Improving Students' Learning and Achievement in CS Classrooms through Computational Creativity Exercises that Integrate Computational and Creative Thinking

Duane F. Shell

University of Nebraska-Lincoln
114 Teachers College Hall
Lincoln, Nebraska 68588-0345
1-402-472-6981
dshell2@unl.edu

Abraham E. Flanigan

Markeya S. Peteranetz
University of Nebraska-Lincoln
114 Teachers College Hall
Lincoln, Nebraska 68588-0345
1-402-472-8331

abrahamflanigan@gmail.com,
markeya.dubbs@huskers.unl.edu

Leen-Kiat Soh

University of Nebraska-Lincoln
122E Avery Hall
Lincoln, Nebraska 68588
1-402-472-6738
lksoh@cse.unl.edu

Elizabeth Ingraham

University of Nebraska-Lincoln
220 Richards Hall
Lincoln, Nebraska 68588-0345
1-402-730-6596
eingraham2@unl.edu

ABSTRACT

Our research is based on an innovative approach that integrates computational thinking and creative thinking in computer science courses to improve student learning and performance. Referencing Epstein's Generativity Theory, we designed and deployed Computational Creativity Exercises (CCEs) with linkages to concepts in computer science and computational thinking. Prior studies with earlier versions of the CCEs in CS1 courses found that completing more CCEs led to higher grades and increased learning of computational thinking principles. In this study, we extended the examination of CCEs to by deploying revised CCEs across two lower division (freshmen, sophomore) and three upper division (junior, senior) CS courses. We found a linear "dosage effect" of increasingly higher grades and computational thinking/CS knowledge test scores with completion of each additional CCE. This dosage effect was consistent across lower and upper division courses. Findings supported our contention that the merger of computational and creative thinking can be realized in computational creativity exercises that can be implemented and lead to increased student learning across courses from freshmen to senior level. The effect of the CCEs on learning was independent of student general academic achievement and individual student motivation. If students do the CCEs, they appear to benefit, whether or not they are self-aware of the benefit or personally motivated to do them. Issues in implementation are discussed.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SIGCSE '17, March 8–11, 2017, Seattle, WA, USA.

© 2017 ACM. ISBN 978-1-4503-4698-6/17/03...\$15.00.

DOI: <http://dx.doi.org/10.1145/3017680.3017718>

CCS Concepts

Social and professional topics→Computer science education

General Terms

Performance, Design, Experimentation, Human Factors

Keywords

Computational thinking; Creative thinking; CS course achievement

1. INTRODUCTION

A primary goal of computer science (CS) education is to produce students who are capable of applying and practicing computational thinking and applying it effectively to problem solving situations. Computational thinking includes skills such as problem decomposition, pattern recognition, abstraction and generalization, algorithm design, and evaluation that are useful in both STEM and non-STEM disciplines. Indeed, computational thinking is becoming increasingly important in a variety of fields because of the reliance of these disciplines on computational techniques for data collection, archiving, processing, and analysis, and it has even been conceptualized as a skill as fundamental as reading, writing and arithmetic [5, 13].

Our project focuses on an innovative approach [6, 7, 9, 12] by which we aim to improve the learning of computational thinking by blending it with creative thinking. We view creative thinking through the lens of Epstein's Generativity Theory [3, 4], which identifies four core competencies of creative thinking. First, *Capturing* novelty refers to the capacity to recognize and note unique ideas as they occur. Second, *Challenging* established thinking and behavior patterns refers to the capacity to generate new approaches to problems. Third, *Broadening* one's knowledge beyond one's discipline allows for the novel integration of ideas. And finally,

Surrounding oneself with new social and environmental stimuli can lead to new experiences and ideas. Our premise is that by blending computational and creative thinking, students can leverage their creative thinking skills to “unlock” their understanding of computational thinking [12]. In this way, we should be able to also make computational thinking more generally applicable and attractive to STEM and non-STEM disciplines.

