STEM Outreach: Assessing Computational Thinking and Problem Solving

Joshua Levi Weese, Kansas State University, weeser@ksu.edu
Russell Feldhausen, Kansas State University, rusfeld@ksu.edu

Abstract: The ever-growing popularity of computer science has fostered the need for computational thinking (CT), especially in K-12 education. Pedagogy that infuses CT, as well as reliable methods for assessing CT, remain open problems. In this paper, we describe a 5th-9th grade STEM outreach program. Classes on micro controllers and computer programming are presented. Data collected through a newly designed self-efficacy instrument is used to determine effectiveness of these curricula at improving confidence in CT and problem solving skills.

Introduction

This paper describes a STEM outreach program where the Manhattan-Ogden Unified School District 383 has partnered with Kansas State University. This program lasts four weeks and is designed to expose 5th-9th grade students to STEM careers and subjects through hands-on activities. The program covers a large range of areas, including robotics, computer programming, agriculture, food science, unmanned aerial vehicles, clean energy, and construction science. Professional educators are paired with small groups (2-4) of pre-service teachers to run each class (maximum size of 18). This allows pre-service teachers to get practical, hands-on experience, as well as to learn new STEM activities to include in their own future classrooms. This also gives an excellent teacher to student ratio, providing a one-on-one learning experience for program participants. We focus, however, on measuring the impact of two classes on the program participants. Each class employed similar pedagogy and the Scratch (2009)[17] programming language. One relied heavily on computer science theory and space exploration as a theme, and the other used micro controllers as the foundation for activities. The goals of this research are as follows: 1. Develop effective curricula for improving student self-efficacy in CT, 2. Develop a reliable and effective way of measuring student self-efficacy in CT, and 3. Enforce the notion that CT is not problem solving (PS), but a component of cognition.

Background and Related Work

"Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science"[26]. However, computational thinking (CT) is not intended to be equated to computer science; rather the essence of CT comes from thinking like a computer scientist when faced with problems from any discipline[8]. Wing expanded the definition of CT in 2011, mentioning that CT is "the thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information processing agent"[27]. The inclusion of intelligent agents in what embodies CT creates a pathway to inclusion in multiple disciplines by means of scientific simulation and real-world problem sets. In a report by the Royal Society, the need to incorporate CT in curriculum is emphasized and defined as "the process of recognizing aspects of computation in the world that surrounds us, and applying tools and techniques from computer science to understand

and reason about both natural and artificial systems and process"[18]. This encompasses the vision of CT, drawing from traditional computer science to encourage new ways of thinking about the world around us.

Computational thinking can be expanded by defining it in terms of concepts, practices, and perspectives[5]. CT concepts have been a popular target for research and curricula development; however, the concepts vary across domains. Some, such as Brennan and Resnick, present CT concepts as referenced in their problem domain (Scratch), while others, such as the Computer Science Teachers Association (CSTA), present CT concepts for application across K-12 curricula[19]. Even from within the general domain application, what authors include in their definition of CT concepts vary[6,8,19]. Brennan and Resnick define computational concepts (also referred to as CT concepts) as the "concepts that designers employ as they program." To encompass more fields, CT concepts are generalized as the usage of one of the computer science principles listed in Table 1 in solving a problem:

Table 1 Computational Thinking Concepts and Related Computer Science Principles

Abbr.	Description
ALG	**Algorithmic thinking** – sequence of steps that complete a task. Operators and expressions are also included.
ABS	**Abstraction** – generalized representation of a complex problem, ignoring extraneous information
DEC	**Problem decomposition** – breaking a problem into smaller, more manageable parts that can be solved independently of each other
DAT	**Data** – collection, representation, and analysis of data[6]
PAR	**Parallelization** – simultaneous processing of a task[6]
CON	**Control flow** – directs an algorithm's steps when to complete
IAI	**Incremental and iterative** – building small parts of the program at each step instead of the whole program at once
TAD	**Testing and Debugging –** performing intermediate testing and fixing problems while developing
QUE	**Questioning** – working to understand each part of the code instead of using code that is not understood well

