Computational Thinking for Youth

The ITEST Small Group on Computational Thinking
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

The term "computational thinking" has been at the center of recent efforts to describe and promote new ways of thinking in an increasingly digital age. But the term remains elusive to those outside the field of computer science. In this paper, we aim to describe what computational thinking looks like among youth from a range of backgrounds, as they engage in innovative uses of technology. The term computational thinking (CT) was coined by Jeannette Wing (2006) to describe a set of thinking patterns that involve 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 understanding the dimensions of scale. While the concept has emerged from computer science, students can engage in CT with or without a computer. CT draws on a rich legacy of studies of human cognition, such as systems thinking, problem solving, and design thinking.

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. Many believe that today's youth are developing new patterns of thinking through their long term, intensive use of technology in and out of school (Jukes and Dosaj, 2004, Ryberg, 2005; Sørensen, Danielsen & Nielsen, 2006; Yoon, 2007; Ryberg and Dirckinck-Holmfeld, 2008; Sørensen, 2010). In this paper, we examine the learning potential of CT through concrete examples—in and out of school—of youth engaged in computationally-rich programs in which they play the roles of designers, creators and innovators.

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. To that end, this paper addresses two essential two questions:

- *What does computational thinking for youth look like in practice?*
- *How can educators support growth in computational thinking?*

This effort is intended to complement The National Academies "Computational Thinking for Everyone" workshop series and the Computational Thinking Thought Leaders project currently being carried out by the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE). We are confident that synergy will emerge from our respective efforts, helping to chart a path forward.

What does computational thinking for youth look like in practice?

In this paper, we focus on how computational thinking ideas have value for pre-college youth, in and out of school. 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 two main concepts: abstraction and automation. As described by Wing, abstraction is about a "separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable" (2006) and automation allows one to "tackle problems of scale" (2009).

Phrased more tangibly, Dave Moursund (no date) 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 robotic systems and game design and development. We see examples from these three domains as important, as much of the innovative work in computing being done with middle- and high-school students draws from these three areas. The boundaries defining game design, simulations or models, and robotics are fluid, intended to provide a thought framework. Since the three domains have intricate connections with one another, we feel that there is a synergy arising from considering each in building a more general understanding of CT for youth.

Modeling and simulation

Modeling and simulation environments are typified by a system of interacting components that together create a larger whole. Users may interact with the system simply by allowing the simulation to unfold and observing it. Deeper interactions involve adjusting parameters, or in some cases, reprogramming the behaviors of particular agents within the simulation. In this latter case—where the user takes charge of determining the behavior of an agent or agents within the simulation—the example crosses over into the domains of robots and/or games.

> *In a Project GUTS (Growing up Thinking Scientifically) middle school program, students actively engage in computational thinking through the modeling and simulation of real-world issues within their communities. In Project GUTS clubs, students investigate local issues within their community. As part of this work they create agent-based models in StarLogo TNG to explore the underlying dynamics of the issues and to test potential mitigation strategies virtually. For example, in an investigation of epidemics, club members collected data on the physical layout and student circulation within their schools, and conducted background research on 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.*

Game design and development

Building a computer game is a complex task, requiring not only programming, but also the ability to think at multiple levels of abstraction and in terms of scale. More broadly, 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). This challenge can be highly appealing to youth.

> *In the iGame after school program, middle school youth engage in computational thinking by programming original computer games using Storytelling Alice. In iGame classes, students engage in relevant CT concepts such as algorithmic thinking, as they solve problems related to game programming using conditionals, iteration, and sequential execution. Students must also think abstractly to create a model of their world, and set up variables to define the state of the world. For example, in the game “Are you Smarter than a Penguin?” students simulate a game show by programming a series of questions, where the course of the game is determined by the player’s responses..*

Robots and robotic systems

In the case of robotics, students are most often concerned with determining the behavior of a single agent in the system that is equipped with sensors and actuators that enable it to interact with or respond to its immediate environment. These systems may be mobile robots, or may consist of other actuators, such as flashing lights or sound synthesizers that interact directly with people. Student programmers need to think about how the robotic agent will interact with 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 things together to achieve the desired results.

