The Impact of STEM Experiences on Student Self-Efficacy in Computational Thinking and Computer Science

Abstract: Since the introduction of new curriculum standards at K-12 schools, computational thinking has become a major research area. Creating and delivering content to enhance these skills, as well as evaluation, remain open problems. This paper describes two different interventions based on the Scratch programming language which aim to improve student self-efficacy in computer science and computational thinking. The two interventions were applied at a STEM outreach program for 5th-9th grade students. Previous experience in STEM related activities and subjects, as well as student self-efficacy, were collected using a developed pre- and post-survey. We discuss the impact of our intervention on student performance and confidence, and evaluate the validity of our instrument.

Keywords: computational thinking, self-efficacy, computer science, design

Introduction

The Next Generation Science Standards (NGSS) (2013)[22] and Common Core Standards (CCSS) (2010)[19] have influenced a STEM movement with ever-increasing needs for Computational Thinking (CT). CT has been defined in a variety of ways, but discussion between researchers on what the definition of CT should include was born from Wing's vision to make CT a fundamental skill for everyone, not just computer scientists[32]. *Computational Thinking: A Digital Age Skill for Everyone* (2011)[2] emphasizes the importance of Wing's vision and notes the significance of CT as a vital 21st century skill, which is noted in the P21 Framework for 21st Century Learning[23]. Not all definitions of CT are created equal; however, among various definitions[2,3,20,29,32], abstraction and algorithms are two main concepts that everyone agrees upon. More so, Bar and Stephenson point out that whatever an educator's interpretation or definition of CT includes, it "must ultimately be coupled with examples that demonstrate how computational thinking can be incorporated in the classroom[3]". This leads to a conclusion that K-12 educators do understand the importance of CT skills; however, they lack a clear, practical definition with established pedagogy to help bring CT to their classrooms[2,3]. Solving this problem would also remove the preconceived notion that computational thinking is Computer Science (CS) or computer programming, and show educators that CT is a skill used across many disciplines[5,27,29,32].

Not only is CT needed for the K-12 student body, but also for existing and up and coming teaching professionals. This paper describes a summer STEM institute where the Manhattan-Ogden Unified School District 383 has partnered with the Department of Education at Kansas State University. The STEM institute is a month long outreach program designed to engage, excite, and teach 5th-9th grade students science, technology, engineering, and math through various classes and activities ranging from food science and agriculture, to computer science and robotics. During this institute, groups of pre-service teachers from the university are paired with experienced K-12 educators from the school district, or instructors/professors from the university, who run the different classes. This provides a practical scenario for education students to gain hands-on experience in a classroom setting, while also learning innovative ways to incorporate STEM into their future classrooms.

The focus of this paper covers two new interventions with similar pedagogy implemented for the institute which focuses on video game design and robotic agents. Each intervention used the visual programing language Scratch (2009)[26] as a tool in order to seed CT and CS concepts in both institute participants and pre-service teachers. We also describe a self-efficacy instrument used to measure STEM experiences, 21st century learning skills, and CT. The importance of this research is to discover whether or not past STEM activities and experiences will transfer to student self-efficacy in CT, as well as develop a method for delivering and measuring CT skills in the K-12 environment.

Background

Visual based programming tools have become largely popular due to their ease of use for beginner programmers in not only K-12, but also higher education. These block-based programming languages have made their way into many STEM outreach programs in order to train both students and educators. Code.org (2015)[10] has been a major leader advocating for CS in the K-12 classroom by providing materials for educators, as well as providing interactive tutorials on programming using Blockly[14]. Most recently, Code.org released two new programming tutorials themed around the popular game, Minecraft, and up and coming release of *Star Wars VII*. Outreach programs have a large range of focus. Scalable Game Design (SGD), for example, developed CT tools using AgentSheets and AgentCubes which enabled middle school students to develop video games[25]. The tools increased student understanding of CT concepts, which then allowed them to apply their new skills on scientific simulations, not just video games. More importantly, Repenning et al measured student's learning of CT as patterns rather than concepts. This led to an automatic analysis tool using latent semantics to determine student growth in CT. While still using block-based programming tools, another outreach program, GK12 INSIGHT, worked with K-12 teachers and graduate students, to incorporate embedded systems and sensor technology with emphasis on CT in K-12 curriculum[21]. Other researchers have focused on creating various outreach programs, such as CS4HS, that emphasize training teachers in computer science[8]. Another CS4HS workshop focused on the measurement of the ability of teachers to incorporate CT concepts into lesson plans[9]. While Bort & Brylow developed a rubric for general CT concepts (i.e. abstraction, algorithms, etc.), they did not measure the teacher's own understanding of CT. Further expanding, Bean et al developed a two-part self-efficacy survey for a pre-service teacher training program[4]. The first measured the pre-service teachers' confidence that they are capable of incorporating computer programming into their classroom, as well as recognizing how programming concepts relate to NGSS and CCSS. The second survey delved into their self-efficacy in their understanding of CT concepts in relation to programming. In another related project, Bell conducted a CT intervention as part of an art-based program component of a summer STEM institute; his experimental approach served as a basis for this work[5].