The importance of incorporating CT into existing curricula and programs is not always an easy task, especially in K-12 where problem solving (PS) skills are traditionally emphasized. Most non-computer science educators see the importance of teaching CT, though it is often difficult to distinguish CT from PS. Therefore, in many cases, they assume teaching PS is sufficient to also teach CT. Authors Voskoglou and Buckley define PS as "an activity that makes use of cognitive or cognitive and physical means to overcome an obstacle (problem) and develop a better idea of the world that surrounds us"[23]. Voskoglou and Buckley also note that humans, even college graduates, struggle applying theory and problem solutions to practice. The authors continue to describe CT as a hybrid mode of thinking, combining logical, abstract, modelling, and constructive thinking. By synthesizing critical thinking and existing knowledge, these modes of thinking are essential in solving real-life technological problems[23]. PS is an activity that combines various components of cognition (2005,2012)[7,23], concluding that CT is a type of cognitive skill used to solve particular types of problems.

Measuring Computational Thinking

Measuring computational thinking remains an open problem. One of the commonly used approaches focuses on using evidence-centered assessments[21]. Developed curricula vary for

assessments, but video games seem to be a very common theme as it provides a high level of interest for students in the K-12 environment[25]. Games were encoded under three main categories: programming concepts, code organization, and designing for usability. Each of these contained sub categories which were coded for the presence/absence of or to the extent of which that sub category was used ranging from zero to three. Repenning et al. measured student's learning of CT as patterns rather than concepts using Agent Sheets[11]. This led to an automatic analysis tool using latent semantics to determine student growth in CT. Recently, the same group created a system entitled Real-Time Evaluation and Assessment of Computational Thinking (REACT), a real-time assessment tool allowing teachers to get immediate feedback on what students are struggling with or where they are succeeding[2]. Seiter developed the Progression of Early Computational Thinking model (PECT)[20]. The PECT model combined evidence of programming concepts in Scratch projects with levels of proficiency (basic, developing, and proficient) in a set of design patterns to understand student ability in CT.

Apart from evidence-centered assessments, researchers employ cognitive science methods like attitude or self-efficacy surveys[16]. Self-efficacy can be defined as one's belief that they can accomplish a task[1]. Self-efficacy has also been linked to learning outcomes[13], making it an effective, low-overhead method for assessment. Yadav et al. created an open-ended questionnaire, as well as an attitude survey to understand if introducing computational thinking material in pre-service education courses influenced pre-service teachers' understanding of CT and attitudes toward computing[28]. Likewise, Bean et al. developed a self-efficacy survey for measuring pre-service teachers' confidence that they are capable of incorporating computer programming into their classroom, how programming relates to core curriculum standards, and CT skills[3]. These authors also created a CT self-efficacy survey for an art-based class part of a STEM outreach program. Their experimental approach served as a basis for this work[4].

Methods

Teaching programming can be a difficult task when involving students who have no background in foundational computer science skills. Our curriculum emphasizes reducing cognitive load through scaffolded examples and the Scratch programming environment which eliminates complex syntax and programming errors. We also utilize problem-based learning and inquiry learning (2016)[24], effectively improving student self-efficacy through vicarious experiences, verbal persuasion, enactive attainment, and psychological state (reducing cognitive load)[1].

From these methods, we created two different curricula as part of the summer STEM outreach program. The Saving the Martian (Mars) class for 5th and 6th grade students introduced CT using the Scratch programming environment. Many of the activities were modeled on situations or ideas taken from *The Martian*, by Andy Weir, to make them more interesting and exciting for the students. The Mighty Micro Controllers (MMC) class was for 7th-9th graders and focused on teaching CT through programming Arduino Uno micro controllers using Scratch. It also included a short exposure to the Arduino IDE and text-based language. Physical computing has been shown to be a successful method of teaching computer science concepts and introductory computer programming[14,15]. Overall, the format of this class included guided examples on how to create certain circuits and programs, followed by problem driven exploration to help enforce programming, electrical, and CT

skills. MMC heavily utilized pair programming. Each class consisted of four days of activities lasting three hours each. As we describe each activity below, we will highlight the CT areas it was designed to cover using the information from Table 1 above.