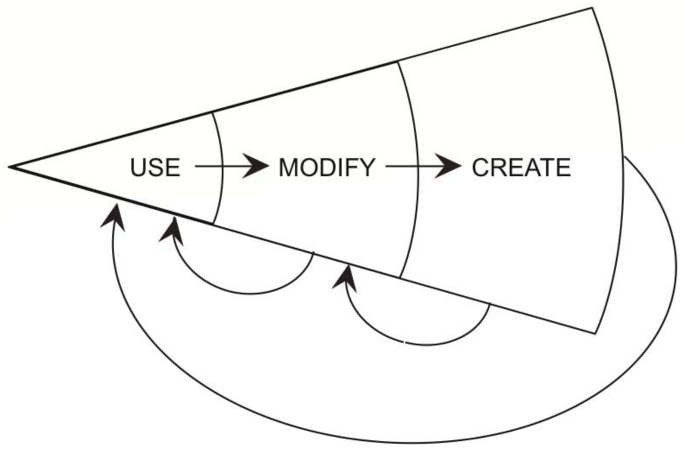
> *In iCODE (Internet Community of Design Engineers) project, middle and high school youth complete a variety of microcontroller-based projects. They meet once per week in an after-school program, and for two weeks during the subsequent summer. With the guidance of a program leader, iCODE participants work from a set of online project guides, which present design ideas and project plans. Students begin with a simple project consisting of programmable flashing lamps, and progress to a musical memory game, to finally a fully autonomous (self-controlled) robot. Students present their work at an informal “Botfest” design show and a separate, Sumo robot contest event* (Martin *et al.*, in press).

The examples cited are intended to illustrate how activities within each of the three domains promotes computational thinking. As stated previously, there is a degree of overlap among the domains. For example, within a game design project, students create a model of a real or make believe world, typically with many agents on-screen, just as in a simulation. Student game designers are also directly involved in programming the behaviors of these agents—just as they would program the individual robot—with the intention of achieving a game narrative with multiple possible outcomes, each contingent on a player’s actions.

Note that there are also many examples of explicitly designed “cross-over” applications that consciously draw from more than one domain as they support development in computational thinking. For example, Squire (2004) has shown how the simulation/game Civilization promotes historical understanding. After playing the game, students then use the game’s modification tools to create their own game scenarios. Likewise, the Community Science Investigators (CSI) program engages youth in “augmented reality” games that provide an overlay of an environmental mystery 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 CSI program, participating students build on their experience with simulations to design their own games.

How can we support growth in computational thinking?

Based on our work with several youth projects across the US, we propose a three-part framework of how youth develop CT skills. Implicit in the previous discussion is the premise that growth in computational thinking often begins with *using* rich computational environments. Over time we continue to use the environments of course, but also begin to *modify* them with increasing levels of sophistication. As youth gain skills and confidence, they *create* computational models based on original designs. Each “stage” is essentially a superset of what has come before.



The Use-Modify-Create Framework

USE: During this phase, youth learn how to use the technology, including the interface and tools, and the kinds of products that others have made. This may involve performing scripted tutorial operations and exploring software. Work at this stage builds the foundation for higher levels of engagement with CT.

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

CREATE: In this phase, youth apply their growing computational thinking skills to create a substantially original product. This work will show increasing levels of abstraction and automation than may have been present in an earlier exploratory experience. Implicit in the development, of course, is that the creation will be used and modified over time.

While we are advocating use of this "use-modify-create" framework to describe growth in CT, it is not intended to depict three distinct phases. In practice, these changes are as much a metamorphosis as a clean transition. Just as a "tweenage" youth is moving from childhood to adolescence, there are no clean break points among using, modifying, and creating. When learners are modifying a project, they will still be using it. Likewise, creations almost always build on a student's previous experiences using and modifying project resources.

Over time, students will traverse through these phases recursively, looping back through other phases as they build-their capacity for innovative creation over time. Some learners will progress sequentially through mastery of each stage, while others might dive right in and modify an example before having an extensive "use" experience. Just as children can learn to create music with an instrument before learning music composition, there is no one pathway to developing computational thinking. Being immersed in a rich learning environment is the key, supported by appropriate tools and a collaborative culture.

