Improving Learning of Computational Thinking Using Creative Thinking Exercises in CS-1 Computer Science Courses

L. Dee Miller, Leen-Kiat Soh, Vlad Chiriacescu
Department of Computer Science and Engineering
University of Nebraska, Lincoln
{lmille, lksoh, vchiriac}@cse.unl.edu

Elizabeth Ingraham, Duane F. Shell, Stephen Ramsay, Melissa Patterson Hazley
Center for Computational Creativity
University of Nebraska, Lincoln
{eingraham2, dshell2, sramsay2}@unl.edu,
mpatterson.hazley@gmail.com

***Abstract*—Promoting computational thinking is one of the top priorities in CS education as well as in other STEM and non-STEM disciplines. Our innovative NSF-funded IC2Think project blends computational thinking with creative thinking so that students leverage their creative thinking skills to "unlock" their understanding of computational thinking. In Fall 2012, we deployed creative exercises designed to engage Epstein's creative competencies (Surrounding, Capturing, Challenging and Broadening) in introductory level CS courses targeting four different groups (CS, engineering, combined CS/physical sciences, and humanities majors). Students combined hands-on problem solving with guided analysis and reflection to connect their creative activities to CS topics such as conditionals and arrays and to real-world CS applications. Evaluation results (approximately 150 students) found that creative thinking exercise completion had a linear "dosage" effect. As students completed more exercises [0/1 - 4], they increased their long-term retention [a computational thinking test], $F(3, 98) = 4.76$, $p = .004$, partial $Eta^2 = .127$ and course grades, $F(3, 109) = 4.32$, $p = .006$, partial $Eta^2 = .106$. These findings support our belief that the addition of creative thinking exercises to CSCE courses improves the learning of computational knowledge and skills.**

Keywords—CS1, Creative Thinking, Computational Thinking, College, Computer Science

I. INTRODUCTION

Computational thinking is an approach to solving problems, building systems, and understanding human behavior that draws on the power and limits of computing [1]. Computational thinking includes skills such as conceptualizing at multiple levels of abstraction, defining and clarifying a problem by breaking it down into relational components, and testing and retesting plausible solutions. Already considered to be one of the top priorities in computer science (CS) education, computational thinking is becoming more important in both STEM and non-STEM disciplines given the reliance of these disciplines on computational techniques for data collection, archiving, processing, and analysis. In this way, computational thinking has become an *enabler* that promotes problem solving in a wide range of disciplines and a *bridge* that allows for interdisciplinary innovation and discovery. It has even been argued that computational thinking is as fundamental as skills such as reading, writing and arithmetic [2].

The increased demand for computational thinking has led to numerous articles in educational research venues such as the proceedings of FIE and the communications of the ACM. These articles demonstrate the increasing momentum of research addressing the need for effective education in computational thinking, both for CS and across the broader spectrum of STEM and non-STEM disciplines. This research is diverse, ranging from course specifications to course development, from community building to setting policies, and from teaching and learning to assessment [2][3][4][5][6][7].

However, there is a potentially serious problem for expanding computational thinking from CS into STEM and non-STEM disciplines. The students in non-CS disciplines come from diverse backgrounds and many are likely to ***have limited understanding of computing concepts*** that are used as the basis for computational thinking. Without a basic level of understanding, these students will have a very difficult time developing computational thinking fluency. As noted by Epstein et al. [8] and Shell et al. [9], one cannot think beyond the boundaries of one's cognitive tools or knowledge. Students who are forced to use computational thinking based on concepts that they do not understand may become frustrated, potentially lowering both their motivation and their self-efficacy. As both student motivation and self-efficacy are strongly connected to student learning [10][11], if learning computational thinking leads to lowered student motivation and self-efficacy, then ***it could actually be detrimental—reducing learning for students in STEM and non-STEM disciplines alike—instead of being beneficial***.