Saving the Martian

Students were first introduced to the Scratch environment by building a computer program that could draw an n-gon (regular polygon with n sides). Students were shown a sample program that drew three lines and were asked to modify it to create various shapes. As the number of sides grew larger, students were introduced to iteration to reduce the amount of code in the program. Students were encouraged to calculate angles and request user input for the number of sides as variables instead of hard-coding those values. At each step, we emphasized the importance of testing their program and make sure that each block used was understood. (ALG, ABS, CON, IAI, TAD, QUE)

The class then transitioned into sorting algorithms. The students participated in a sorting network activity where they followed lines on the floor that intersected, changing their direction based on some value, with all students having a higher value going one direction and all students with a lower value going the other. Afterwards, students were taught to sort playing cards using several common algorithms such as insertion sort and bubble sort. Students were then shown how to swap variables in Scratch, and applied that knowledge along with iteration from the previous day to write a bubble sort program. For the second activity, students were shown how to create a simple simulation program in Scratch to demonstrate how a chemical reaction would alter the presence of different materials in the atmosphere of a Mars habitat. Once the simulation was started, if the output of certain materials became too high, the simulation would stop due to a failure. Students were encouraged to adjust the variables of the simulation to see if they could find a way to grow plants fast enough to sustain life. (ALG, ABS, DEC, DAT, PAR, CON, IAI, TAD, QUE)

Next, students were introduced binary and hexadecimal number systems using cards as value placeholders. Students were shown the scenario from *The Martian* where the main character must communicate with others using only a camera that rotates by placing sixteen signs around the camera representing hexadecimal values, converting each pair of values to its equivalent ASCII value. Students were given a similar situation in Scratch, and were lead through the process of translating the data to ASCII. This involved calculating the angle of the camera, converting it from a degree value to a hexadecimal value, and then converting a sequential pair of values into an ASCII character. As an added learning experience, the original version of the program contained an intentional typo in the message received, leading to ambiguity in the message. Students were encouraged to describe ways the system could be improved to minimize or eliminate ambiguous messages. (ALG, ABS, DEC, DAT, CON, IAI, TAD, QUE)

Finally, students were introduced to several concepts in artificial intelligence, including the Turing test and neural networks. Afterwards, students were lead through an activity to create simple AI agents for an arcade video game following a simple perceptron model. The final activity built upon that structure by using a situation from *The Martian*, where the main character must plot a course around several terrain obstacles while driving across the surface of Mars. The students were given a program that randomly generated simple obstacles on a terrain, and upon reaching an obstacle, students had to use a simple

perceptron model to determine how to get around the obstacles while still moving toward the goal. (ALG, ABS, DEC, CON, IAI, TAD, QUE)

Mighty Micro Controllers

Students were first introduced to the basic principles of electricity. Most had their first exposure to electricity, as well as the concepts of conductivity and insulation. By using an example of marbles in a tube (2014)[12], students could visualize and understand the flow of electrons. Once this basic principle was established, students were introduced to resistance, voltage, and current (Ohm's Law), as well as digital, analog, and pulse width modulation (PWM) signals. Before wiring the first circuit, students were required to create a circuit diagram (diagrams were provided for all other activities) using Fritzing to visualize how the circuit should be laid out[10]. After everyone completed the blinking LED example, students were introduced to an activity called "Resistance is Futile." They were challenged to rank a set of resistors in order of strength. By using the previous blinking LED circuit and program, students could visualize how resistors impede the flow of electrons. We then expanded the single blinking LED to include five LEDs of different colors. This led to a discussion about abstraction, problem decomposition, and reusing/remixing. The class was guided through the process to recognize patterns in the program to reduce the number of blocks that were repeated by using Scratch custom blocks. The next activity extended this program to include pushbuttons. This focused on teaching analog signals, pull-down circuits, open/closed circuits, and control flow. (ALG, ABS, DEC, CON, IAI, TAD, QUE)

After working with basic LEDs and digital signals, we introduced activities for PWM signals. The first utilized RGB LEDs. Before being able to program, students needed to learn how to convert colors to traditional RGB format and how that was translated to the RGB LED connected to the Arduino. Abstraction and custom blocks were emphasized to make setting the different intensities of red, green, and blue simple. Students were then given a large amount of discovery time to see what kind of colors they could produce using this circuit. Afterwards, a complete program that gradually changed through all the colors the LED could make was demonstrated. A video of an RGB LED matrix was also used to inspire students with more ideas for a final project. (ALG, ABS, DAT, CON, IAI, TAD, QUE)