Methods

While our interventions vary in theme, both center on the same learning and scaffolding theory. One of the major challenges of teaching CT concepts through computer programming in both K-12 and higher education environments is that students quickly become overwhelmed with learning a new language. By starting with text-based languages, beginning programmers spend more time struggling with the syntactic structure of programming languages instead of learning the core concepts like algorithms, abstraction, and data analysis. By using Scratch, a block-based language, students can learn the language quickly. This allows us to focus more on teaching CT concepts, rather than giving drawn out instructions on how to use the programming language. This is especially important since, like most outreach programs and school districts, we have an extremely limited time to work with students about CT and CS.

By using a block-based programming language, we are effectively reducing the students' cognitive load. Cognitive Load Theory (CLT) (2010)[24] summarizes that an individual's ability to learn is compromised when the intake of a learning task exceeds their working memory capacity. Morrison et. al. developed sub goal labels in worked examples which reduced cognitive load in text-based programming[18]. We take a similar approach by using seed Scratch programs for our intervention activities. The seed programs are partially completed. This skeleton allows us to outline the structure of the program and complete low learning potential portions of the program (for example, have sprites, costumes, and backgrounds already created), which allows us to focus on any new CT skills or computer science principles in a limited timeframe while reducing cognitive load.

We continue to reduce the cognitive load, specifically extraneous load (2015)[18], by putting scaffolding in place for each activity or project in the interventions. Scaffolding is a support structure put in place for learners to accomplish tasks that they could otherwise not complete[7]. We take the approach of instructional scaffolding which correlates to programming tutorials. However, as Repenning notes, direct instruction can actually limit student motivation, especially in females[25]. We utilize Problem-based Learning (PBL) (2009)[28] alongside Inquiry Learning (IL) to keep students motivated. Kirschner argues that PBL and IL do not provide enough guidance for students to learn based on human cognition (2006)[17]; Hmelo-Silver refutes this statement by providing evidence that PBL and IL have enough scaffolding to be effective learning practices[16]. Our coding activities in day one use mostly direct instruction scaffolding (step-by-step instruction of what blocks to use), but as the class progresses into later projects, we move into using guided discovery or inquiry-based learning, which has been shown to increase student abilities in scientific literacy[15,33] as well as students' motivation to learn[25]. As we move through the intervention, we also make use of PBL. This allows us to let students who are progressing quickly in activities to work ahead or on their own while we assist others who are struggling. By asking the students questions about the task, often relating it to real world or previous classroom experiences, they will often discover how to use the blocks available to them in Scratch to solve the task. Throughout the intervention, we keep removing scaffolding until the last day where students are tasked with their final project.

The goals of our methodology is to maximize the increase in student self-efficacy. Self-efficacy can be defined as "an individual's belief that they can accomplish a particular task"[4].

Measuring self-efficacy relative to CS and even more so in CT is required, because there does not yet exist any widely-adopted standardized assessments which measure student progress (apart from AP CS). Bandura notes that self-efficacy can by improved through enactive attainment, vicarious experiences, verbal persuasion, and psychological state[1]. By using PBL we enable the students to achieve tasks on their own without direct instruction. This relates to enactive attainment (individual mastery of skills), although we have to structure problems carefully so that they are not too easy (students will get bored) or too difficult (increased anxiety). This is also referred to as the zones of proximal flow and development[25]. Verbal persuasion occurs in our intervention through IL. Though IL is indirectly guiding the student (asking the right questions), we are able to convince students that they are able to solve the tasks at hand, whether it's difficulty with a CT concept or a technical problem with placing the correct blocks in Scratch. Vicarious experiences are achieved through group activity. During activities, students are encouraged to talk to their neighbors about how they solved the programming tasks, and in others, students are partnered up for group projects. Finally, a stable psychological state is achieved through our scaffolding and use of a block-based language to reduce the cognitive load. By designing our interventions around Bandura's methods of improving self-efficacy, we craft powerful and achievable learning experiences.