In our framework, both computational thinking and creative thinking are viewed as cognitive tools that expand the knowledge and skills that one can apply to a problem [10]. The blending of computational thinking with creative thinking is not conceived as a set of dichotomies, but rather as complementary or symbiotic abilities and approaches. Computational tools expand the knowledge and skills that one has available thereby broadening the array of knowledge that one may bring to a problem. For example, *challenging* forces computational tools to be used in unanticipated and unusual ways leading to the development of new computational approaches to both old and new problems, *surrounding* creates new ways of looking at problems and attention to different stimuli and perspectives that may be relevant to how a problem is approached computationally, and *capturing* encourages consideration of new ways to represent and save data and solution procedures.

2. EXERCISES

The principles underlying the design of our Computational Creativity Exercises (CCEs) are (1) balancing of attributes between computational and creative thinking and (2) mapping between computational and creative concepts and skills as they manifest in different disciplines. For each exercise, we have a set of creative objectives, computational objectives, and collaborative problem solving objectives. As noted, our creative objectives are based on Epstein’s four competencies: capturing, challenging, broadening, and surrounding.

For the set of computational objectives, we include two aspects: the actual CS concepts—such as sorting and logical conditionals—as well as computational thinking skills. The computational thinking skills that we use are: problem decomposition, pattern recognition, abstraction, generalization, algorithmic design, and evaluation.

2.1 Overall Designs

As reported in [6, 7, 11, 12], for all our CCEs, students work in groups to engage in a combination of hands-on problem solving, written analysis, and reflection. The collaborative nature of the CCEs facilitates creative thinking by bringing together students from diverse backgrounds. Although each CCE requires approximately 1-2 hours of work per student, groups are given two weeks to complete the exercises due to the amount of collaboration required. The exercise handouts have four common components: (1) Objectives, (2) Tasks, (3) CS Light Bulbs, and (4) Questions. Each exercise is supported using an online collaborative wiki platform called the Written Agora [2].

In terms of **Objectives**, each exercise handout begins with a list of all the computational and creative thinking objectives—including concepts such as problem decomposition, pattern recognition, abstraction, generalization, algorithm design, and evaluation, and other fundamental CS concepts—for that exercise. These objectives (1) help students understand the utility of the exercises by showing how an exercise lacking any programming code is related

to CS concepts and (2) indicate how students can use relevant creative thinking skills to complete the exercises.

The **Task** component is the largest component of an exercise handout and lists all the tasks the group must complete during the two weeks of the exercise. These tasks require collaboration among the students and some require individual contributions. 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.

The **CS Light Bulbs** are text snippets embedded within the Tasks section of the exercise handout that explicitly connects the exercise tasks and CS topics. The embedded CS Light Bulbs help students maintain awareness of the underlying computational thinking skills as they perform the creative thinking tasks. For different classes, we have designed relevant CS Light Bulbs. For example, for our Pathfinding CCE, our CS Light Bulbs include looping, boundary conditions, and relative coordinates. But, for the course in Automata, we included a new Light Bulb on context-free grammars.

The **Questions** component presents open-ended analysis and reflection questions that require collaboration among students. Students respond to the questions during Week 2. These questions require students to apply *creative thinking* to CS problems and to evaluate 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.

2.2 Exercises

Table 1 shows the suite of CCEs that we have developed. Each exercise has been deployed, evaluated, and revised. Some have multiple versions to meet the needs of different courses.

As an example, here we look at an abridged description of the Marble Maze exercise.

In terms of **Objectives**, we have *Computational*: (1) Learning how to collaborate on complex programs by designing the marble maze as a team and separately building the maze segments; (2) Learning how to utilize the restrictions of an abstract programming interface (API) by developing maze segments using a fixed list of materials; (3) Learning about the need to design functions (or modules) with clear inputs and outputs by making sure that the segments fit together; and (4) Developing testing and debugging skills when the marble gets stuck and cannot move on to the next segment. We have *Creative*: (1) Surrounding: using your senses of sight, touch and sound to help you design and construct a maze for a marble; (2) Capturing: documenting the height and length of a marble maze, calculating its path and measuring its performance; (3) Challenging: using limited materials and devising new methods of fastening to build a segment of a marble maze through which a marble can travel without stopping; and (4) Broadening: acquiring new skills in testing and collaborating by connecting individual segments into a functioning maze which meets operational parameters.