We continued the class with sensors to emphasize analog signals. This included a guided activity using small motion and ultrasonic sensors, and the applications of how the sensors could be used in everyday life. The ultrasonic sensor activity was done using the Arduino IDE because Scratch was not able to accurately detect distance with the sensor. A side-by-side comparison was used with Scratch and the Arduino IDE to demonstrate how the text-based language translated into Scratch blocks. After this activity, students were given time to complete group projects where they could design, build, and program their own circuits. They had to produce a design document, containing a circuit diagram and materials list, before they could start building or programming. This helped emphasize the engineering design process, as students had to keep revising their design when they discovered flaws in their original circuit. At the end of the day, students presented their projects to the class to strengthen their technology communication skills. (ALG, ABS, DEC, DAT, CON, IAI, TAD, QUE)

Instrument

To measure student learning, we developed a self-efficacy survey to collect attitudes about students' ability to think computationally. This survey largely expands our previous work[3,24] by adjusting question language to be more age appropriate with our audience, as well as with the addition of questions assessing student self-efficacy in problem solving. These questions are framed to correlate to appropriate computational thinking skills. As such, we organized the survey in four main sections: problem solving, computer programming skills, computer programming practices, and computer programming impact. Questions are categorized by relevant CT concept, practice, or perspective as seen in Table 2. Each of these questions measured self-efficacy on a five-value Likert scale: strongly agree, somewhat disagree, not sure, somewhat agree and strongly agree. Apart from these questions, the survey also contained questions collecting information about gender, participation in STEM activities/camps, and background in computer programming.

Table 2 The four core sections of the self-efficacy survey, denoting which CT skill each question falls under.

When solving a problem I...			I can write a computer program which...		
1	create a list of steps to solve it	Algorithms	10	runs a step-by-step sequence of commands	Algorithms
2	use math	Algorithms	11	does math operations like addition and subtraction	Algorithms
3	try to simplify the problem by ignoring details that are not needed (3)	Abstraction	12	uses loops to repeat commands	Control Flow
4	look for patterns in the problem	Abstraction	13	responds to events like pressing a key on the keyboard	Control Flow
5	break the problem into smaller parts	Problem Decomposition	14	only runs commands when a specific condition is met	Control Flow
6	work with others to solve different parts of the problem at the same time	Parallelization	15	does more than one thing at the same time	Parallelization
7	look how information can be collected, stored, and analyzed to help solve the problem	Data	16	uses messages to talk with different parts of the program	Parallelization
8	create a solution where steps can be repeated (8)	Control Flow	17	can store, update, and retrieve values	Data
9	create a solution where some steps are done only in certain situations (9)	Control Flow	18	uses custom blocks	Abstraction
When creating a computer program I...			When creating a computer program I...		
19	make improvements one step at a time and work new ideas in as I have them	Being Incremental and Iterative	22	break my program into multiple parts to carry out different actions	Problem Decomp.
20	run my program frequently to make sure it does what I want and fix any problems I find	Testing and Debugging	Impact		
21	share my programs with others and look at others' programs for ideas	Reuse/Remix, Connecting	23	I understand how computer programming can be used in my daily life.	Questioning

Our experiment was carried out in a pre-post survey format. Pre-surveys were administered online on the first day of each week-long session before any class material was given. The post-survey, which did not contain demographic or STEM participation questions, was given on the last day of each session once all projects were finished. Survey participation was voluntary. Out of 110 students, one student was excluded for opting out of the survey, one student was excluded for missing the pre-survey, and three students were excluded for having incomplete responses. A Chronbach's Alpha of .872 on the pre-survey and .908 on the post-survey shows that our survey described in Table 2 is reliable.

Table 3 The distribution of students included in the survey within each compared group as well as average pre and post self-efficacy.