Mission to Mars

Students in the lower grade levels (5th, 6th, and 7th grade) attended a program called Mission to Mars. The goal of this intervention design was to introduce students to CT through many different activities revolving around tasks that must be completed to send an autonomous rover to Mars. Each session of the program consisted of four days of activities (each day being 3 hours long).

The first day was primarily an introduction to the Scratch visual programming language. The students were led through many short activities to familiarize themselves with the language, culminating in a challenge to draw regular polygons (2015)[4] using methods very similar to the turtle graphics features of the classic Logo programming language. As the students slowly built shapes with more sides, concepts such as iteration, variables, user input, and mathematical operators were introduced, leading to a generalized program that could draw any regular polygon. We also briefly introduced the fact that this program demonstrated a fundamental theory of calculus.

The second day focused on using computers to simulate real-world ideas. The students began by playing with human-powered compressed air rockets (Stomp Rockets). While doing so, they plotted the distance each rocket traveled and discussed reasons for the wide variance of results. This led to a discussion of the scientific method, independent and dependent variables, and how to design an accurate experiment. Afterwards, students were led through an activity to simulate a rocket's trajectory in Scratch, using the launch angle as the independent variable. With that knowledge, students were introduced to high-performance computing as a way to solve even bigger problems, such as the trajectory of a real rocket, and were given a guided tour of a nearby supercomputer. Students then learned how to create a simple acceptor finite state machine that accepts a secret key though a series of clicks.

The third day introduced the concept of artificial intelligence (AI). First, students were led through a project to recreate three of the four enemy AI ghosts from the classic Pac Man arcade game. In doing so, they were introduced to a two-step artificial agent pattern of perceiving the environment and acting based on that perception. Following that, students were assigned an activity to explain how more complex AI, such as neural networks, can be trained. After completing that activity, the students were introduced to the final project: building an AI for an autonomous Mars rover. The concept was first shown to them as a game, where they were challenged to get the highest score possible. This required planning ahead to find the best path and learning how the rover operates. These activities drew on many areas of CT, including modeling and simulation, abstraction, and data representation.

On the fourth day, the rover was re-introduced as a game, but this time the rover could only see the squares immediately adjacent to it. This required students to "sense" their surroundings and act based on limited information, just as the rover would. This helped reinforce the "perceive" phase of an artificial agent and forced the students to adjust their thought process to match that of an algorithm. Finally, the students were given a rover project that allowed them to build an AI following the perceive-and-act model previously used. They worked independently but with some guidance on how to build the best rover AI possible, and compared their results with other students.

Game Design

The second intervention was for the 8th and 9th grade students. This intervention focused on video game design within Scratch. While game design contains a significant amount of established theory, we focused only on a small subset of common principles of game design inspired by *100 Principles of Game Design*[11] and a popular YouTube series[27]. While the delivery focus of this intervention was game design principles, we used the development of different games in Scratch to teach CT concepts. Like Mission to Mars, this program consisted of four days of activities.

Day one began with an introduction to game design principles. These consisted of seven principles: (1) Principle of isolation: introducing new elements in a way that allows players to familiarize with new enemies or mechanics before they are set in a real situation. (2) Principle of accomplishment: gives players a sense of motivation and direction either through story progression or the mastering of skills. (3) Teach without teaching principle: help players learn by doing instead of relying on step-by-step tutorials. (4) Growing stronger principle: a game storyline can often be rewarding alone; however, progression can be improved by letting the player grow stronger and accomplish tasks that they could not earlier in the game. (5) Silent storytelling principle: allow the player to experience the story for themselves instead of having it spelled out. (6) Hidden reward principle: give the player extras (bonus levels, collectables, etc.) to add an extra feeling of accomplishment beyond the original gameplay/story. (7) Balance principle: gameplay must have a good balance between boredom and anxiety to keep the player interested and coming back. We chose these principles due to their relation to educational theory and how our scaffolding is constructed. Examples of these principles were discussed in popular video games. Students were also asked to give examples of the principles from games that they play at home. This discussion was followed by an introduction to scratch using shapes as mentioned in day one of the Mission to Mars intervention. Students were then asked to split off into pairs or groups of three to brainstorm their own game for as a final project of the course.