To that end, we argue that learning spaces that enable students to interact thoughtfully with well-designed modeling, robotic design, or game programming environments can offer a leap forward in computational thinking in comparison to work done with traditional media, such as a textbook. The real value for CT comes, however, when students take ownership of a project. They can then investigate the computational representation (including its underlying assumptions), adjust these representations or assumptions, and see what the impact might be. With even a modicum of experience modifying existing projects, learners have a stronger basis for building their own. A closed simulation is a latter-day version of the black box technologies that Dewey railed against nearly a century ago as the industrial era gave people automated machines, closing off avenues to see and feel how things really worked. Dewey's concern is just as valid today, as considerable computational power is housed in sealed boxes, some small enough to fit in our pockets. To realize the full benefit of computational thinking, students need a level of ownership and interactivity with the models underlying the abstractions and automation.

To illustrate this growth in capacity, we offer two examples of how middle school students might engage in CT in each phase of the framework. The first example is from the previously-mentioned iGame after school program where students learn to create digital games. First, they learn how to use the Storytelling Alice programming environment through interactive tutorials, and by playing games made by peers. The goal is to introduce the software interface, and the kinds of games they might make. In the Modify phase, students adapt and expand on existing programs through a series of self-directed “challenges” that deepen understanding of the mechanisms used to program an original game. The challenges get increasingly difficult with more complex and abstract concepts, and with fewer and fewer instructions. Along the way, participants learn different aspects of CT including key programming concepts such as variables and conditionals. In the Create phase, students program original games, with varying degrees of complexity. There is a continuum of sophistication within this phase, with some engaging in high-level abstraction (creating complex new methods or embedded loops) and others creating more linear code. For instance, many apply the concept of conditionals using simple If/Else commands, while others use nested If/Else commands, suggesting a high level of mastery of these concepts. Through iGame, this involves not only creating, but also analyzing, testing, and revising a variety of games made students and their peers.

In a second example, EcoScienceWorks is a project for middle school science classes that begins with packaged environmental simulations (Allan, et al, in press). During the Use phase of the project, students learn the basics of software interface as they discover important features of the simulated habitat and perform directed experiments. For instance, the microscope tool can hover over an icon, enabling students to discover an organisms’s “gut contents” as they work out the habitat’s food web. In the eutrophication lab students discover the impact of different levels of phosphorus within a lake ecosystem on population sizes for algae, zooplankton and trout, and uncover an explanation for the decline in trout population by measuring the lake’s dissolved oxygen content. This “use” phase of the project is rich in the core CT skill of abstraction, developing students’ understanding of concepts such as the 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 “Program a Bunny” challenge lab comes next. 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 a bunny with student-created code and its pre-programmed opponent.

In these examples, learners progress from using to modifying and eventually to creating, with each growth increment contingent on what has gone before. This development depends on both an active student and a conducive learning environment. As noted by the National Academies of Science (2010) report on computational thinking, the full realization of the affordances provided by a computational thinking environment depends on the education, training, and experience of the user. With this in mind, we turn to consideration of the learning spaces—in school and out—that nurture budding computational thinkers.

Learning CT in School and Out

Learning environments such as the ones described here offer many compelling advantages, but they also pose some real challenges to implementation in many school settings. Consequently, as our working group explored CT, we found our richest examples occurring in out-of-school (OST) environments. This is likely due to a confluence of factors that typify many school environments, including:

- Balancing curricular demands;
- Building teacher capacity;
- Ensuring infrastructure access.

These inter-related challenges have constrained many previous educational innovations, and CT is likely no different. Our comments that follow describe some of the changes we feel are needed for CT to flourish in formal K-12 school environments and in the growing variety of informal learning opportunities available to youth. Where we raise concerns, they are meant to provide a vision of alternatives, fully recognizing that there are examples of successful practice that can guide our collective efforts.