	Mars	MMC	Male	Female	No-STEM	STEM	OUTSIDE STEM	STARBASE	STEM INST
% of Students	53.33%	46.67%	65.71%	34.29%	29.52%	70.48%	35.24%	27.62%	35.24%
Avg. Pre Mean	3.763	3.619	3.71	3.661	3.642	3.718	3.673	3.671	3.764
Avg. Post Mean	4.187	3.884	4.083	3.978	4.016	4.058	4.108	4.165	4.008
Avg. Std. Dev.	1.016	1.056	1.048	1.032	1.074	1.026	1.043	0.988	1.006

Results

In the analysis of survey results, we looked at 8 groups outlined in Table 3: Saving the Martian (Mars), Mighty Micro Controllers (MMC), male, female, no previous participation in STEM activities/groups (No-STEM), previously attended any STEM program (STEM), previously attended a different STEM program (Outside STEM), previously attended this STEM program (STEM INST), and previously attended Starbase (a separate STEM outreach program). Average over questions 1-23 pre and post means can also be observed in Table 3 Table 4 shows the effect size[22], calculated using pooled standard deviation, for each question, broken down by group. Note that effect sizes of 0.2 are considered small, 0.5 are medium, and 0.8 are large.

Overall, 70.42% of students had previously attended some STEM related group activity or program, while nearly all students in the program had used a visual-based programming language, mostly though Scratch, Hour of Code, and Lego robotics. These results show that in our area, outreach efforts are beginning to spread through the local K-12 population. Due to small sample sizes, we were not able to break the groups in Table 3 and 4 into language background (Scratch vs Lego robotics for example). Students who had never

attended any type of STEM program performed just as well as those who had attended a STEM program, apart from problem solving skills. The inverse applies when comparing the outside STEM group to those who had previously attended this STEM program. We hypothesize that this result is partially due to a selection bias with this STEM camp and Starbase, which accounts for 78% of the outside STEM group. Those who had participated in Starbase in the past showed significantly higher effect sizes on most CT concepts. This could be explained by the difference in the two programs. The summer camp focuses on getting students to have fun in STEM. While Starbase includes many fun, hands-on activities, it has a richer, deeper focus in STEM learning outcomes. Also, Starbase participants are from complete classes, whereas this STEM camp contains participants who volunteered.

When comparing the two curricula, Saving the Martian had a larger positive effect on student self-efficacy in all four question sub-areas. This is further confirmed when looking at the average initial self-efficacy in Table 3, where MMC could not capitalize on its students' higher potential to learn (lower initial self-efficacy compared to Mars). We hypothesize that this was caused by additional overhead and distractions from making and controlling circuits, even though the activities were designed to reduce cognitive load. In a recent study, Jin et al. also experienced this result in a similar camp using physical computing to teach young

Table 4 The effect size for each survey question, broken into each comparison group. Italicized indicates a p-value of $\leq .\mathbf{05}$, italicized, underlined indicates a p-value of $\leq .\mathbf{01}$, and bolded indicates a p-value of $\leq .\mathbf{001}$.

Cat	Skill	Mars	MMC	Male	Female	NO-STEM	STEM	OUT-SIDE STEM	STEM INST	Star-base
PS	Algorithms	0.400	0.061	0.194	0.306	0.245	0.229	0.269	0.186	0.344
PS	Abstraction	0.215	-0.082	0.101	0.036	0.056	0.089	0.058	0.117	0.153
PS	Control Flow	0.434	0.230	*0.299*	0.427	0.279	0.371	*0.558*	0.173	*0.645*
PS	Data	0.082	0.291	0.166	0.178	0.028	0.251	*0.349*	0.157	*0.387*
PS	Parallel.	0.181	0.037	0.165	0.000	-0.086	0.192	0.203	0.181	0.135
PS	Prob. Decomp.	*0.254*	0.154	*0.270*	0.059	0.090	*0.268*	*0.367*	0.173	0.310
CT	Algorithms	**0.828**	0.370	**0.667**	*0.448*	0.723	**0.538**	**0.702**	0.373	0.878
CT	Abstraction	0.501	**0.625**	**0.513**	0.639	0.362	**0.653**	0.545	**0.769**	0.692
CT	Control Flow	0.480	*0.353*	0.361	*0.574*	0.444	*0.408*	***0.583***	0.232	*0.682*
CT	Data	**0.728**	**0.537**	**0.629**	0.642	0.716	**0.603**	**0.818**	*0.395*	**0.892**
CT	Parallel.	**0.628**	**0.513**	**0.633**	*0.468*	0.706	**0.521**	**0.653**	*0.381*	0.704
CT	Prob. Decomp.	**0.530**	0.196	0.371	0.380	*0.432*	*0.344*	**0.560**	0.111	0.621
CT	Being Incremental and Iterative	0.229	*0.269*	*0.278*	0.189	0.295	*0.224*	0.274	0.169	*0.344*
CT	Questioning	**0.631**	0.083	**0.478**	0.203	0.429	0.339	**0.540**	0.141	**0.752**
CT	Reuse, Remixing, Connecting	*0.248*	0.198	*0.305*	0.054	0.248	0.211	0.024	0.412	0.132
CT	Testing and Debugging	0.091	0.230	0.167	0.143	0.309	0.093	0.126	0.056	0.176