Our innovative solution is to improve the learning of computational thinking by blending it with creative thinking. Creative thinking is thinking patterned in a way that tends to lead to creative results [12]. Creative thinking is not limited to the arts but is an integral component of human intelligence that can be practiced, encouraged and developed within any context [13][14][15]. Epstein's Generativity Theory breaks creative thinking down to four core competencies: *capturing* novelty,

challenging established thinking and behavior patterns, *broadening* one's knowledge beyond one's discipline, and *surrounding* oneself with new social and environmental stimuli [8]. By blending computational and creative thinking students can leverage their creative thinking skills to "unlock" their understanding of computational thinking [9]. As an example, rather than trying to directly explain multiple levels of abstraction in computer programming, we ask students to think about why a corporation is organized using layers. Students are likely to reach the conclusion that having executives who are experts on a single layer is the most efficient and effective way to identify and solve problems. Thus, by using creative thinking skills students have unlocked why programs use multiple layers of abstraction. In this way, we make computational thinking more generally applicable to STEM and non-STEM disciplines where students may have creative thinking skills but lack understanding of computing concepts. The reverse is also true: students who understand computational thinking could leverage it to improve their creative thinking skills.

We have designed four creative thinking exercises that blend computational and creative thinking. These exercises are designed to provide instruction on CS concepts such as conditionals, arrays, modular programming, and debugging. However, these exercises completely lack any programming code. Instead, these exercises involve tasks seemingly unrelated to CS such as individually writing the chapters for a story based on separate plot points and then working in a group to revise the story to make the content more cohesive. These exercises are designed to foster development of Epstein's creative competencies by engaging multiple senses, requiring imaginative thought, presenting challenging problems and combining both individual and group efforts. During these exercises, students are given explanations, which we call CS Light Bulbs, which relate the tasks directly to the CS topics. Finally, the students answer analysis and reflection questions designed to further promote both computational thinking and creative application of computational skills.

We deployed these creative thinking exercises during the Fall 2012 semester at the University of Nebraska, Lincoln. Over 200 students in four different introductory CS courses took the exercises and the exercises counted as part of their final course grades. Each course was tailored to a different target group (CS majors, engineering majors, combined CS/physical sciences majors, and humanities majors) and so these courses contained a mix of students with both computational and creative thinking skills. Our evaluation results found that after completing the exercises creative competency was significantly correlated with computational thinking as measured by a computational thinking test. Further, completion of the exercises improved both long-term retention of course content and course grades—even when student motivation was factored in. Overall, these results support the effectiveness of blending computational and creative thinking.

II. CREATIVE THINKING EXERCISE DESIGN

All our exercises combine hands-on problem solving with written analysis and reflection and are designed to be completed by a group of students working collaboratively in order to facilitate creative thinking from students with multiple backgrounds. Each exercise requires approximately 1-2 hours per student but students are given two weeks to work on the exercises because of the collaboration required. The exercises have four common components (Objectives, Tasks, CS Light Bulbs, and Questions) described in more detail below. Exercises are designed so that the students have hands-on and group tasks first, in Week 1, and then reflect on their Week 1 activities in Week 2 by answering questions. Both Week 1 and Week 2 are graded.

Objectives. The Objectives component, at the beginning of the exercise handout, lists the computational and creative thinking objectives for the exercise. These objectives help the student understand why they are doing these exercises by showing how an exercise lacking any programming code is related to CS concepts and how students can use relevant creative thinking skills (described in terms of Epstein's core competencies of Surrounding, Capturing, Challenging and Broadening) to solve the exercises.

Tasks. The majority of the content of an exercise handout is the Tasks component which lists all the tasks the group of students must complete during the two weeks of the exercise. These tasks require students to work collaboratively (and may require individual contributions as well). Tasks are designed to engage group members with different backgrounds, to encourage creative approaches and to promote open-ended discussions as the group produces concrete, specific artifacts.

