Infusing Computational Thinking into the Middle- and High-School Curriculum

Amber Settle
DePaul University
243 S. Wabash Avenue
Chicago, IL 60604
(312) 362-5324
asettle@cdm.depaul.edu

Baker Franke, Ruth Hansen, Frances Spaltro, Cynthia Jurisson, Colin Rennert-May, Brian Wildeman
The University of Chicago Laboratory Schools
1362 E. 59th Street
Chicago, IL 60637
(773) 702-9450
{bfranke, rhansen, fspaltr, cjurisson, crenner, bwildem} @ucls.uchicago.edu

ABSTRACT
In recent years there have been significant efforts to revamp undergraduate and K-12 curricula to emphasize computational thinking, a term popularized by Jeannette Wing in 2006. We describe work introducing and enhancing computational thinking activities and assessments in the middle- and high-school curriculum at the University of Chicago Lab Schools. In total six courses were altered as a part of the Computational Thinking across the Curriculum Project: middle-school and high-school computer science, and high-school Latin, graphic arts, English, and history. We detail the modifications to the curriculum and discuss the successes and challenges of the project.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *curriculum.*

General Terms
Management, Design, Standardization.

Keywords
Computational thinking, K-12, high school, middle school, history, English, Latin, computer science, graphic arts.

1. INTRODUCTION
In her seminal article, Jeannette Wing argues that computational thinking is an emerging basic skill that should become an integral part of education. "To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" [16]. In a subsequent article, Wing formally defined computational thinking as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" [17]. Spurred by the creation of the NSF CPATH (CISE Pathways to Revitalized Undergraduate Computing Education) Program, researchers have considered how to revamp undergraduate courses and entire curricula to place a greater emphasis on computational thinking concepts. A number of CPATH projects have focused on finding ways to enhance computational thinking in undergraduate courses outside the standard computing curriculum [5, 6, 8, 10, 14, 15]. These projects have the potential to expand the number of college graduates who have exposure to computational thinking, an important goal in a world in which computational competency is increasingly important for people in all professions.

However, a small minority of people worldwide attends and graduates from colleges and universities. The potential of any approach to integrating computational thinking into the curriculum will be limited by a focus on undergraduate education. A much greater impact can be obtained by modifying the K-12 curriculum to include a stronger emphasis on computational thinking. This is especially true if the goal is to reach a larger number of students for the anticipated information technology workforce shortage. This idea has been embraced in the United States and has resulted in a great deal of activity in the computer science education community. It has been justifiability recognized that the K-12 arena is a complex and politicized environment, where multiple competing priorities exist [1]. Having outsiders inject computational thinking into an already overburdened curriculum is not a formula for success, and it has been suggested that inviting teachers into the process is a promising technique [19]. Further, a practical approach that demonstrates how computational thinking can be achieved in the curriculum is crucial [1].

Additional complexities in the United States are that computer science is not a core topic covered in high school, and there are too few K-12 computing teachers to implement a national-scale computing requirement [3]. Focusing on high school is a good start but may not be the ultimate goal since motivational concerns need to be addressed as early as middle school [11], particularly for girls and underrepresented minorities. Reaching students who do not consider themselves candidates for Science, Technology, Engineering, and Mathematics (STEM) disciplines is also important if the broadest audience possible is to be engaged [19].

In 2008 we began a project at DePaul University to modify existing general education courses to teach computational thinking concepts in context, funded under the CPATH program

ITiCSE'12, July 3–5, 2012, Haifa, Israel.

described previously. Our project is described in more detail in the next section. A natural extension of our project was to move into the K-12 arena, and in 2010 we began work with teachers at the University of Chicago Laboratory Schools. During the next two years we worked both with teachers trained in computer science and with teachers in the language and visual arts and humanities to develop computational thinking activities and assessments for their courses. The courses are at the middle- and high-school levels for students ages 10 through 18, and the materials were developed by the teachers responsible for the courses in consultation with the principal investigators of the DePaul University project. Using a process of iterative design, refinement, and reflection that is recommended in the literature [1], we found ways to integrate computational thinking into six courses: middle-school and high-school computer science, and high-school Latin, graphic arts, English, and history.