learners how to program[9]. The authors discuss that, while physical computing can be engaging and fun, students become distracted by the physical devices making it "challenging to teach rigorous computing concepts in such a busy environment."

Breaking down the results to specific concepts, the Mars curriculum emphasized algorithmic thinking through sorting algorithms, using CS unplugged to explain sorting before implementing in Scratch. This is shown to be highly effective compared to MMC, which focused on using LEDs as a method to teach algorithms. MMC focused creating and programming parts of each circuit one component at a time. For example, to make a circuit with five LEDs, students first had to make a circuit with only one. Surprisingly, MMC had no significant result for problem decomposition. We were also surprised to see no effect on MMC students' understanding on how programming can be used in their daily lives. The MMC curriculum leveraged physical computing to provide tangible results from programming lights and sensors that could be used at home in practical applications; however, it was unsuccessful in our context.

Our curriculum did not perform well for questions 19-21 across all groups when compared to CT concepts. This shows that CT practices need to be balanced more alongside the other CT skills. Our curriculum focused on CT, with PS as an effect of the process of building circuits and programs. We observed very little gains in self-efficacy related to problem solving, revealing little to no effect in many PS areas. Skills like algorithms and control flow show some relation the effects in CT skill questions, but no discernable pattern was found. If CT was inherently problem solving, we would have expected a correlation between both CT and PS skills. Results between male and female were also interesting, particularly with conditionals in control flow, though sample sizes were too small to make conjectures.

Conclusions

In this paper, we have discussed the application of two curricula applied to a 5th-9th grade STEM outreach program. By adjusting the language from previous work (2015,2016)[3,24] to be more age appropriate and using a smaller Likert scale, we created a survey that was more consistent within student responses and effective at measuring self-efficacy in CT. From reviewing the survey results, we found that using micro controllers as a tool for teaching CT was less effective than a pure computer science related curriculum. Although MMC was effective at fostering improvement in CT skills, our curriculum has room to improve when using physical computing. We may be able to further improve self-efficacy by implementing our MMC curriculum using the Active-Media Design method used by Jin et al.[9] Finally, we have shown that CT framed inside PS is largely decoupled from CT. This may be the initial indicators that support our goal of showing that CT is not PS. We recognize that this relies on the small sample sizes of our experiment, and that larger studies will be needed to fully confirm that goal. Due to limitations of the STEM program, we conducted our research as transparently as possible. We recognize that student interviews (especially when evaluating the relation between PS and CT) and knowledge-based assessments would likely provide more insight into our research questions. We felt implementing knowledge-based assessments would make it feel like a normal classroom and not a summer camp, and using interviews is currently not feasible in the time we have the students. More so, self-efficacy has been shown to be a good predictor of student learning outcomes[13]. One possible area of future improvement is the inclusion of trivia games to collect some knowledge-based

assessment data while still maintaining the spirit of the summer camp atmosphere. Apart from knowledge-based assessments, we also chose not to conduct static analysis on code produced by students. Since many of our activities were heavily scaffolded, most solutions that students produced were identical leaving little information to be gained.