The second day focused on introducing basic AI concepts, an important aspect of video games. Students were asked to think about what it means to be intelligent. Most responses tended to be things like "smart." After describing intelligence as reasoning, problem solving, ability to construct knowledge, planning, learning, and perception (of which all relate back to the core concepts of CT), students were presented with the Turing test and how computers could be considered “intelligent.” We discussed the importance of AI in video games and began the first game tutorial called Cat and Mouse. This is a partially completed game where students are walked through implementing a basic AI for a cat that chases a mouse, the player, which tries to eat pieces of cheese. After they had a working game, students were presented a problem to improve the AI to exhibit more complex behavior. As a follow up, we worked with the students to complete the starter AI for the game *Strikers 1945*. The day ended with time for students to complete storyboards for their final project.

The theme of day three was dungeon crawlers, a classic game style. To demonstrate this, students were given a starter project for *One Tap Quest*, a simple, yet popular dungeon crawler/RPG. This game was used to illustrate all of the game design principles taught since the first day. *One Tap Quest* requires only a single click from the player and their hero starts off on a quest through a randomized set of enemies to slay for experience and power-ups to collect before reaching the boss. We walked students through setting up randomization of the first level of monsters. They were then tasked with adding another level of monsters, as well as a power-up. The rest of the day was left for students to work in their group on their final project. Before students left for the day, a discussion was led on career options in the video game industry.

The final day was reserved time for groups to work on their projects while we walked around to assist. At the end of the day, groups got up in front of the class to demonstrate their games and describe what game design principles they used. Groups were allowed to use any of the seed projects used any of the previous days, as long as they added additional content or mechanics. Some groups did use the seed projects, but most designed their own game and used what they learned from programming the seed projects as the basis for their mechanics. To encourage the students to continue to collaborate, all projects from each week were added to a Scratch studio.

Table 1: Number of students (after survey exclusion) in each intervention

	Mission to Mars				**Game Design**			**Total**
Grade Level	5th	6th	7th	Total	8th	9th	Total	
Week 1	0	8	5	**13**	8	2	**10**	23
Week 2	3	7	3	**13**	6	3	**9**	22
Week 3	1	9	6	**16**	6	5	**11**	27
Week 4	0	5	3	**8**	5	6	**11**	19
Total	**4**	**29**	**17**	**50**	**25**	**16**	**41**	91

Instrument Design

We developed a hybrid instrument, combining the questions 6-17 from the Self-Efficacy for Computational Thinking (SECT) survey (with the addition of a question about Boolean operations) (2015)[4] with questions extracted from the math (27, 28, 31,and a new question: “I can apply math concepts to other subjects”), science (35-37, 40, 42), engineering and technology (44 – changed products to things, 45, 50-52), and 21st century skills (38, 44, 46, 48, and a new question “I am confident I can manage my time wisely when working in a group”) sections in a survey built for measuring attitudes towards STEM[12]. The new questions were added for better coverage of our interventions. Additional questions were asked about the student’s previous experience in STEM activities (if they attended this institute before or any other STEM-related outreach activities), as well as whether they had previous experience programming in the Hour of Code, Scratch, Blockly, TouchDevelop, text-based languages, or any other computer language. A teacher survey was also created in a similar fashion by extending the Teacher Self-Efficacy for Computational Thinking (TSECT) survey from (2015)[4] to include teachers' background in STEM activities, interest in teaching STEM, and experiences with programming languages. The full surveys are excluded from this paper in interest of length and can be produced upon request, although an abbreviated, partial list of questions can be found in Table 3.