2. BACKGROUND

In the Computational Thinking across the Curriculum project we focused on modifying existing general education courses to teach computational thinking concepts in context. Our approach was to begin with courses closest to computing and work our way out to faculty in non-computing disciplines, creating a framework that all instructors teaching undergraduates could use to deepen their treatment of computational thinking in their courses and discipline. Because computational thinking involves a broad set of approaches and skills and because we were interested in finding commonalities across disciplines, we adopted a set of categories from Peter Denning's "Great Principles of Computing" project [4] for classification. According to Denning, the Great Principles of Computing are: computation, communication, coordination, recollection, automation, evaluation, and design. We used these categories throughout the project to identify types of computational thinking.

During the 2008 – 2009 academic year ten faculty from the College of Computing and Digital Media were recruited. They worked to modify courses from cryptography to screenwriting highlighting or including for the first time computational thinking class activities and assignments. The widest variety of courses possible were chosen so that the materials produced could provide inspiration for the second year of the project. During 2009 – 2010, eight additional faculty members from the College of Liberal Arts and Sciences joined the project. They developed materials for courses in the sciences and humanities. Over the life of the project faculty from eight different departments or schools modified 19 courses across five different areas in the Liberal Studies Program. In a previous publication we discussed the first two years of the project [10], and all of the materials created for the project are available online [9].

In an extension of the project funded through a Research Experience for Teachers (RET) supplement from the NSF three teachers from the University of Chicago Laboratory Schools extended the project into the K-12 curriculum in the academic year 2010-2011. The University of Chicago Laboratory Schools, denoted the Lab Schools in the remainder of this paper, was founded in 1896. The mission statement of the Lab Schools includes among its goals, "to provide an experience-centered, rigorous and well-rounded education for a diverse community" [13]. The school enrolls nearly 1800 students in nursery school through 12th grade, 44% of whom identify as students of color, and there are over 200 faculty employed at the school. The Lab Schools teachers planned to modify three courses, a high-school and a middle-school computer science class and a high-school Latin class, to include computational thinking activities and exercises. Their work then inspired further modifications during the 2011-2012 academic year by three additional teachers. Like the part of the project conducted at DePaul, the second year at the Lab Schools involved reaching out to teachers in areas other than computer science. The goal was to experiment with computational thinking in various courses and disciplines, and the strong support for professional development at the Lab Schools provided a unique environment for this work.

3. CURRICULAR MODIFICATIONS

In total six courses were altered for this project: middle-school and high-school computer science, and high-school Latin, graphic arts, English, and history. The changes in the computer science and Latin classes were made in 2010-2011 academic year as part of the RET supplemental grant. In the 2011-2012 academic year the project was expanded to include high school graphic arts, English, and history classes.

Our work on integrating computational thinking into the curriculum at the Lab Schools involved several distinct approaches. As an extension of the project at DePaul, we investigated the ways that computational thinking could be used to teach discipline-specific topics outside of computing, specifically history, graphic arts, English, and Latin. We were also interested in how computational thinking could be used in an interdisciplinary setting earlier in the curriculum, which was the focus of the work done in the middle-school computer science course. We also considered the effectiveness of several pedagogical approaches to computational thinking in a high-school computer science course.

3.1 Computational Thinking in High School

Four high-school courses were modified to include computational thinking approaches to teaching discipline-specific topics.

3.1.1 English

Students in high-school English classes often make generalizations about authors and their work. Sophomores who have read a single play by Shakespeare have no issues making major claims about his entire oeuvre. Erroneous generalizations can be corrected directly by the teacher or can be handled by asking students to support them, which they typically cannot. But while this approach may encourage students to think seriously about offering support and evidence for their arguments, it may also discourage them from thinking broadly about the texts they read. To test intuitions that deal with large number of texts, for example the 37 plays written by Shakespeare, one needs an efficient way to deal with a large amount of information. The widespread availability of digital texts and free online tools for analyzing them makes it relatively easy to generate data about texts. Even for intuitions that focus on only one text, these tools can still be efficient and illuminating.