For **Tasks**, using only the specified materials, each team is required to make a maze through which a marble will travel. The completed maze must have an intentional path, along which a marble can travel without stopping, and with a clear beginning

Table 1. Computational Creativity Exercises Developed

Name	Brief Description
Every-day Object	Identify an “everyday” object (such as nail clipper, a paper clip, Scotch tape) and describe the object in terms of its inputs, outputs and functionalities.
Cipher	Devise a three-step encoding scheme to transfer the alphabet letters into digits and encode questions for other teams to compete to decode.
Story Telling	Develop a chapter (100-200 words) individually and independently in week 1 and work as a team in week 2 to resolve all conflicts or inconsistencies.
Exploring	Explore sensory stimuli at a particular site (sounds, sights, smell, etc.) and document observations.
Simile	Poses “simile” descriptions and participate in team-to-team Q&As to solicit guesses and descriptions relevant to a particular object.
Machine Testing	Devise ways to test a black-box mysterious machine without causing harm to humans while attempting to reveal the functionalities of the machine.
Calendar	Build a calendar for a planet with two suns, four different cultural groups with different resource constraints and industrial needs.
Path Finding I	Create a step-by-step instruction on drawing lines to create a quilt pattern on a $n \times n$ grid and identify similar structures in other teams’ quilt patterns.
Path Finding II	Use rotation, reflection, and loop to generate a more complex quilt pattern based on simpler base pattern.
Marble Maze I	Each team member creates a sub-structure allowing a marble to travel at least for n seconds in week 1 and the team puts all sub-structures together to make a super-structure in week 2.
Marble Maze II	Teams are broken up and now must adapt their own sub-structure to work with other sub-structures in their new teams.
Marble Maze III	All teams bring together their super-structures and build a mega-structure.

and ending point. This path can include loops. Each team may start the marble’s journey through the maze by dropping it. The marble may not travel on the floor more than 4 inches. The marble must travel from a minimum height of 12 inches and must travel a minimum of 4 linear feet (in any configuration) and must reach the end of the maze without stopping. Any pathway for the marble must be elevated at least 4” off the floor. The marble should take at least 4 seconds to complete its travel from beginning to end. The team building the maze with the longest marble travel time will receive bonus points.

In Part 1, each team member will construct a 12” (minimum) segment of maze which a marble can travel along or through without stopping. In Part 2, each team integrates all segments built by its team members into a single, final maze structure. The specified materials allowed for use in the construction of a maze segment are: (1) any paper or cardboard such that *no piece may be larger than 4” in its longest dimension*; (2) paper clips such that they *must be transformed* (cut, bent, etc.) from their default shape; (3) string; and (4) a marble (as supplied by the instructor). Glue, tape or staples may *not* be used.

Furthermore, In Part 2 where team members need to combine the separate segments they have constructed and tested into a maze structure, the connection methods must meet the following two requirements: (1) the connections should allow the marble to traverse the entire maze on an uninterrupted path without stopping, and (2) the connection methods must be *reversible*, meaning that it is possible to separate the single maze back into segments. Team members will likely need to resolve problems that occur in the connections between different segments. These connections may even require a partial redesign to one or more segments. After testing, each team should take a video of the completed maze in operation and upload that video to YouTube. And bonus points will be awarded to the team whose maze has the longest travel time.

In terms of **Questions** for the exercise, we have the following *analysis* questions:

- (1) How did your testing go for both your individual unit and the integrated maze? What changes did you have to make after integration testing to fix individual units?
- (2) What do you think is key to the success or failure of your design? Be specific about what aspects worked well and what aspects need work.

And, we also have the following *reflection* questions:

- (1) How did the group distribute its work of making and combining the maze modules? Did your planning work well? What would you do differently?
- (2) Did your group have to redesign segments in order to meet the testing requirements? If yes, identify the reasons for why the initial designs failed; if no, identify the reasons why the initial designs succeeded. What considerations were (or should have been) considered during the initial designs in order to meet the testing requirements?