Like others (Henderson, 2009; Lu & Fletcher, 2009), we advocate for constructive engagement during the regular school day with technology when this advances educational goals. The National Council of Mathematics (1995) has campaigned for years on behalf of students being equipped to do *different* mathematics, not just the same mathematics with technology. The key is in noting how the learning potential is enhanced through the creative interplay of technology and the students. Creative use of technology facilitates a qualitatively different learning experience. The same premise advanced by NCTM holds for any discipline. As we have illustrated in this paper, computational thinking involves much more than just adding computers to an existing program. A core operating principle of computational thinking is that learners need opportunities for thoughtful, reflective engagement with the phenomena represented.

To draw a simple contrast, virtually every middle school student “knows” from their textbook and common knowledge that trees help mitigate pollution. Students in an after-school program in Missouri have a chance to go much further, using CITYgreen modeling tools to map the trees in their school yard and record relevant data on species, health, growing conditions, and the like. With this abstraction of their schoolyard created in the form of maps and data tables, automated models calculate the benefits of the trees in terms of pollution removal and runoff mitigation. These students can model alternative growth scenarios as they either “plant” new trees in the model, let the existing trees continue to grow, or remove the trees for expanded parking. Re-running the model leverages the power of automation to quickly adjust the underlying parameters and see what the impacts are.

Computational thinking projects like these support an iterative cycle that enables an 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 harder 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 opportunities to use technology in ways 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—implicitly suggesting engagement with a longer-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.

Looking at issues relating to teacher capacity, enabling CT in schools requires teachers to have a set of skills that are not currently taught in most teacher education programs. Most simply, teachers need to be computational thinkers themselves in order to teach it effectively. Thus, providing in-service training is needed but fraught with challenges. Research has demonstrated the challenges of simultaneously asking teachers to change their pedagogy, increase their content knowledge to support richer inquiry, and embed advanced technologies in their professional practice (Feldman, Konold, and Coulter, 2000).

Making this change in the context of the numerous demands of regular teaching assignments makes implementation of CT in schools on a wide scale a daunting task. In the tree investigation just cited, any teacher could transmit and test whether students could repeat the proposition that trees help mitigate pollution. However, leading a CITYgreen investigation requires integration of field and classroom study, as well as the ability to support students in the use of specialized equipment for measurement, species identification, and data recording. As the project progresses, technology skills in the use of the underlying ArcView GIS software and CITYgreen extension are required, as is the capacity to support students in thinking about the data embedded in the results and in posing "what if?" questions. Emerging work in "technological pedagogic content knowledge" or TPACK (Koehler and Mishra, 2008) provides useful direction, but leading a rich learning environment informed by computational thinking requires a particular skill set that requires more than a single teacher workshop can provide. With this in mind, creating a path for teacher professional development in CT is a challenge that will need to be addressed.

Establishing technological infrastructure and leveraging it for rich learning experiences is a long-term process. Many schools have established computer labs and/or purchased laptop carts, but use of these poses a logistical challenge for teachers, who must plan use well in advance. Fortunately, new program models are emerging with more ubiquitous access, such as the schools with one-to-one computing initiatives where each student has a laptop for personal use, or where youth have access to handheld computers. In these situations, emergent investigations and more casual use of technology can flourish. Lessons learned from these experiences will no doubt be invaluable in guiding CT efforts as access becomes less of an issue.

One promising example of school-day implementation is the EcoScienceWorks program described earlier that leverages Maine's one-to-one laptop initiative to engage students with environmental simulations. 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 allocate time to teach CT as a stand-alone unit. The EcoScienceWorks staff addressed this limitation by designing a CT-rich, simulation-based ecology curriculum *to replace* the existing curriculum that focused on content transfer. In this way, required ecology concepts could be covered in much greater depth, and computational thinking skills fostered through the use and understanding of models.

While this work offers a promising example, its success required a comprehensive approach to reform. Infrastructure was provided by the state's laptop initiative, a partnership with the school district supported curriculum reform, and intensive support was provided by the project staff in the form of professional development and ongoing assistance. The key lesson here is that CT integrated into the school day curriculum can work with a systemic, strategic approach.