The implementation of the project began in the Winter 2012 as a part of the ten-week term spent studying *Romeo and Juliet*. In addition to having read *Macbeth* in the ninth grade, students read several sonnets by Shakespeare. The central focus of the unit is a close reading of the text based on an examination of word choice, imagery, and syntax. The work for the project involves three to four class days of work as well as some related homework. The

unit is divided into three parts: introducing the approach, using the approach on *Macbeth* and *Romeo and Juliet*, and finally generating and testing intuitions independently. The teacher models the process of testing intuitions using text analysis as follows:

1. Form or identify intuitions
2. Hypothesize results – what results would suggest/confirm your claims?
3. Apply the tool(s), for example TAPor (http://portal.tapor.ca) and TagCrowd (http://tagcrowd.com)
4. Analyze results and assess the intuition
5. Form conclusions

This process involves computational thinking in every step. To form an intuition students must abstract ideas from the text to design an intuition and then construct results that would confirm their claims. Abstraction is a part of the design category in the Great Principles of Computing [4]. Finally, the connection between the intuition and the hypothesis must be evaluated. Using the tool(s) involves computation, albeit indirectly. Analyzing the results is a form of evaluation, and drawing conclusions involves both evaluation and abstraction.

When modeling intuitions, there are two types of thoughts that can be considered. The first are intuitions that could be answered through a traditional reading of the text but for which textual analysis might provide support, such as "Macbeth is the most violent character in the play". The second are claims that must be supported by evidence drawn from a number of texts, such as "Shakespeare's plays are violent." For both types of intuitions, the instructor goes through the five steps given above, so that students could see that while the tools produce results quickly, it is necessary to spend time reviewing the results to see if they produce useful evidence for the claims.

After the introduction of the textual analysis process, students work in small groups to use a tool to answer a question that relates to the class's work on *Romeo and Juliet* or the sonnets. Each group is assigned to use one of the tools demonstrated, including programs that allow users to examine collocation and word frequency. Students follow the process, write a report describing their findings, present that report to the class, and offer an assessment of the tool's capabilities and drawbacks. Finally, later in the term, students are asked to form intuitions independently and to explain how they would apply one of the text analysis tools to test that hypothesis.

3.1.2 History

Effective note taking is essential for every student, but it is usually assumed and rarely verified. In a subject like history where understanding and classifying information during a lecture is important this is particularly problematic. Efficacious note taking is not a passive activity, since a good set of notes should be an accurate record not just of what the teacher said, but also of the broader discussion including student comments and questions, and is a multi-stage process. The goals of the activities introduced into the year-long required American history course taken by the 11th and 12th grade students are to use computational thinking to help students:

- Improve their note-taking ability
- Improve their active-listening skills
- Recognize the difference between homework and studying
- Improve their recall skills

The activities ask students to use hashtags to improve their understanding of the process of note taking. The activities involve abstraction (to identify appropriate hashtags), evaluation (to consider if an effective set of hashtags has been chosen and applied correctly), and recollection (because the purpose of the hashtags is to allow classification and recall of information). The activities are sequenced as follows:

Step 1: Give a formal presentation on the definition and value of efficacious note taking. Preface the presentation with a discussion of note taking and affirm student observations whenever possible.

Step 2: Introduce the idea of hashtagging. This activity will help students to understand that hashtags are useful for note taking because it allows someone to choose the term or phrase that will best help them retain and retrieve the information they have recorded. It is important to convey that effective hastags are based on a thoughtful abstraction of the material and will not be proper nouns or words from the lecture. Assign as homework a short reading on hashtagging, to be discussed during lecture the next day.

Step 3: Present a short, formal lecture of approximately twenty minutes in length, during which students need to take class notes. The instructor should identify in advance up to ten key ideas that students might plausibly tag. At the end of the lecture students will be asked to quietly read over their lecture notes, adding or correcting information as necessary and supplying a maximum of ten tags for key ideas in their notes. Finally students will, in small groups, compare, contrast, and debate their hash tag choices and then present a summary of their work to the group for consideration.