Pre-surveys were administered online at the beginning of each week long session. The post-survey (excluding initial background questions) was given on the last day of each session (day four) after ending discussions. While both the student survey and the teacher survey were optional, the student survey was administered during each session, and the teacher survey was only emailed each session. Out of 94 surveys sent to all educators and pre-service teachers involved with the institute, only 33 responded to the pre-survey and fewer than 10 responded each week for the post-survey. For this reason, results for the teacher survey are excluded from analysis. Student response rate (after exclusions) can be seen broken down in Table 1. Out of 101 student respondents, 7 were excluded for not taking the post-survey (absent those days), one was excluded for not taking the pre-survey (absent), and two were excluded for incomplete surveys (the missing data in these responses were classified as MCAR). The reliability of our instrument was confirmed with a Chronbach's Alpha of .908.

Findings

Overall, both interventions showed a statistically significant ($p < .001$) positive gain in CT concepts from the pre-survey ($M = 52.39, STD = 27.7$) to the post-survey($M = 64.76, STD = 21.3$). From each intervention we gathered background information in STEM, including programming experience as seen in Table 2. Surprisingly, over 30% of students had been exposed to a text-based programming language. Over 59% of students had experienced some sort of block-based programming language, and half had participated in the Hour of Code. Less than half of those who participated in the Hour of Code (which is written using the Blockly language) knew that they were using Blockly. From students who had previously attended some sort of STEM program before the institute, 70% of them had used Scratch. This shows that most outreach programs in this geographic area highly favor the Scratch language. With more than 80% of students having used some programming language, it shows that all students are being exposed as much to computer programming at home or school as those who participated in outreach programs. However, the low level of exposure is reflected in the self-efficacy in CT concepts.

Students who had previously attended outreach programs improved more in CT concepts such as algorithms, procedures, parallelization, data collection, and data representation, as shown in Table 3 (this includes students who attended this particular institute before as well as those who attended other outreach programs). This hints that even though both samples of students had about the same amount of experiences using programming languages, STEM outreach programs have better success in seeding CT skills in students, compared to exposures in school or at home.

Table 2: Programming experience before the interventions

	Any Language	Hour of Code	Scratch	Blockly	TouchDevelop	Text-based	Other
No Previous Attendance in STEM Programs (37 students)	83.78%	51.35%	56.76%	18.92%	16.22%	35.14%	35.14%
Previous Attendance in STEM Programs (54 students)	81.48%	51.85%	70.37%	20.37%	18.52%	35.19%	31.48%
Overall – Mission to Mars	84.00%	52.00%	58.00%	18.00%	16.00%	32.00%	34.00%
Overall – Game Design	80.49%	51.22%	60.98%	21.95%	19.51%	39.02%	31.71%

Table 3: Comparison of effect sizes for the 21st century learning and CT focused questions

Abbreviated Question	No Previous Attendance in STEM Programs	Previous Attendance in STEM Programs	Overall	
			Mission to Mars	Game Design
Math is my worst subject	-.132	0	-.117	.024
Consider a career that uses math	.264	.127	.156	.215
Perform well in other subjects, but not math	-.193	.050	-.125	.044
Apply math to other subjects	.058	.020	.129	-.079
Consider a career in math	.138	.151	.143	.149
Like to imagine creating new things	.036	.098	-.106	.291
If I learn engineering, I can improve things people use everyday	.191	.196	.165	.230
Would like to use creativity and innovation in my future work	.087	.199	.064	.262

Math and science together will help me invent useful things	.251	.064	.255	0
I can be successful in a career in engineering or technology	.113	-.039	-.021	.077
Lead others to accomplish goals	.501	-.023	.198	.181
Work well with others who have different backgrounds and opinions	-.033	.135	.097	.030
Make changes when things don't go as planned	.036	.098	.132	0
Manage my time wisely when working on my own	.027	.074	.060	.048
Manage my time wisely when working in a group	.088	.141	.153	.0798
Executes a sequence of commands	.352	.632	.496	.546
Uses loops to repeat commands	.641	.785	.683	.779
Responds to events	.259	.539	.584	.231
Parallelism	.482	.656	.556	.621
Conditional commands	.498	.508	.582	.408
Perform math operations	.265	.387	.481	.162
Perform Boolean operations	.606	.626	.626	.608
Store, update, and retrieve values	.405	.550	.429	.568
Ask user for input	.292	.694	.537	.522
Iterative development	.331	.417	.322	.456
Frequent tests/debugging	.519	.481	.533	.452
Share and collaborate with programs	.337	.573	.445	.517
Break program into parts	.537	.412	.448	.482