Aside from EcoScienceWorks, most of the examples in this paper are drawn from out of school time (OST) environments. With few curricular constraints, the capacity to hire staff with the requisite technology skills, and the ability to dedicate the necessary technological infrastructure to the project, it is not surprising that many of the best examples of CT-rich learning occur outside of a traditional school day. However, this is not to say that such programs are the ideal environment. Instead, the benefits come with trade-offs that must be acknowledged. 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 access becomes more common in rural areas, virtual learning opportunities won't provide meaningful programs either, further exacerbating the opportunity gap faced by rural communities. Inner-city areas also face problems specific to their location, such as chronic underfunding, security of expensive hardware, and the safety of youth participants.

Another limitation of OST is that many of the most ambitious programs are funded through expensive, time-limited grants from government and foundation sources that serve only a very small portion of the potential pool of participants. Continuation past the grant cycle is often dependent on next grants, as is replication in other locations. Without grants, continuation 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. An additional concern is the fact that 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, it is important to identify ways to make CT environments more universally accessible through the school environment. In most industrial societies, schools remain the primary means of reaching young people. Fundamental changes will be required in the nature of schooling, including decisions about who is teaching and what is taught, all built upon a better understanding of what and how kids learn. As Papert (2005) notes, we need to "recognize that districts have not been transformed; recognize that when districts talk of access it's inflated; and recognize that districts must explore ideas of what should be learned at what age."

In short, we recommend a two-pronged approach. We should leverage what is possible in both formal and informal learning environments, and use each to inform the other. To that end, in the next section we share lessons learned and offer 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 projects to describe what CT looks like, and to consider strategies for engaging youth in CT, both in and out of school. Given the importance of the work before us, 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 contributing to 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 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. 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 youth develop CT over time. Its greatest benefit is in illustrating the value of engaging youth with progressively more complex tasks and allowing them to take increasing ownership of their learning. Traditional assembly line models of education don't foster this depth or agency on the part of teachers or students. For schools to support computational thinking, changes are needed in curriculum, accountability, professional development, and infrastructure.

Recommendations for Next Steps

Recognizing the challenges educators face, we make several recommendations for next steps in practice, and for focused research that will enable us to better understand the issues and challenges involved. These recommendations are based on two related premises:

(1) Teachers and learners need to have and be able to exercise a reasonable degree of agency or the capacity to make a difference in their surroundings. Using Emirbayer and Mische's framework (1998), we exercise agency through making practical / evaluative judgments, drawing on an iterative cycle of previous experiences and directed toward projected outcomes. For example, a Project GUTS student making sense of pollution in her community needs to make a series of judgments in her modeling, drawing on previous experience with pollution and with the modeling software. In a sense, this element builds on Hawkins' (1974) notion of "messing about" and Papert's bricolage (1993). The take home message here is that new learning doesn't come out of nowhere—prior experiences give the building blocks for concept development, but only if they can be accessed and drawn on intelligently. Banking or warehousing models premised on "learn it and store it for later" are not sufficient.

Exercising agency, or actually making a difference in your work, is also guided by a projected outcome--knowing what a "win state" or a desirable outcome would be. In the pollution example, this might be a model that includes the key variables or parameters; projection in a robotic project would likely be a vision of what tasks the robot can perform. In any case, a learner with agency can make the judgments needed to realize the projected goal, building on iterations of previous experience. A teacher exercising agency reflects on her practice and makes changes in the timing and sequence of activities to better realize her projected success—in this case, high quality CT-enhanced student investigations.

(2) Implicit in the idea of agency is the notion of continuous growth and development. In the flow model cited previously, challenges need to increase to keep pace with growing skill. A learner exercising agency will—over time—exercise more complex judgments and build toward more ambitious projections as the iterative pool of experience grows. Framing this issue, Dweck (2000) draws a critical distinction between *incremental* and *entity* thinking. A teacher or student who is exercising agency is growing incrementally as he uses iterations of previous experience and refined practical / evaluative decision making. Employing his sense of agency, he can make increasingly sophisticated and relevant projections of future courses of action, with the result of each projection feeding back into the "pool" of iterations that will inform future efforts. Conversely, someone with a low sense of agency who feels that they can't make much of a difference will remain stuck where they are. This entity mindset can be seen in the all-too-common complaint "I'm just not good at..." Computational thinkers are incremental learners with a strong sense of agency. Framed within Self Determination Theory (Deci, 1995), they develop competence, autonomy, and relatedness as they build skills, act on them, and build a strong learning community that shares ideas freely.