We provide four **Light Bulbs**: collaboration, constraint satisfaction, modularity and testing and debugging. Here we provide two example light bulbs. For the *modularity* light bulb, we refer to the starting and end points of each segment, and how the endpoint of a segment needs to find a compatible starting point of another segment when integrating the segments together into a maze structure:

*“The use of clearly defined beginning and ending points for **modules** is very important in computer science. These beginning and ending points work as an instruction manual for an existing module so that a programmer knows what to expect from the module. In this way, a programmer can utilize an existing module without having to understand every line of source code in that module. This is critically important since the source code is often unavailable and even when it is, understanding it line-by-line would take a prohibitively large amount of time and money. (Imagine reading all 1.7 million lines of code in the F-22 Raptor jet fighter.) However, a set of ambiguous instructions, such as using variables x and y , is nearly as useless as no instructions at all. On the other hand, for real-world abstract programming interfaces, programmers do not have the space to provide exhaustive instructions on how every module works. Instead, programmers must think creatively about how to succinctly describe the beginning and ending points for their modules.”*

For the testing and debugging light bulb, we have:

*“Developing **testing and debugging skills** is vitally important in computer science. Real-world software applications are massive in scope and may involve millions of lines of code. These applications are so large it is inconceivable for a single programmer to have read all the lines of code and so large groups of programmers must work together collaboratively. In such an environment, mistakes are bound to be made when programmers work with unfamiliar code. Furthermore, a complex application may change constantly during development with new features added and old features modified or removed. Obviously, programmers cannot go back through the source code line-by-line after every feature change to make sure that all potential bugs have been removed. Instead, the programmers (and quality assurance staff) are constantly testing and debugging use cases for the application until (and often after) it ships. The quality of these use cases and thus, the number of bugs removed and quality of the final application, depends directly on the creative thinking skills of those testing and debugging the application. Note also that testing and debugging are not limited to software engineering or computer science. These are skills important in various disciplines involving, for example, evaluation and refinement of a process or methodology, a product design or an experiment setup.”*

Note that we also have Marble Maze II and Marble Maze III exercises in our suite. Marble Maze II extends Marble Maze by swapping out one team member with another team’s, together with the segment of his or her creation. Each team is required to compensate for the loss of one segment from their existing working maze structure while integrating the new segment into their maze structure. Marble Maze III extends Marble Maze II where all teams work together to build a super maze structure, putting into use abstraction—understanding the roles of different segments (such as speeding up the marble, slowing down the marble, preventing the marble from jumping the track, etc.) and generalization—leveraging “integration” tricks that worked for a 4-segment maze towards a many-segment maze.

3. PRIOR STUDIES

As alluded to earlier in the paper, prior studies have examined earlier versions of the CCEs in CS1 courses at a large, public Midwestern University. In the very first study [6], students in four different introductory CS1 courses, tailored to different student populations, did the CCEs and the exercises counted as part of their final course grades. CCEs were done as extra-credit and completion of CCEs was voluntary. A linear “dosage” effect of higher grades and increased learning of core computational thinking principles with increasing CCE completion from 0-1 to 4 CCEs was found in all courses [6].

Subsequent follow-up studies examined differences between CS majors and non-CS majors, freshmen and upper class students, and men and women students [7]. Results showed that for the knowledge test, the linear “dosage effect” of increasing knowledge test scores with student completion of each additional exercise was consistent across all comparisons. For grades, however, the effects were more nuanced, with CS majors having a consistent linear increase for each exercise completed, whereas non-majors had grade increases only with completing at least three exercises. Also, upper class students had increases with completing at least three exercises, whereas freshmen students

needed to complete all four exercises to see an increase. There were no differences between female and male students.

A final study used a quasi-experimental design to compare CCE implementation in a CS1 course tailored for engineers during Spring 2013 to a control semester of no implementation in the same course in Fall 2013. Results indicated that students in the CCE implementation semester had significant higher knowledge test scores than students in the control semester [12].

4. CURRENT STUDY

This paper reports on a Fall 201 comprehensive study that significantly extends the prior studies previously reported. While prior studies examined *only* CS1 level courses, this study examined whether similar effects would be found for additional lower and upper division courses. The courses examined were (1) 100 level: CS1: Engineering and Science Focus (3 CCEs), (2) 200 level: Computer Organization (4 CCEs), (3) 300 level: Data Structures and Algorithms (3 CCEs), (4) 300 level: Programming Language Concepts (3 CCEs), (5) 400 level: Automata (2 CCEs).

Based on student feedback from the prior implementations, CCE implementation was changed to make them more integrated into the class. CCE light bulbs were tailored by course instructors and project faculty for the specific course in which they were implemented. Instructors were expected to include the CCEs as regular course assignments and grade them accordingly. CCEs typically represented 3-5% of the final grades depending on the course.