CS Light Bulbs. The CS Light Bulbs are self-contained explanations which make explicit the connections between the exercise tasks and a set of CS topics. By being embedded in the exercise handout within the Tasks, the Light Bulbs help students stay aware of the underlying computational thinking skills while they are performing the creative thinking tasks.

Questions. The Questions component, at the end of the exercise handout, uses open-ended questions that require collaboration among students as they engage in both analysis and reflection. To answer these questions, students must apply *creative thinking* to CS problems and revise the results of their original tasks using *computational thinking*. Questions thus build upon the CS Light Bulbs and reinforce the connections between the creative thinking fostered by the tasks and the computational thinking in the CS topics.

III. INDIVIDUAL EXERCISES

Here we describe each of the creative thinking exercises in more detail. We developed four different exercises for the Fall 2012 deployment: (1) Everyday Object, (2) Storytelling, (3) Cipher, and (4) Exploring. These exercises, each of which has the set of components described above, are summarized below, but due to space considerations we provide just one example for each exercise of the multiple CS Light Bulbs. Furthermore, we highlight Everyday Object and Storytelling exercises and only summarize the other two.

A. Everyday Object Exercise

Objectives. ***Computational***: (1) Learning about the description and design process for modular programming by describing an everyday object in detail including why the object is needed and how the object functions; (2) Learning

about abstraction and function characterization by identifying properties of an everyday object

Creative: (1) Surrounding: looking at an everyday object in new ways, using all of your senses to understand how it's made and how it functions; (2) Capturing: using written language to describe all the different details and characteristics of this everyday object so you can work with it in new ways; (3) Challenging: describing the operations of an everyday object with words and also as a computer program; (4) Broadening: imagining that this everyday object doesn't exist and acting like its inventor who is trying to fulfill a need by creating something new and useful

Tasks. For the next two weeks, you will be using language to try to clearly and thoroughly describe the functions of an ordinary object that you might use every day. You will be acting like the inventor of that object, imagining that it does not yet exist and trying to describe what need would be fulfilled by your (new) object and how (specifically) it will function.

Your group will choose a common, everyday object from the list. Your challenge is to imagine that this object does not exist and to describe in written language (1) the mechanical function of your object, (2) what need is fulfilled by this object, and (3) the physical attributes and characteristics of your object.

You must describe the object's function, the need it will fulfill and its attributes in clear, non-technical language which any user could understand. Your description must be specific enough so that someone who had never seen the object could recognize it and understand how it works and understand what benefits it provides.

Note: Students were given a list of objects (zipper, mechanical pencil, binder clip, Ziploc bag, scissors, tape measure, stapler, nail clippers, umbrella, flashlight, can opener, clothespin, sticky notes, toilet paper holder, revolving door) from which to choose.

CS Light Bulbs. This description process is very important for developing algorithms in computer science. An algorithm consists of the series of steps necessary to solve a given problem. By using algorithms, we can solve problems without having to constantly "reinvent the wheel" and spend the time, money, etc. to figure out each step ourselves. However, if one or more of these steps are unclear, we can have difficulty following the algorithm which can lead to serious repercussions as described in the following two examples. First, if the formulation algorithm used to mix the concrete for a road or bridge is unclear, workers may make a mistake during pouring leading to reduced service life. Second, if the business plan algorithm for a new company is confusing, venture capitalists may be reluctant to invest leading to failure of the business. To avoid these repercussions, the developer should make every effort to make the algorithm's description as clear as possible for all steps. In other words, characterization of processes is key; it allows us to abstract a process and then convert it into a formal problem or solution. *Note: The handout also includes three other Light Bulbs on writing functions in CS, the diagramming process, and abstraction in programming languages.*

Questions. ***Analysis***: (1) Consider your object as a computer program. Draw a diagram that shows all its functions as boxes (name them), and for each function, its inputs and outputs. Are there shared inputs and outputs among the functions? (2) Consider the list of physical attributes and characteristics. Organize these such that each is declared as a variable with its proper type. Can some of these attributes/characteristics be arranged into a hierarchy of related attributes/characteristics?