One could say as well that since the students who attended STEM programs previously had more exposure to Scratch (70.37% vs. 56.76%), they were able to move more quickly through our activities and focus more on learning CT concepts rather than the language itself. Students who had not attended STEM programs previously showed higher pre-survey self-efficacy than those who had. We hypothesize that since they may not have been exposed to CT as much, this led to overconfidence, which is reflected in the amount of improvement when looking at post-surveys. By comparison with the overall program, the two interventions were less distinguishable, though students in Mission to Mars had a strong improvement in self-efficacy in writing programs that respond to events and for being able to perform math operations in their programs. However, after inspecting mean pre- and post-survey responses, the mean pre-survey self-efficacy for Mission to Mars in these questions was much lower than that of the Game Design intervention, although the mean post-survey responses were nearly equivalent.

This verifies that even though the topic of interest in each intervention is different (as well as the age groups), the end results for both are comparable.

When looking at 21st century learning skills, improvements were less noticeable as most of our students came in with high confidence in these areas. For example, over 80% of students came into our sessions highly confident in math, which led to little improvement. However, students who had previously not attended STEM outreach programs showed stronger improvement for their value of math and science in inventing new things. Leadership also showed a strong improvement in these students, which reveals that the STEM outreach programs are doing well in improving student confidence in leading others to accomplish goals. Only weak improvements are present when comparing the two interventions, though Game Design had slightly stronger results in imagination and creativity. This is due to the fact that the Game Design intervention offered more room for students to create and implement their own ideas in their final project video game.

Conclusions and Future Work

In this paper, we discussed two novel interventions applied during a 5th-9th grade summer STEM institute. Though both interventions differed in topics (video games vs. Mars rover), they showed similarly strong improvements in student self-efficacy in CT concepts. This pedagogy shows that it has a positive impact how CT concepts are delivered through CS and computer programming at the K-12 level. Likewise, by expanding the survey done by Bell, we were able to gain more insight into specific CT concepts learned by students in a similar environment[5]. Furthermore, we were able to reveal some lasting impacts that STEM outreach programs have on students who continue to stay active in science, technology, engineering, and math activities. These students who have participated in the outreach programs show greater capacity in improving their CT skills over those who have not. This important finding cannot be explained completely with data we collected, though we have made hypotheses, and warrants further investigation through revised instruments or longitudinal studies. Background survey questions also revealed that the STEM outreach in our areas (apart from this summer institute) does not have large participation by upper middle school and high school students (19 students in 5th-7th grade vs 4 students in 8th-9th).

Future work will include ways of improving advertisement of other outreach programs to reach a wider audience. MOOCs are also a future possibility as they have shown promise in other research[13,30]. In order to dive deeper in understanding transfer and self-efficacy of CT, our future work will also include studying the move from block-based to text-based programming language[31]. We also learned from our survey administration, that we need to use a different medium in giving educator assessment surveys. Based from the administration of the student surveys, we believe rewording questions to be friendlier for lower grade levels will help with levels of overconfidence in pre-survey responses.