For our analysis, we adopted the approach used in prior studies [6, 7] and analyzed whether there was a linear increase in course grades and scores on a computational thinking/CS knowledge test for each additional CCE completed. This is because not all students completed all exercises.

Based on the prior findings reported in [6, 7], our research hypothesis was:

There would be a linear effect of CCE completion on both course grades and the computational thinking knowledge test.

Because of the broader array of lower and upper division implementation courses, our secondary research question was:

How would CCE completion effects differ for courses that were lower division (freshmen-sophomore) versus upper division (junior-senior)?

5. DEPLOYMENT AND PLATFORM

The exercises were deployed using the Written Agora system [2], 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 their responses to the analysis and reflection questions with group members. 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. 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.

6. METHODS

Students voluntarily participated in study data collection approved by the participating large, public Midwestern University’s Institutional Review Board. The courses altogether had 405 students

initially enrolled and 370 students who completed the courses. Of those who completed the courses, 271 students (231 male, 40 female; 13 freshmen, 127 sophomore, 66 junior, 42 senior, 23 other) consented to participation in the evaluation and use of their course grades and university grade point average. Of these students, 17 did not have a course grade (Incomplete; Pass-NoPass) and six had invalid student ID's. The final analysis sample was 248 students (209 male, 39 female; 13 freshmen, 117 sophomore, 61 junior, 39 senior; 18 other). Demographics for sub-group analyses (course level, class standing, CS major/non-major) and the sample N values for specific analyses (which vary due to missing data) are shown in Tables in the Results section.

Course grades were used to determine the impact on student achievement in the course. To standardize grades across courses, grades were converted to Z -scores within each course. To provide a common cross-course measure of core computational thinking knowledge and skills, a computational thinking knowledge test developed by CSCE faculty and used in prior studies was used [6, 7, 11]. The test contained 13 conceptual and problem-solving questions for the core computational thinking content generally taught initially in CS1 classes. The coefficient alpha reliability estimate was .81 for the sample in this study. The computational thinking knowledge test was administered using Survey Monkey® during the last week of classes as part of broader evaluation data collection. Students' cumulative Grade Point Averages (GPA) were obtained from university records and adjusted to remove the students' course grade.

We used Analysis of Covariance (ANCOVA) to test whether the number of exercises completed was associated with higher course grades and computational thinking knowledge test scores.

7. RESULTS

We included students' cumulative GPA as a covariate in all analyses to *statistically control for differences that might be attributable to students' general level of academic ability*. Students' cumulative GPA was a significant covariate for course grades in all analyses. Students with higher GPAs earned higher grades indicating that their achievement in the class generally reflected their overall academic achievement. Students' cumulative GPA also was a significant covariate for the knowledge test in all analyses, indicating that general academic achievement was also related to learning of core computational thinking knowledge and skill. Because GPA was controlled, any effect of exercise completion was independent from the student's GPA.

CCE Completion. As shown in Table 2, the number of CCEs completed was significantly associated with course grade ($F(4, 239) = 3.34, p = .011$, partial $\eta^2 = .053$). ANOVA planned polynomial contrasts found a significant linear trend ($p = .005$) from 0 to 4 exercises completed.

For the knowledge test (Table 2), The number of exercises completed also was significantly associated with computational thinking test score ($F(4, 153) = 2.49, p = .045$, partial $\eta^2 = .061$) with ANOVA planned polynomial contrasts indicating a significant linear trend ($p = .012$) from 0 to 4 exercises completed. These findings replicated the "dosage" effect found in previous studies of the CCEs in CS1 course implementations [6, 7]. *For each additional CCE completed, course grades and the knowledge test score increased.*

Table 2. Mean Scores by CCE Completion

CCEs Completed	Course Grade			Knowledge Test		
	M	SD	N	M	SD	N
0	-.634	1.07	66	6.00	3.66	31
1	-.138	1.00	43	6.92	3.91	24
2	.175	.97	48	7.89	3.15	37
3	.355	.70	60	7.59	3.45	46
4	.634	.29	28	9.90	2.93	21

Course Level. Courses differed in the number of CCEs implemented from two to four. To account for this in examining the differences between lower and upper division courses, the number of CCEs completed was re-coded into three levels: 0, 1, and 2-or-more. As shown in Table 3, the number of exercises completed was significantly associated with course grade ($F(2, 238) = 5.49, p = .005$, partial $\eta^2 = .044$), with ANOVA planned polynomial contrasts indicating a significant linear trend ($p = .002$) from 0 to 2-or-more exercises completed. The course level by CCE interaction was not significant ($F(2, 238) = 2.37, p = .096$, partial $\eta^2 = .020$) indicating that the linear trend for CCE completion did not differ for lower and upper division courses.