Reflection: (1) Consider your response to Analysis 1, are there functions that can be combined so that the object can be represented with a more concise program? Are there new functions that should be introduced to better describe your object such that the functions are more modular? (2) Have you heard of abstraction? How does abstraction in computer science relate to the process of identifying the functions and characteristics as you have done in this exercise?

B. Storytelling Exercise

Objectives. ***Computational***: (1) Learning about how large programs are developed using modular programming by separately writing chapters for a story which are connected with fixed story points at the beginning and end of each chapter; (2) Learning about the need for the debugging and testing process by revising the chapters to make the flow of the story more logical and cohesive; (3) Learning about the need for a methodical and logical debugging process to make sure that a program's observed output matches the expected output; (4) Learning about identifying logical inconsistencies in a product or solution and using methodical approaches to resolve them

Creative: (1) Surrounding: using your senses of sight, sound, smell, touch and your imagination to connect two seemingly unrelated things (two story points) in a logical and coherent way; (2) Capturing: learning to take novel and spontaneous outputs (your two story points) and use language to fill in the blanks by writing your way from one point to another; (3) Challenging: applying computational debugging to a story by taking independently generated chapters and editing them so they form a consistent narrative; (4) Broadening: increasing your ability to problem-solve and to collaborate by taking the inputs of others (your group's individual chapters) and making them into an effective and functional whole—a logical and cohesive story.

Note: The following is an example of the story points to be developed as chapters: (1) "OPEN UP! THIS IS THE POLICE!", (2) Panting, Kevin ran down the unfamiliar alley, hoping there would be an exit ahead, (3) The pendant wasn't particularly remarkable, but something drew his eye to it…something he couldn't quite put into words, and (4) "It doesn't respond like that for everybody," the green-eyed girl said. "You're the first person it has activated for in over 25 years."

Tasks. For the next two weeks, your group will be telling a story with several short chapters. However, you won't write the story as a linear line from A to B. Rather, you will be given a series of story points. These story points will be the pivotal moments in the story. They will act as inputs and outputs to your individual chapters. Each person in your group will write

a chapter of the story that falls between two story points. You will all write your chapters independently of each other. All you know is how the story (and each chapter) starts and how it ends up; the input and the output.

During Week 1, group members will separately write the chapters. You will not consult with your group members and you are not allowed to share your chapter with your group members. You will shape your chapter solely on the story points.

During Week 2, review the chapters which have been written by your group members. Most likely not all of your chapters will make sense in the context of each other (this is normal and expected). After you review all of your group's chapters, you need to reconcile the differences. Pick one chapter as the anchor and make minimal changes to it. Then revise the other chapters so that they logically fit with this anchor chapter.

CS Light Bulbs. Writing a chapter based on story points is similar to how we write functions in computer science. To make the design of large programs feasible, the inputs and outputs for a function are generally known before that function is written. In a sense, the way the code for the function is written is less important than whether it produces the correct output value for a given input value. Of course, the function must mesh seamlessly with all the other functions in the program. Additionally, the function must work for all possible input values not just one or two. For example, a function which controls airbag deployment must work for all manner of collisions not just one at 30 mph. Meanwhile, a program solution is often made up of a sequence of functions, where the output of the first function is fed into the second function as input, and the second function's output is fed into the third function, and so forth.

Note: The handout also includes two other Light Bulbs on the debugging and testing process and finding logical errors in programs.

Questions. ***Analysis***: (1) What was the most difficult part of "debugging" your story? Did entire chapters need to be rewritten? Or could you manage to reconcile between each chapter with each other by making only minor changes? (2) What would you have changed about your initial story telling process to make the debugging process easier? Would you have made your story more straight forward and logical, or more ridiculous and expansive? Essentially, how would you go about writing the story so that it includes the fewest number of "bugs"?