Curriculum Recommendations

To integrate computational thinking into the in-school experience, strong examples need to be connected to curriculum standards. These efforts will help to create a vision of what is possible within a standards-driven curriculum. As noted in the EcoScienceWorks example, it is possible to promote computational thinking within a traditional academic study. To move from being able to recite propositional phrases like "trees are good for the environment" to using models to build understanding of how those trees mitigate pollution is to begin thinking computationally. In current school practice, few curriculum units embrace this richness, however, as they provide surface-level coverage of a broad range of topics. To move toward an educational culture that supports meaningful computational thinking, curriculum models will need to move away from covering a plethora of discrete topics and toward engaging learners in more fully integrated, intellectually rich clusters of ideas.

An example of a rich idea cluster is an ecology unit that uses models and simulations to tie together a range of formerly discrete topics including adaptations, form and function, biotic-abiotic interactions, variability, and heredity. As students move from being users of the simulation toward making modifications and original creations, their understanding of the core concepts and their interdependencies will increase. Most students can recite a definition of protective coloration; a more advanced science learner will see how the protective coloration favors survival in comparison with species members lacking this coloration. A model in which those with the "right" colors survive longer promotes a deeper understanding of adaptations, form and function, and evolution. The key point here is that computational thinking doesn't have to come at the expense of rich content.

This deeper understanding of the underlying concepts is also much more likely to be portable to novel situations, both in the robustness of the conceptual networks, and in the general disposition toward learning and understanding that this work fosters. As discussed previously, learners' sense of agency and disposition toward continuous growth are fundamental building blocks of computational thinking. Curriculum structures that better support this growth and agency need to be developed, thereby enabling students to take ownership of their learning and build their capacity for self-directed learning over time.

Accountability Recommendations

Embedded within most current school curricula is a vision of accountability that all too often tests for recitation of many discrete concepts instead of looking for the ability to draw on rich networks of understanding. This is driven by a standardized testing model that seeks to promote a rigorous and challenging program, but in practice often leads to a fragmented and over-packed curriculum (Ravitch, 2010). We believe that CT will be better accommodated within an accountability structure premised on fewer, deeper learning goals (e.g. Popham, 2001) and that relies on students demonstrating their competence in practical, meaningful applications (e.g. Meier, 1995). Models of authentic assessment need to be employed so that teachers can better understand the richness and depth of students' understanding, and be equipped to use this information in improving their own practice and in providing constructive feedback to the students.

Teacher Professional Development Recommendations

Few K-12 educators have benefited from many years of training and acculturation into a community of computational thinkers. Thus, greater understanding and awareness of computational thinking must dovetail with increased motivation to embed CT-rich learning into the curriculum. As much as we support full integration of CT into the curriculum, change in this regard cannot be forced on teachers as if they were obstacles to be overcome. Teachers will need ongoing support and nurturing to build their own level of comfort with computational thinking as they work toward ownership, integrating computational thinking in their classes. A key element within this is true integration within the teachers' worldview and pedagogy.

If computational thinking is truly foundational, it will need to be integrated into teachers' core identities and allowed to grow over time. Thus, professional development experiences need to be supported by ready access to user-friendly software and instructional resources that scaffold teachers' formative efforts. Effective mentoring and support will also be required. Just as the Use-Modify-Create framework applies to student efforts, a similar model is essential in supporting teachers' efforts to promote computational thinking. The pedagogic and technical challenges they face need to provide just the right level of challenge to avoid boredom or frustration.

Real change takes time, but building support for powerful learning experiences is worth the investment.

Acknowledgment
The authors appreciate helpful comments that were made by several colleagues who informally reviewed this paper. In particular, we wish to thank Leslie Conery, Joanna Goode, Eric Klopfer, and Mitchel Resnick. Any inconsistencies or lack of clarity that remains in the paper is our responsibility.