For the computational thinking knowledge test (Table 3), the number of exercises completed was not significantly associated with knowledge test scores in the overall model ($F(2, 152) = 2.36, p = .098$, partial $\eta^2 = .030$); however, ANOVA planned polynomial contrasts indicated a significant linear trend ($p = .034$) from 0 CCEs completed to 2-or-more CCEs completed. The course level by CCE interaction was not significant ($F(2, 152) = 0.19, p = .827$, partial $\eta^2 = .002$), indicating that the linear trend for CCE completion did not differ for lower and upper division courses.

Table 3. Mean Scores by CCE Completion and Course Level

Exercises Completed	Lower Division [100-200 level]			Upper Division [300-400 level]		
	M	SD	N	M	SD	N
Course Grades						
0	-.705	1.05	44	-.493	1.12	22
1	.182	.70	21	-.445	1.16	22
2-or-more	.301	.85	96	.463	.49	40
Knowledge Test						
0	5.75	3.14	20	6.45	4.59	11
1	6.08	3.48	12	7.75	4.29	12
2-or-more	7.69	3.17	72	9.22	3.52	32

8. DISCUSSION

Our hypotheses that there would be a linear effect of CCE completion on both course grades and the computational thinking knowledge test were confirmed. As in prior studies [6, 7], there was a linear "dosage" effect for completion of more CCEs. The increases were *not* trivial. Students improved about a grade point for each addition CCE completed (C+ to B- to B to B+ to A from 0 to 4 completed CCEs). Similarly, students improved on the knowledge test by about one point for each additional CCE completed with those completing four CCEs scoring 50% higher than those not doing a CCE. When exercises were capped at two to control for differences in available CCEs across courses, results indicated that completion of the CCEs had similar linear "dosage" effects in both lower division and upper division course. *This is an important extension from prior findings* [6, 7, 11] that have only examined CCEs in introductory CS1 courses. Even though

the basic computational thinking principles at the core of the CCEs would be covered in CS1 courses, with minimal adaptations, the CCEs appear to be effective in courses at the sophomore to senior level across an array of CS topics.

9. CONCLUSIONS & FUTURE WORK

The findings support our central contention that the incorporation of Computational Creativity Exercises based on Epstein's [3, 4] creative competencies can improve learning of computational thinking and achievement in CS courses. We believe that the CCEs impact student achievement and learning because they make students deal with computational principles and skills abstracted from coding. This enhances their ability to connect the computational thinking knowledge to more diverse applications consistent with the Unified Learning Model (ULM) [10]. Also consistent with the ULM, completing CCEs provides more retrieval and repetition of course content, which strengthens knowledge connections ultimately resulting in better understanding of course content and more potential for creative application of computational thinking and CS.

The findings—from this and prior studies [6, 7, 11]—confirm that the merger of computational and creative thinking can be realized in exercises that can be implemented in CS courses across the entire curriculum from freshmen to senior level, and these exercises can help students improve their course achievement and learning of CS and computational thinking. The CCEs appear to have an *independent instructional effect*. In all studies, the “dosage effect” was distinct from general student achievement as students’ overall GPA was *controlled*.

As in prior administrations [6], student focus group and post-survey responses indicate that for the most part students did not see the CCEs as being useful or connected to course content. They typically had a negative perception of the CCEs. The effect of the CCEs, therefore, does not appear to depend on individual student motivation or engagement. *If students do the CCEs, they benefit, whether or not they are self-aware of the benefit.* Of course it is desirable that students are positively motivated toward the CCEs and see their value. The effect of student motivation is seen in the different numbers of students completing CCEs. When completion of CCEs is voluntary, many students will not be motivated to choose to do them. As a result, these students do not achieve as high of grades and do not learn as much.

