computational thinking, abstraction, automation, analysis.

DOI: 10.1145/1929887.1929902