Acknowledgements

We would like to thank Jesse Peters and Brooke Snyder, our K-12 in-service teachers, as well as the pre-service students who observed and assisted our classes. Many thanks are also extended to the large number of staff at USD 383 for organizing this event. This research was made possible by the Department of Defense Education Activities grant "STEMing into College and Career Readiness." This material is also based upon work supported by the National Science Foundation under Grant No. 0948019. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Bandura, A. (1982). Self-Efficacy Mechanism in Human Agency. *American Pyschologist, 37*(2), 122-147.
2. Barr, D., Harrison, J., & Conery, L. (2011). Computational Thinking : A Digital Age Skill for Everyone. *Learning & Leading with Technology, 5191*, 20-23.
3. Barr, V., & Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community ? *ACM Inroads, 2*(1), 48-54.
4. Bean, N., Weese, J. L., Feldhausen, R., & Bell, R. (2015). Starting From Scratch: Developing a Pre-Service Teacher Program in Computational Thinking. *Frontiers in Education*.
5. Bell, R. S. (2014). *Low Overhead Methods for Improving Capacity and Outcomes in Computer Science.* Manhattan, KS: Kansas State University.
6. Bennett, V., Ioannidou, A., Repenning, A., Kyu Han, K., & Basawapatna, A. (2011). Computational Thinking Patterns. *American Educational Research Association*.
7. Bliss, J., & Askew, M. (1996). Effective Teaching and Learning: Scaffolding Revisited. *Oxford Review of Education, 22*(1), 37-62.
8. Blum, L., & Cortina, T. J. (2007). CS4HS: An Outreach Program for High School CS Teachers. *Proceedings of the 38th SIGCSE technical symposium on Computer science education* , (pp. 19-23).
9. Bort, H., & Brylow, D. (2013). CS4Impact: measuring computational thinking concepts present in CS4HS participant lesson plans. *Proceeding of the 44th ACM technical symposium on Computer science education.*
10. Code.org. (2015). *Code.org*. Retrieved from https://code.org
11. Despain, W. (2013). *100 Principles of Game Design.* New Riders.
12. Faber, M., Unfried, A., Corn, J., & Townsend, L. W. (2012). *Student Attitudes toward STEM: The Development of Upper Elementary School and Middle / High School Surveys.* Friday Institute for Educational Innovation.
13. Falkner, K., Vivian, R., & Falkner, N. (20015). Teaching Computational Thinking in K-6: The CSER Digital Technologies MOOC. *Australian Computing Educaiton Conference*, 27-72.
14. Google. (2015). Retrieved from Blockly: https://developers.google.com/blockly/
15. Gormally, C., Brickman, P., Hallar, B., & Armstrong, N. (2009). Effects of Inquiry-based Learning on Students' Science Literacy Skills and Confidence. *International Journal for the Scholarship of Teaching and Learning, 3*(2).
16. Hmelo-Silver, C. E., Duncan, R. G., & Chinn, C. A. (2007). Scaffolding and Achievement in Problem-Based and Inquiry Learning: A Response to Kirschner, Sweller, and Clark (2006). *Educational Psychologist, 42*(2), 99-107.
17. Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist, 41*(2), 75-86.
18. Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015). Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15* (pp. 21-29). Omaha, NE: ACM.
19. N. G. (2010). *Common Core State Standards.* Washington, DC: Authors.
20. National Academy of Sciences. (2010). *Report of a Workshop on The Scope and Nature of Computational Thinking.* National Academies Press.
21. Neilsen, M. L., Shaffer, J., & Johnson, N. (2015). Time Lapse Photography for K-12 Education. *Int'l Frontiers in Education: CS and CE*, 201-207.
22. NGSS Lead States. (2013). *Next Generation Science Standards: For States, By States.* Washington, DC: The National Academies Press.
23. P21 Partnership for 21st Century Learning. (2015, November). Retrieved from http://www.p21.org
24. Plass, J. L., Moreno, R., & Brunken, R. (2010). *Cognitive Load Theory.* Cambridge University Press.
25. Repenning, A., Webb, D. C., Kho, K., Nickerson, H., Miller, S. B., Brand, C., . . . Repenning, N. (2015). Scalable Game Design: A Strategy to Bring Systemic Computer Science Education to Schools though Game Design and Simulation Creation. *ACM Transactions on Computing Education*, 1-31.

26. Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., . . . Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM, 52*(11), 60-67.
27. s. G. (2015). *Snowman Gaming: Home of Good Game Design*. Retrieved from YouTube: https://www.youtube.com/channel/UCmY2tPu6TZMqHHNPj2QPwUQ
28. Savery, J. R. (2009). Overview of problem-based learning: Definitions and Distinctions. *Interdisciplinary Journal of Problem based Learning, 1*(1), 269-282.
29. Sengupta, P., Kinnebrew, J. S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*(2), 351-380.
30. Spradling, C., Linville, D., Rogers, M., & Clark, J. (2015). Are MOOCs an appropriate pedagogy for training K-12 teachers computer science concepts? *Computing Sciences in Colleges, 30*(5), 115-125.
31. Weintrop, D., & Wilensky, D. (2015). Using Commutative Assessments to Compare Conecptual Understanding in Block-based and Text-based Programs. *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15*, 101-110.
32. Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33-35.
33. Wu, H.-K., & Hsieh, C.-E. (2006). Developing Sixth Graders' Inquiry Skills to Construct Explanations in Inquiry-based Learning Environments. *International Journal of Science Education, 28*(11), 1289-1313.