Note that despite the attempts to integrate the CCEs into the course, in focus groups and open-ended evaluation survey questions at the end of the course, most students indicated that they did not see the CCEs as being highly associated with course material nor did they feel that the CCEs were part of the course. But, focus group responses also suggest that *when CCEs are more fully integrated into the class, students enjoyed them and appreciated the rationales behind the exercises.* Accomplishing this integration appears to require instructors to overtly connect the CCEs to the course by (1) discussing each exercise in class and going through the exercise assignments with student, (2) explicitly mapping activities in exercise to topics in class, and (3) relating both computational and creative objectives to real-world problems.

We are continuing to refine and improve the CCEs. We have begun to specify collaborative objectives for each CCE. Our collaborative objectives are based on our experience working in teams and coordinating student group work and are informed by

the recognition of the importance of collaboration as one of the essential skills for effective problem solving in the 21st century [1, 8]. We have developed a stand-alone computational creativity course that is anchored in the exercises. We are currently evaluating the impact of this course on student learning of CS and computational thinking. We also are implementing the CCEs in non-CS courses in Art and Music to determine if they can help non-CS students utilize computational thinking in their own discipline.

10. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation grants no. DUE-1122956 and DUE-1431874.

11. REFERENCES

- [1] American Management Association. 2016. The four skills that give you the advantage in today's business environment. <http://www.amanet.org/training/21st-Century-skills/>
- [2] Eck, A. Soh, L.-K., and Brassil, C. 2013. Supporting active wiki-based collaboration. In *Proc. of CSCL* (Madison, WI, June 15-19, 2013), 176-183.
- [3] Epstein, R. Generativity theory and creativity. In M. A. Runco & R. S. Albert, eds., *Theories of creativity* (Rev. ed.), Cresskill, NJ: Hampton Press, 2005, 116-140.
- [4] Epstein, R., Schmidt, S., Warfel, R. 2008. Measuring and Training Creativity Competencies: Validation of a New Test. *Creativity Research Journal*, 20, 7-12.
- [5] Guzdial, M. 2008. Paving the Way for Computational Thinking. *Comm. ACM*, 51, 25-27.
- [6] Miller, L.D., Soh, L.-K., Chiriacescu, V., Ingraham, E., Shell, D. F., and Hazley, M. P. 2013. Improving Learning of Computational Thinking using Creative Thinking Exercises in CS-1 Computer Science Courses. In *Proc. FIE* (Oklahoma City, OK, October 23-26, 2013), 1426-1432.
- [7] Miller, L.D., Soh, L.K., Chiriacescu, V., Ingraham, E., Shell, D.F., and Hazley, M.P. 2014. Integrating computational and creative thinking to improve learning and performance in CS1. In *Proc. SIGCSE* (Atlanta, GA, March 5-8, 2014), 475-480.
- [8] Resnick, M. 2014. Give P's a chance: Projects, peers, passion, play. In *Constructionism and Creativity: Proc. 3rd Int. Constructionism Conf.* (Vienna, Austria, August 19 - 23, 2014), 13-20.
- [9] Robinson, K. *Out of Our Minds: Learning to be Creative*. Capstone, New York, 2001.
- [10] Shell, D. F., Brooks, D. W., Trainin, G., Wilson, K., Kauffman, D. F., and Herr, L. *The Unified Learning Model: How Motivational, Cognitive, And Neurobiological Sciences Inform Best Teaching Practices*. Springer, Netherlands, 2010.
- [11] Shell, D. F., Hazley, M. P., Soh, L.-K., Miller, L. D., Chiriacescu, V. and Ingraham, E. 2014. Improving learning of computational thinking using computational creativity exercises in a college CS1 computer science course for engineers. In *Proc. FIE* (Madrid, Spain, October 22-25, 2014), 3029-3036.
- [12] Soh, L.-K., Shell, D. F., Ingraham, E., Ramsay, S. and Moore, B. 2015. Viewpoint: Improving Learning and Achievement in Introductory Computer Science through Computational Creativity, *Comm. ACM*, 58(8). 33-35.
- [13] Wing, J. Computational Thinking: What and Why?. *Link Magazine*. 2011. Retrieved from Carnegie Mellon University: <https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why>.