Step 4: Present a longer, formal history lecture of about thirty minutes in duration for which students need to take notes. As with the last step, the teacher will have identified up to ten likely ideas that the students might plausibly tag. This time students will correct and augment their notes as homework, again identifying a maximum of ten hashtags in the margins of their notes. The ten tags will also be recorded on a separate piece of paper to be submitted to the teacher. In the following class their work will be reviewed in small groups and then with the larger class.

This process, and variations of it, will be repeated with regularity throughout the year. It will be applied not only to lectures but also to large group discussions and other classroom activities. Later a similar process will be used to improve students' critical reading skills.

3.1.3 Latin

This part of the project was designed to implement three particular exercises for learning Latin that require and encourage computational thinking: grammar notation, sentence diagramming, and metaphrasing. Notation involves identifying the grammatical parts of sentences as well as their relationship to the rest of the sentence, maintaining the integrity of the Latin syntax or sentence structure. Notating primarily uses abstraction. Diagramming requires a spatial realignment of sentence parts according to a precise visual diagram of English syntax. When

one diagrams a Latin sentence, one moves the sentence out of its natural order and into English word order, a process that involves evaluation in addition to abstraction. Finally, metaphrasing involves analyzing each component of the Latin sentence in its given word order, establishing its various possible uses and meanings, and trying to predict into which of three general Latin clause patterns it falls. This process also involves evaluation. Notating, diagramming, and metaphrasing were used in the course prior to the project, but typically on an ad-hoc basis in response to the needs of a particular class or student, or as a short-term activity designed to illuminate a difficult grammatical or syntactical concept. The project inspired the systematic use of the methods, with the intent that the students would learn the associated computational thinking skills.

The implementation of the new activities was in a class of twenty-two students with a broad range of ability and achievement. The teacher intended to introduce the activities as a part of a layered curriculum, or differentiated instruction, so that each student could find success with at least one of the practices. For the study data was gathered from weekly reading comprehension exercises and self-assessments or surveys administered every four weeks. The project began in the winter with notation exercises. The stronger students quickly moved onto diagramming and metaphrasing. But the other students had extreme difficulty with the notation as soon as the English sentences became more complex, providing some evidence that abstraction is crucial and challenging to teach in more than just computer science classes [7]. Those students then moved to notating Latin sentences, but it seemed best to keep those students working in Latin so the diagramming was skipped for the whole class. By the spring the whole class was either notating Latin sentences or metaphrasing them. Throughout the project data collection proved difficult. No usable data could be obtained from the monthly survey, as only seven students consistently completed them. On the reading comprehension quizzes, only two students seemed to perform more solidly and seemed to be making mistakes in reading rather than from simply guessing. For the rest, comprehension did not seem to increase or decrease notably.

There were unanticipated difficulties in the project, mainly the late start and the reduced amount of time devoted to it in the class. The teacher also hadn't anticipated how much notating or diagramming Latin sentences would confuse some students. In the end, only a handful of the students seemed to have acquired the particular skill set (notation, diagramming, metaphrasing) for reading Latin, and these were either the high achieving students, or in two cases, the students who struggle with language study. Moving forward, this work has helped to more clearly define the parameters for continued and regular computational thinking activities in the Latin classroom which the teacher is hopeful will produce more consistent results. Diagramming sentences will be avoided in the future since it forces English word order on Latin sentences and undermines the goals of the activities and class. In the 2011-2012 academic year notation or metaphrasing will be made a regular part of exercises, activities, and quizzes, with the hope that it becomes automatic when the students read Latin prose or poetry.