Reflection: (1) Compare this process to working in a large team on a software project. In what ways are the two processes similar? In what ways are they different? (2) How would you change the rules of the assignment to guarantee that a minimal number of "bugs" are created? Assume that all chapters must still be written simultaneously.

C. Cipher Exercise

For the Cipher exercise, student teams are required to come up with three ciphering rules to "code" several questions during Week 1. In Week 2, all teams will try to decipher these coded questions by answering the questions in the same code. Computational objectives include (1) Learning about the code generation process for ciphers and encryption programs by creating rules and a one-to-one mapping to encode messages and (2) Learning about the trial-and-error code breaking process by trying to guess the mapping to read coded messages; while creative objectives include (1) Capturing: creating new outputs and using new ways to represent and save data by inventing, testing and documenting a simple system of rules to transform English sentences into a cipher so that their meaning is obscured; and (2) Broadening: acquiring new information and skills by understanding how just three simple rules can generate complex products (ciphers) and how your everyday experience with the English language or with the world can inform your code-generating and code-cracking efforts

In the Questions component, the analysis questions include: (1) Analyze the questions asked by other teams. Were they too easy to answer? Were the questions faulty? (2) How easy would it be to implement your cipher in a computer program? What would your concerns be with implementing your cipher? Reflection questions include: (1) How strong are your rules? Are they clear, or is there some level of ambiguity? Could anybody follow them and understand them? How could you improve your rules? (2) Reflect on your "code cracking" approach. What kind of weaknesses and strengths did it have? Imagine you had to decode over 9000 characters rather than just the 26 in the English alphabet. Could you scale your approach? Would brute force work (i.e. simply try to determine a mapping of characters to their cipher)?

D. Exploring

In this exercise, each group is required to pick a location on campus from one of ten locations on a campus map. Each member of the group must make several observations at the group's location, as well as taking a photo of him- or herself making observations at this location. They are also required to use a data collection form to record their results and to verify that they have gathered sufficient data.

Computational Objectives include: (1) Learning about the data collection process used in many fields by making observations about a specific location on campus; (2) Learning why the ability to brainstorm and visualize the "big picture" is important for writing programs; (3) Learning why the data collection process is important for evaluating existing products and programs; and (4) Learning about capturing and representing different aspects and patterns of an environment for solving a problem

Creative Objectives include: (1) Surrounding: looking at a familiar location on campus in a new way and using all of your senses (sight, sound, smell, touch and thought) to become more aware of what is around you; (2) Capturing: Making a written inventory of everything you observe at a site—plants, animals, people, sounds, and smells—and also documenting your observing by taking a photo of yourself at the site; (3) Broadening: Increasing your ability to solve problems by looking at tables of data in new ways. What did you expect to observe at the site? What did you actually observe? What does the data document about the site and your experience at the site? What does it leave out?

In the Questions component, the analysis questions include: (1) What would be the benefits and drawbacks of using arrays, cell arrays, or structures to store the data you have collected? (2) After completing the previous analysis, you must now analyze the applications of the data you collected. To do this, each team member must brainstorm at least three different real world applications of the data—"Be bold, think big, do not constrain yourself to feasibility or practicality!" Reflection questions include: (1) What other data (that you might ordinarily ignore) could be collected that could be manipulated programmatically and exposed to solve some problem? (Think along the lines of Google scanning and analyzing millions of books to observe long term cultural and language changes.); (2) Speculate on how you might broaden the classes of data, and on how you might capture overlooked data.

IV. Exercise Deployment

The exercises were deployed using the Written Agora system [16]. This is a wiki system designed to facilitate online collaboration between groups of students. The wiki system includes a content page where students can work together on completing the tasks and an online forum where students can discuss, with group members, the responses to the analysis and reflection questions. As the wiki was always online, students could log in and work on the exercises whenever it was convenient. The wiki also kept track of all the revisions so that we could determine which students were contributing to the group.