References

1. Bandura, A. (1982). Self-Efficacy Mechanism in Human Agency. *American Pyschologist, 37*(2), 122-147.
2. Basawapatna, A., Repenning, A., & Koh, K. H. (2015). Closing The Cyberlearning Loop. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, (pp. 12-17).
3. 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.*
4. Bell, R. S. (2014). *Low Overhead Methods for Improving Capacity and Outcomes in Computer Science.* Manhattan, KS: Kansas State University.
5. Brennan, K., & Resnick, M. (2012). *Using artifact-based interviews to study the development of computational thinking in interactive media design.* Vancouver, BC, Canada: American Educational Research Association.
6. Google. (2016, April 1). *Computational Thinking Concepts Guide.* Retrieved from Google for Education: https://docs.google.com/document/d/1i0wg-BMG3TdwsShAyH_0Z1xpFnpVcMvpYJceHGWex_c/edit#heading=h.ld02iaxpskpn
7. Green, A. J., & Gilhooly, K. (2005). Problem Solving. In N. Braisby, & A. Gellatly, *Cognitive Pychology.* Oxford: Oxford University Press.
8. Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher, 42*(1), 38-43.
9. Jin, K. H., Haynie, K., & Kearns, G. (2016). Teaching Elementary Students Programming in a Physical Computing Classroom. *Proceedings of the 17th Annual Conference on Information Technology Education - SIGITE '16*, (pp. 85-90).
10. Knörig, A., Wettach, R., & Cohen, J. (2009). Fritzing – A tool for advancing electronic prototyping for designers. *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction - TEI '09* (pp. 351-358). New York, NY, USA: ACM Press.
11. Koh, K. H., Basawapatna, A., Bennett, V., & Reppening, A. (2010). Towards the Automatic Recognition of Computational Thinking for Adaptive Visual Language Learning. *IEEE Symposium on Visual Languages and Human-Centric Computing*, (pp. 59-66).
12. Kuphaldt, T. R. (2014, Feb). *Lessons in Electric Circuits.* Retrieved 2016, from allaboutcircuits: http://www.allaboutcircuits.com/textbook/
13. Lishinski, A., Yadav, A., Good, J., & Enbody, R. (2016). Learning to Program: Gender Differences and Interactive Effects of Students' Motivation, Goals, and Self-Efficacy on Performance. *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)* (pp. 211-220). New York, NY, USA: ACM.
14. Mason, R., & Cooper, G. (2013). Mindrostrms Robots and the Application of Cofnitive Load Theory in Introductor Programming. *Computer Science Education, 23*(4), 296-314.
15. Przybylla, M., & Romeike, R. (2008). Physical Computing and its Scope - Towards a Constructivist Computer Science Curriculum with Physical Computing. *International Journal of Technology, 4*(3), 93-102.
16. Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and Mental Models in Learning to Program. *ACM SIGCSE Bulletin, 36*(3), 171-175.
17. 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.
18. Royal Society. (2012). *Shut Down or Restart? The way Forward for Computing in UK Schools.* The Royal Academy of Engineering. Retrieved from https://royalsociety.org/topics-policy/projects/computing-in-schools/report/
19. Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., . . . Verno, A. (2011). *CSTA K-12 Computer Science Standards.* New York: Association for Computing Machinery. Retrieved March 31, 2016, from http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf
20. Seiter, L., & Foreman, B. (2013). Modeling the Learning Progressions of Computational. *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13*, (pp. 59-66).

21. Snow, E., Haertel, G., Fulkerson, D., Feng, M., & Nichols, P. (2010). *Leveraging Evidence-Centered Assessment Design in Large-Scale and Formative Assessment Practices.* Denver, CO: National Council on Measurement in Education.
22. Sullivan, G. M., & Feinn, R. (2012). Using Effect Size-or Why the P Value is Not Enough. *Journal of Graduate Medical Education, 4*(3), 279-282.
23. Voskoglou, M. G., & Buckley, S. (2012, September). Problem Solving and Computers in a Learning Environment. *Egyptian Computer Science Journal, 36*(4), 28-46.
24. Weese, J. L., Feldhausen, R., & Bean, N. H. (2016). The Impact of STEM Experiences on Student Self-Efficacy in Computational Thinking. *Proceedings of the 123rd American Society for Engineering Education Annual Conference and Exposition* (ASEE 2016). New Orleans, LA, USA.
25. Wilson, A., Hainey, T., & Connolly, T. (2012). Evaluation of Computer Games Developed by Primary School Children to Gauge Understanding of Programming Concepts. *Proceedings of the European Conference on Games Based Learning*, (pp. 549-558).
26. Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33-35.
27. Wing, J. M. (2011, March 6). Research Notebook: Computational Thinking--What and Why? (J. Togyer, Ed.) *The Link Magazine*. Retrieved March 29, 2016, from http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why
28. Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational Thinking in Elementary and Secondary Teacher Education. *ACM Transactions on Computing Education, 14*(1), 1-16.