3.1.4 Graphic Arts

The fine arts course modified for the project is Digital Design for Communication, a year-long class in graphic design that is taken by students in grades 9 through 12. The activities added to the course have the students design a prototype for an object to be printed using a 3D printer off site, as well as design packaging for the item. The duration of the computational thinking activities is approximately one month, with the following objectives:

- To explore the nature of a designer and client relationship
- To learn about the concept of a prototype
- To experience what it is like to be a part of a design team where individuals are responsible for different aspects of product design and development, but where no single individual controls all phases of a project
- To learn how to use a simple 3D modeling software system such as Sketchup
- To learn about package design

The primary computational thinking focus of the activities is abstraction, with the various stages of the project designed to help students understand how to do graphic reduction and how and why to leave out details in that process. A second computational thinking concept that will be revealed is communication and how difficult it can be to specify information well enough to avoid ambiguity. The included activities are done in stages:

Stage 1: Introduce students to the concept of a prototype. Ask them to sketch out ideas for a small object that they would like to make. Each student should produce at least five different ideas, but these should be simple sketches rather than rendered images. Next a group critique of the sketches is conducted. The students are asked to pick one of their prototype sketches that they would like to develop further. This choice must be fixed at this stage.

Stage 2: Introduce Google's Sketchup software. First students will be given some in-class sessions where they play with the software but are not yet allowed to use it to design one of their prototypes.

Stage 3: Ask students to trade their hand-drawn sketches with each other and use the software to design each other's prototype. There should be no communication between the person who made the sketch and the person who reproduces it with the software. Once the prototypes are designed they will be sent off site to be printed on a 3D printer.

Stage 4: Introduce students to the concept of package design, including logo design, pamphlet design, and hands-on experimentation with paper folding to create packages. During this stage the difficulties of communication will be discussed in the abstract by playing fax machine, a telephone-like game that involves passing drawings around the group rather than whispering messages.

Stage 5: Bring students together to compare the results of the sketches to the printed 3D objects. The student who drew the original sketch and the student who designed it will come together to compare notes. These pairs of students will then describe and explain their prototype to another pair of students, with the second pair responsible for designing the packaging for that prototype. Communication between prototype teams and packaging teams is not only allowed but also considered very important.

A last step is to critique the final projects. The focus here is on emphasizing how the students felt about the trading back and forth that was designed into the activities.

3.2 Interdisciplinary Computational Thinking

The project modified in the middle-school curriculum is a creative-writing assignment with an implementation in Logo to create an interactive, text-based adventure game. Like other similar projects [11, 18], the goal is to allow middle school students to learn some programming in a more creative context. The modified curriculum is part of a fifth-grade class required of all students. The first part of the class relevant for the project focuses on graphics and design and the second on programming in Logo using MicroWorlds. For the latter, students work through a series of assignments, beginning with learning to control the turtle's movements one command at a time and then writing new procedures for the turtle. It culminates in two major assignments: a Pac-Man style maze game and a choose-your-own-adventure story with interactive text.

For the project the choose-your-own-adventure story assignment was modified. The middle-school teacher collaborated with all of the fifth-grade homeroom teachers during the 2010-2011 academic year to write an interdisciplinary curriculum unit based on the project. The curriculum required students to write their stories in the language arts class with the homeroom teacher and then bring the stories to the computer science classroom, where they learned how to use Logo to turn their stories into an interactive, text-based game. The collaboration between the homeroom teachers and the computer science teacher was novel and inspired by the project.

The goal for the students is to apply computational thinking to the task of writing a story. Because the choose-your-own-adventure story is necessarily segmented, students need to think about writing in a new and different way. The story is non-linear, and keeping track of the various threads of the story requires them to abstract portions of the plot. The students were encouraged in particular to use graphs to visualize and organize their plot lines. Graphs assist students in seeing that they have sufficient material for an interesting story and also how various pieces of the story can be reused in different plot lines. For example, a story involving three choices at each branch and allowing two plot progressions but containing no duplicated elements will result in 1 + 3 + 9 + 27 = 40 story segments. Allowing reuse of story lines can dramatically reduce the writing while not hindering the story.