The exercises represented 3-5% of the final grades depending on the course. After completing the tasks and answering the questions, students in each group were assigned individual grades based on their contributions to the group's wiki page.

V. Evaluation Methods

Students voluntarily participated in evaluation data collection which was approved by the UNL Institutional Review Board. The courses had 241 students initially enrolled and 196 students who completed the courses. Of those who completed the courses, 150 students (133 male; 17 female) consented to participation in the evaluation and 129 students (114 male; 15 female) consented to the use of their course grades and university grade point average. Not all students completed all measures. Sample for specific analyses are shown in Table 1.

Course grades were used to determine impact on student achievement in the course. Retention of core computational thinking knowledge and skills was assessed by a test developed by CSCE faculty [17]. The computational thinking knowledge test contained 13 conceptual and problem-solving questions for the core computational thinking content common to all CS-1 classes. The coefficient alpha reliability estimate was .76. The computational thinking test was administered on a Web platform (Survey Monkey) during the last week of classes as part of evaluation data collection. Students reactions to the exercises were obtained in two ways: (1) after each exercise, 3-6 volunteer students participated in a focus group; (2) using instruments adapted from Shell, Snow, and Claes [18] students were asked to rate exercises and provide open-ended comments in surveys done as part of evaluation data collection. Students' overall GPAs were obtained from university records and were used as a covariate for analysis.

VI. Results

A. Impacts on Student Learning

We used Analysis of Covariance (ANCOVA) to test whether the number of exercises completed (0-1, 2, 3, 4) was associated with higher course grades and computational thinking test scores. We included students' cumulative Grade Point Average (GPA) adjusted to remove the grade for the course as a covariate to control for differences that might be attributable to students' general level of academic ability.

Cumulative GPA was a significant covariate with course grades ($F(1, 109) = 56.26$, $p < .0001$, partial $Eta^2 = .340$) confirming that students' achievement in the class generally reflected their overall achievement record. With GPA controlled, the number of exercises completed was significantly associated with course grade (Table I: $F(3, 109) = 4.32$, $p = .006$, partial $Eta^2 = .106$). There was a significant linear trend ($p = .0001$) from 2 to 4 exercises completed.

TABLE I. Mean Scores By Exercise Completion

	Computational Thinking Test			***Course Grade[a]***		
Exercises Completed	***M***	***SD***	***N***	***M***	***SD***	***N***
0-1	4.31	3.09	13	2.48	1.38	14
2	6.91	3.20	23	2.33	1.16	26
3	7.41	3.22	29	3.24	1.17	33
4	8.21	2.26	38	3.53	0.59	42

[a]Grade=12-point scale from 0=F to 4.0=A.

For the computational thinking test, cumulative GPA was not a significant covariate ($F(1, 98) = .296$, $p = .588$). This suggests that long-term retention of course content is not necessarily the same as course achievement reflected in grades. This distinction was also confirmed by the relatively moderate correlation of $r = .35$ between course grade and the computational thinking test. The number of exercises completed was significantly associated with computational thinking test score (Table I: $F(3, 98) = 4.76$, $p = .004$, partial $Eta^2 = .127$). There was a significant linear trend ($p < .0001$) from 0-1 to 4 exercises completed.

These findings indicate an apparent "dosage" effect, i.e., learning and course achievement increase with each additional creative thinking exercise completed. The increases are not trivial. Students completing 2 or fewer exercises averaged approximately a C to C+; whereas, students completing 3 exercises averaged a B and those completing 4 averaged a B+. Similarly, students improved on the knowledge test by about 2 1/2 points from 0-1 to 2 exercises completed, another 1/2 points from 2 to 3, and over 3/4 points from 3 to 4. Results for the knowledge test did not differ for CS majors and non-CS majors. Non-CS majors; however, appeared to benefit more in relation to grades with an increase from 1.67 for 0-1 exercises, to 2.33 for 2 exercises, to approximately 3.5 for 3 and 4 exercises. CS majors did not exhibit as sharp of increase but those completing all 4 exercises achieved a 1/2 grade point higher

score than any other group. In relation to traditional findings for classroom educational interventions, these are strong effects. They also demonstrate meaningful "real world" impact. Of course, we cannot conclusively determine that the results are solely due to the exercises; however, by controlling for students' cumulative GPA, we can say that the results are not due to students who are better students in general completing more exercises than poor students.