References

Allan, W., Erickson, J., Brookhouse, P., and Johnson, J. (in press). *EcoScienceWorks: Teacher Professional Development Through a Collaborative Curriculum Project - an Example of TPACK in Maine.* TechTrends .

Collins, A. and Halverson, R. (2009). *Rethinking education in the age of technology.* New York: Teachers College Press.

Csikszentmihalyi, M. (1990). *Flow: The psychology of optimal experience.* New York: Harper.

Cuban, L. (1986). *Teachers and machines: The classroom use of technology since 1920.* New York: Teachers College Press.

Deci, E. (1995). *Why we do what we do.* New York: Penguin.

Dweck, C. (2000). *Self-theories: Their role in motivation, personality, and development.* New York: Psychology Press.

Ed leaders talk tech (2005). *District administration* June 2005. Available online at http://www.accessmylibrary.com/coms2/summary_0286-9511617_ITM. Accessed April 19, 2010.

Emirbayer, M. and Mische, A, (1998). What is agency? *American journal of sociology 103*(4), 962-1023.

Feldman, A., Konold, C., and Coulter, B. (2000). *Network science a decade later: The Internet and classroom learning.* Mahwah, NJ: Lawrence Erlbaum.

Hawkins, D. (1974). *The informed vision.* New York: Agathon Press.

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.

Klopfer, E., Scheintaub, H., Huang, W., & Wendel, D. (2009). StarLogo TNG: Making agent based modeling accessible and appealing to novices. In Komosinski, M. (ed.) *Artificial life models in software* (2nd edition). New York: Springer.

Koehler and Mishra (2008). Introduction to TPCK, in *Handbook of technological pedagogic content knowledge.* AACTE Committee on Innovation and Technology, eds. (pp. 1-21). New York: Routledge.

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.

Martin, F., Scribner-MacLean, M., Christy, S., Rudnicki, I., Londhe, R., Manning, C., Goodman, I. (in press). “Reflections on iCODE: Using Web Technology and Hands-On Projects to Engage Urban Youth in Computer Science and Engineering.” Forthcoming in *Autonomous Robotics*, Springer.

Meier, D. (1995). *The power of their ideas.* Boston: Beacon Press.

Moursund, D. (no date). *Computational thinking*. Available online at: http://iae-pedia.org/Computational_Thinking. Accessed June 29, 2010.

National Academies of Science. (2010). *Report of a workshop on the scope and nature of computational thinking.* Washington DC: National Academies Press.

National Council of Teachers of Mathematics. (2005). *Technology supported learning environments* (Sixty-seventh yearbook). William Masalski, ed. Reston, VA: Author.

Papert, S. (1993). *Mindstorms* (2nd Ed.). New York: Basic Books.

Papert, S. (1998). Does easy do it? Children, games and learning. *Game Developer* (June 1998). Available online at http://www.papert.org/articles/Doeseasydoit.html. Accessed April 19, 2010.

Popham, J. (2003). *Test better, teach better: The instructional role of assessment.* Alexandria, VA: ASCD.

Ravitch, D. (2010). *The death and life of a great American school system: How testing and choice are undermining education.* New York: Basic Books.

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.

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.

Resnick, M. (2008). *Falling in love with Seymour's ideas.* American Educational Research Association (AERA) annual conference, New York. Available online at http://web.media.mit.edu/~mres/papers.html. Accessed April 19, 2010.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11), 60-67.

Sørensen, B.H., Danielsen, O., & Nielsen, J. (2007). Informal Learning in the Contexts of Schools of the Knowledge Society. *Journal of education and information technologies*. 12(1), 17-27.

Sørensen, B.H. (2010). 2.0 – Children in and outside school. In Carlsson, U. *Children and youth in the digital media culture*. University of Gothenborg: Nordicom.

Squire, K. and Jenkins, H. (2004). Harnessing the power of games in education. *Insight (3)*1, 5-33.

Wing, J. (2006). Computational thinking. *Communications of the ACM 49*(3), 33-35.

Wing, J. (2008). Computational thinking and thinking about computation. *Philosophical transactions of the royal society A 366,* 3717-3725.

Wing, J. (2009). Computational thinking. *Journal of computing sciences in colleges*, 24(6), 6–7.