When the change to the class was implemented during the 2010-2011 academic year, some challenges were identified. Some students struggled conceptually with creating a coherent, non-linear story. The project is also not very portable, since many students do not have the MicroWorlds software at home. A goal for the future is to find a way to publish or present their finished work outside of that environment. It was also difficult to identify ways to formally assess the project's goals and outcomes. However, there is a wealth of anecdotal evidence that the project was successful. Student engagement was improved. The students produced longer, and more complicated stories than students had in the past. There were more student questions about complex ways of traversing the story paths, and more students completed their stories as planned than in previous years. The students also seemed to enjoy the projects more, and many students chose to revisit the project during free-choice class time. Another success of the project was the collaboration between the teachers. Other fifth grade teachers have contacted the participating co-author, asking about possible collaborations on interdisciplinary projects.

3.3 Different Pedagogical Approaches

The portion of the study designed by the high school computer science teacher considered the efficacy of two different instructional approaches to teaching computer programming. The modified course provides instruction in Python to 9th grade students in a required computer science course. For most of the students in the class it was their first programming experience. The students were taught some basics of programming in Python during two weekly 45-minute class sessions for approximately two months. Prior to the programming unit students had done some CS Unplugged activities [2], learned about the history of computing, and created some web pages with HTML and CSS. The goal in teaching the basics of Python was to make students familiar with how computational solutions to problems are implemented in a real language.

Students in four sections of the course were split into two groups, roughly even in number and gender. The two groups produced the same programs and output but were instructed differently. The first group used a worked examples approach [12]. This group received a lecture on a topic, where the instructor programmed examples in front of them, a process known as live programming. The teacher would pose a problem orally and ask the group through discussion and brainstorming to suggest a solution. The teacher would demonstrate how to program that solution in Python and illuminate any nuances of the language that arose when solving the problem. At the end of the lecture, the teacher handed out a packet containing lightly annotated copies of each of the examples that the class had generated during the discussion. In the packet were a series of problems that the students were asked to solve. The required solutions ranged from near verbatim copying of the examples seen during the lecture to problems that required more abstract thinking and application of the programming concepts to new situations. In this group the students were instructed, verbally and on the assignment sheet, to study the worked examples in the packet provided and use them as a guide to solve the problems.

The second group used a modified examples approach. Instead of a lecture and demonstration, these students receive a work packet that guided them through the problem making a series of modifications to some starting code that ultimately led them through the same series of problems and solutions that the lecture-and-worked-examples group received. This group was able to work at their own pace, following the packet and getting a much more hands-on approach to developing the solutions to the example problems.

The programming skills of the students were measured using a commercial online tutoring system, and the data is still being analyzed. Anecdotally there was not much difference between the two groups in terms of student performance, ability, confidence or stress. It is likely that the largest contributing factor to student performance is the small and infrequent amount of contact time with the students. Because giving homework was not possible, there was a lot of momentum lost between each instructional period and the students' opportunities for practice were not contiguous enough to make a difference.

Interestingly, the number of students in a given section seemed to be a larger determining factor in which instructional method worked over another. In larger sections the self-paced modified examples packet seemed to work better. Students were encouraged to help each other and in the larger sections they

seemed more willing to do so. The teacher was also able give attention to those students who asked for it or were clearly in need of it. In smaller sections the modified examples were not received well, as the students seemed reluctant to work together. In the smaller groups the lecture and worked example approach seemed better because the interactive demonstration felt more like a conversation or brainstorming session in which each student was able to make a contribution.

4. DISCUSSION

The work done at the Lab Schools has demonstrated how computational thinking can be integrated into middle- and high-school courses. Examples of computational thinking in non-computer-science disciplines are particularly important for progress in this area [1]. The project has also identified a number of challenges in implementing computational thinking activities into the 5-12 curriculum, including student participation, differences in student ability, and the difficulty in assessing the effectiveness of the new activities. Recognition that failed attempts are nevertheless beneficial to the effort is something that has both been identified by the project participants and in the wider literature [1].

One of the clear successes of this project has been the way in which it has impacted the participating teachers. This project has impacted the participants' teaching in courses beyond the one selected for modification. Like many professional development opportunities, it has provided a new lens through which to view existing courses, and the changes from the project are expected to filter through the curriculum. Fostering such professional development activities has been recognized as a crucial step in bringing computational thinking to the K-12 curriculum [1, 19].