B. Student Reactions to the Creative Thinking Exercises

One focus group was conducted for each of the four exercises with each done in one of the four participating classes. A majority of students reported that they did not enjoy completing the exercises and found it difficult to understand how they related to course objectives. These comments were mirrored in open-ended survey responses. When asked general questions in focus groups and in the survey about impacts on computational thinking and computing, most students indicated little impact expressing that exercises were unrelated to the concepts covered in the class. Most reported that the exercises did not have enough payoff in terms of grades or learning to justify the amount of time they took. Students expressed difficulty communicating with classmates outside of class which impacted their willingness and ability to complete assignments. Students felt that instructor facilitation of the exercises was too limited and they found it problematic that little class time was devoted to discussion of the exercises.

When asked to rate exercise effectiveness, about 75% of students indicated that they thought the exercises were either not effective or *neither* effective *nor* not effective in preparing them to use computational and CS tools in their field, helping them problem solve more creatively, and helping them creatively apply computational thinking in their field. These ratings of effectiveness were not significantly correlated with the number of exercises completed suggesting that perception of effectiveness was not a motivating factor for doing the exercises. The usefulness and interestingness of the story telling and cipher exercise were rated somewhat higher and received more positive responses in focus groups.

This dissatisfaction and self-assessment of impact is in stark contrast to the significant 'dosage' effect identified for exercise completion on grades and computational knowledge. However, when probed in focus groups for exercise impacts on specific computational thinking and computing skills, many students indicated that these were improved by completing the exercises. Furthermore, students stated that the exercises were successful in helping them understand how programming relates to the world around them and how they might apply concepts in a practical setting. These findings suggest that students are not necessarily able to recognize the value of educational activities for enhancing their learning, especially in their first or general impressions of an activity. But, if prompted to reflect further, they may begin to recognize the value.

VII. Lessons Learned and Next Steps

Completing the exercises appears to have benefited student achievement and learning. Both their long-term retention (as measured by a computational thinking test) and their course grades improved the more exercises they completed. It appears that the use of creative thinking skills does help students "unlock" their understanding of computational thinking as we proposed. Students did not always recognize these benefits, however. They generally expressed dissatisfaction with the exercises and thought they were unrelated to the course and did not help their learning of course content. Only when pushed for specific impacts did students begin to recognize positive impacts of the exercises on their understanding and problem solving. To explain, the creative thinking exercises involve a level of abstraction that may mask the connections between the exercise activities and the underlying computational thinking principles. As a result, students may not be able to see these connections, leading them to question the relevance and benefits of doing the exercises.

This student feedback demonstrated the need for us to better and more specifically explain how creative thinking helps CS students and how the creative exercises relate to specific CS topics. This feedback also made clear the need to improve the logistics of exercise deployment. To address these student concerns and to make the benefits of the exercises more visible to students, we have taken the following steps for future deployment: (1) continually revised the objectives in all the exercises to make their connections to CS topics more explicit, (2) designed an "Exercise 0" so students can practice using the system before the graded exercises are assigned, (3) bundled the exercises together with learning objects on those same topics [19] to provide more detail on CS topics, and (4) assigned students to groups so that there were fewer delays in forming groups and so that students were on an equal footing in groups.