Another benefit to come from the project has been the student response. Across a variety of classes, the instructors observed that student engagement and enthusiasm were improved during the project. The activities were also seen by teachers as a way to address heterogeneous student populations, which is a possible explanation for increased student engagement. Investigating the motivational and pedagogical impact of computational thinking activities and assignments in disciplines outside of computing is a fruitful area for future work.

5. ACKNOWLEDGMENTS

Funding for this project was provided by the National Science Foundation under Grant Number 0829671. We are grateful to Ljubomir Perkovi½ for his instrumental role in the project. We thank Randy Connolly for feedback on earlier drafts of the article.

6. REFERENCES

[1] Barr, V. and Stephenson, C. 2011. Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? In *ACM Inroads*, 2:1, pp. 48 – 54.

[2] Bell, T., Witten, I., and Fellows, M. Computer Science Unplugged, www.csunplugged.org, accessed December 2011.

[3] Cooper, S., Pérez, L.C., and Rainey, D. 2010. K-12 Computational Learning. In *Communications of the ACM*, 53:11, pp. 27 – 29.

[4] Denning, P. 2003. Great Principles of Computing. Communications of the ACM, 46:11, pp. 15-20.

[5] Dierbach, C. et al. 2011. A Model for Piloting Pathways for Computational Thinking in a General Education Curriculum. In *Proceedings of the 42nd SIGCSE Technical Symposium on Computer Science Education*, Dallas, Texas.

[6] Fox, E. et al. 2008. LIKES (Living in the KnowlEdge Society). In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, Portland, Oregon.

[7] Kramer, J. 2007. Is Abstraction the Key to Computing? *Communications of the ACM*, 50:4, pp. 37-42.

[8] Martin, F. et al. 2009. Joining Computing and the Arts at a Mid-Size University. *Journal of Computing Sciences in Colleges*, 24:6.

[9] Perkovi½ L. and Settle, A. 2010. Computational Thinking across the Curriculum: A Conceptual Framework. *College of Computing and Digital Media Technical Report 10-001*, DePaul University.

[10] Perkovi½ L., Settle, A., Hwang, S., and Jones, J. 2010. A Framework for Computational Thinking across the Curriculum. In *ITiCSE 2010: The 15th Annual Conference on Innovation and Technology in Computer Science Education*, Ankara, Turkey.

[11] Repenning, A., Webb, D., and Ioannidou, A. 2010. Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools. In *Proceedings of the 41st SIGCSE Technical Symposium on Computer Science Education*, Milwaukee, Wisconsin.

[12] Trafton, J.G. and Reiser, B.J. 1993. The contributions of studying examples and solving problems to skill acquisition. Proceedings of the 15th Conference of the Cognitive Science Society, pp. 1017-1022.

[13] The University of Chicago Lab Schools, http://www.ucls.uchicago.edu/, accessed December 2011.

[14] Wallace, S.A., Bryant, R., Orr, G. 2009. The Northwest Distributed Computer Science Department. In *Journal of Computing Sciences in Colleges*, 25:1.

[15] Way, T. et al. 2010. A Distributed Expertise Model for Teaching Computing Across Disciplines and Institutions. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering*, Las Vegas, Nevada.

[16] Wing, J. 2006. Computational Thinking. In *Communications of the ACM*, 49:3, pp. 33-35.

[17] Wing, J. 2011. Research Notebook: Computational Thinking – What and Why? http://link.cs.cmu.edu/article.php?a=600, accessed December 2011.

[18] Werner, L., Denner, J., Bliesner, M., and Rex, P. 2009. Can Middle-Schoolers use Storytelling Alice to Make Games? Results of a Pilot Study. In the *Proceedings of ICFDG: International Conference on Foundations of Digital Games*, Orlando, Florida.

[19] Wolz, U. et al. 2011. Computational Thinking and Expository Writing in the Middle School. In *ACM Transactions on Computing Education*, 11:2.