We discovered that the logistics of exercise deployment—how large the class was, how often it met, whether students took other courses together, whether they interacted outside of class—had a greater impact than we anticipated on how smoothly the exercises ran and how satisfied the students were with their experience. Students were especially sensitive to difficulties working with other students in their groups. Accordingly, we are in the process of identifying how to combine real-time experience (such as in-class explanations of the exercises) with these improvements to the exercises and with improvements to the online wiki in order to minimize student frustration with the logistics of the exercises. Given the evidence of real benefits to the students from our blending of computational and creative thinking, it is now our creative challenge to improve our exercise design and delivery so that students are eager to complete the exercises and recognize their value in enhancing their learning of computational thinking.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant no. 1122956 and in part by the UNL Phase II Pathways to Interdisciplinary Research Centers grant.

References

[1] J. Wing, "Computational Thinking," in Communications of the ACM, vol. 49, 2006, pp. 33-35.

[2] J. Wing, "Computational Thinking: What and Why?" in Link Magazine, 2010.

[3] P. J. Denning, "Computing is a Natural Science," in Communications of the ACM, vol. 49, 2007, pp. 33-35.

[4] M. Guzdial, "Paving the Way for Computational Thinking," in Communications of the ACM, vol. 51, 2008, pp. 25-27.

[5] P. J. Denning, "The Profession of IT beyond Computational Thinking," in Communications of the ACM, vol. 52, 2009, pp. 28-30.

[6] G. H. L. Fletcher, "Human Computing Skills: Rethinking the K-12 Experience," in Communications of the ACM, vol. 52, 2009, pp. 23-25.

[7] C. Lewis, M. H. Jackson, and W. M. Waite, "Student and Faculty Attitudes and Beliefs about Computer Science," in Communications of the ACM, vol. 53, 2010, pp. 78-85.

[8] R. Epstein, S. Schmidt, and R. Warfel, "Measuring and Training Creativity Competencies: Validation of a New Test," in Creativity Research Journal, vol. 20, 2008.

[9] D. F. Shell, D. W. Brooks, G. Trainin, K. Wilson, D. F. Kauffman, and L. Herr, The Unified Learning Model: How Motivational, Cognitive, And Neurobiological Sciences Inform Best Teaching Practices. Netherlands: Springer, 2010.

[10] S. Bergin, R. Reilly, and D. Traynor, "Examining the Role of Self-Regulated Learning on Introductory Programming Performance," in ICER, 2005, pp. 81-86.

[11] C. Sorge, "What happens: Relationship of Age and Gender with Science Attitudes from Elementary to Middle School," in Science Educator, vol. 16, 2007, pp. 33-37.

[12] D. N. Perkins, "Creativity by Design," Educational Leadership, vol. 42, 1984, pp. 18-25.

[13] U. Kraft, "Unleashing Creativity," Scientific American Mind, vol. 16, 2005, pp. 19-21.

[14] M. Michalko, "Cracking Creativity," Ten Speed Press, 2001.

[15] T. Tharp, "The Creative Habit: Learn it and Use it for Life," Simon & Schuster, 2005.

[16] A. Eck, L-K. Soh, and C. Brassil, "Supporting active wiki-based collaboration," (to appear) in Tenth International Conference on Computer Supported Collaborative Learning, 2013.

[17] D. F. Shell and L-K. Soh, "Profiles of motivated self-regulation in college computer science courses: Differences in major versus required non-major courses," in Journal of Science Education and Technology, 2013.

[18] D. F. Shell, G. R. Snow, D. R. Claes, "The Cosmic Ray Observatory Project (CROP): Results of a summer high-school student, teacher, university scientist partnership using a capstone research experience," in Journal of Science Education and Technology, vol. 20, 2011, pp. 161-177.

[19] L. D. Miller, L-K. Soh, A. Samal, and G. Nugent, "iLOG: a Framework for Automatic Annotation of Learning Objects with Emperical Usage Metadata" in IJAIED, vol. 21, 2011, pp. 215-236.