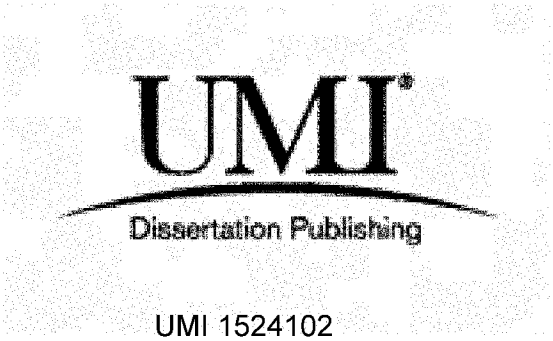
Nerd Alert: Children's Knowledge and Perceptions of Computer Science after a Sixth Grade Computer Programming Course

Erin Elan Mindell
Sarah Lawrence College

Submitted in partial completion of the Master of Arts Degree at Sarah Lawrence College, September 2013



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

As computer science (CS) becomes more prevalent in our culture and the job market, more studies are being conducted on how CS intersects with children's learning. Studies discuss a variety of platforms, age groups, and programs—from daylong workshops to full semester units—to attempt a better understanding of student engagement, enthusiasm, and knowledge of computer science. An element of these studies often not fully explored is children's perceptions of computer science and computer scientists. Through analysis of surveys, interviews, and class observations, this thesis will take a more in-depth look at children's perceptions and knowledge of computer science by observing differences in sixth grade children taking a computer programming course and a control group not taking the course. Findings indicate that there are significant differences between the two groups, including that students who take the programming class obtain a more realistic understanding of CS than those who do not.

keywords: computer programming in a sixth grade classroom, children's perceptions of computer science

Acknowledgments

I am indebted to many wonderful and supportive people without whom this thesis would not exist. Thank you to Carl Barenboim and Candice Reimers, who gave me great advice and suggestions, making this a work of which I can be proud. Thank you to those of you who helped me to calibrate, rework, find test subjects, and make sense of it all when I looked at these pages for too long: Barret Mindell, Ryan Soots, Katie Loewen, Debbie Scher, Gina Gambone, Levi Pierce, and Brendan Ballou. Maggie Johnson and John Atwood, your encouragement got me over the finish line; thank you for your support. Sarah Curtis and Miranda Featherstone, thank you for making it such a joy to get this degree. A special thanks to my editor and friend, Jayanna Roy-Bachman, whose attention to detail gave me the confidence to hand this in. Thanks to Mara Mindell for all of the above as well as knowing exactly how to calm me down and keep me focused during the toughest moments, but especially for giving me the idea to reach out to Aaron Grill. Aaron, your generosity and enthusiasm as a teacher not only made this thesis possible, but also helped me to feel optimistic about the future of computer science for subsequent generations. I can't thank you enough for that. Finally, thank you to my partner, Aemon Cannon, who stopped whatever he was doing to help me get through this beast at its most heinous incarnations. Your love and support are priceless.

Table of Contents

Introduction

Purpose of the Study

The purpose of this study is to detect the role that mandatory computer science (CS) at the early middle school level has on children's overall perception of computer science and how taking a programming course affects their impressions of their aptitude for computer science.

In today's world, the onslaught of technology and computers permeate our existence from moment to moment. As technology becomes more intertwined with our daily lives and more infused in the common vernacular, the clearer the inherently open-ended implications of what we can do with computer science become. Rather than staying a static field, people in a variety of places are constantly exploring new uses of computer science; it has "revolutionized business, education, scholarship, and medicine" (Turkle, 2011, p. 152). Without a doubt, computer science is a pervasive field with a large ability to impact the world. As we become more engrossed in computer technology, the implications for what today's children may benefit from knowing tomorrow are great. In addition to being useful for future career options, computer science is also advantageous for child development by drawing upon "problem-solving and deep thinking skills" (Lamb & Johnson, 2011, p. 64) intrinsic to the discipline.

Yet computer science is a field in which the number of students pursuant is still vastly fewer than the job market for such employment necessitates. In 2010, the US Department of Labor projected a 30% increase in software development jobs over the next ten years, which is 16% higher than the average rating for other jobs within the same

timeframe (Department of Labor, 2013). Because "children's exposure to computing is ubiquitous and centered around the use of computers rather than more fundamental computer science concepts" (Guzdial & Robertson, 2011, p. 13), children do not always know exactly what computer science means. Rather, they merely have a perception of what they think a computer scientist is like and what he — for the perception often includes the image of a male nerd—does (Taub, Ben-Ari, & Armoni, 2009, p. 99).

In order for students to gain interest in this field, they must be exposed to the ins and outs of programming to better understand how the problem solving and logical nature inherent in computer science can be personally interesting. By "inspir[ing] students to study computer science and understand its relevancy to their lives" (Wagner, Gray, Corley, & Wolber, 2012, p. 6) within the classroom, teachers may be able to help break down the preconceived ideas of what computer science is and who is good at it across different backgrounds, and not just in self-selecting groups of children. Additionally, "it has been found that students respond positively to new subjects when teachers and counselors are enthusiastic and knowledgeable about the area" (Munson, Moskal, Harriger, Lauriski-Karriker, & Heersink, 2011, p. 1836). Informed teachers and counselors may help change student attitudes across various disciplines.

Unfortunately, computer programming is new territory for many educators, making it difficult for them to gain confidence and momentum, which might then be passed on to students in the classroom. In fact, "To the degree that programming is found at middle schools at all, it is usually offered as after-school programs" (Repenning, 2012, p. 38), making it difficult for *all* children to have a chance to learn about it. The combination of a lack of teachers who are willing and/or able to address computer

science in the classroom and a dearth in understanding of computer science by students is certainly not alleviating the chasm between the need for computer scientists and the future workforce that will be unable to fulfill those roles.

Hypothesis

My hypothesis is that even with fairly basic hands-on experience, students' perceptions of their own computer science ability will improve significantly. Further, I predict that there will be a correlation between a perception change in computer science, if present, and whether or not students plan to take more courses or obtain a computer science job in the future.

Significance of the Study

As computer science becomes more prevalent in the workforce and the need for computer scientists increases, more studies are being conducted on the intersection between computer science and children to better understand the correlation between what is taught, what is learned, and how that learning affects developmental areas (Munson et al., 2011; Grover & Pea, 2013; Xiao & Carroll, 2007; Werner & Denner, 2009). However, little research is currently focused on children's perceptions of computer science, although such information is sometimes touched upon peripherally (Taub, Ben-Ari, & Armoni, 2009; Wilson, Connolly, Hainey, & Moffat, 2010; Sivilotti & Laugel, 2008; Mason, 2009). My thesis work will begin to address how children's perceptions of computer science is impacted by whether or not they have mandatory, hands-on, programming experience in the classroom. Moreover, this study will look at how the students' perceptions of their own ability — apart from the objective measure of their ability — is related to their propensity to view computer science as a viable future

option, either as a field of study or a career. I will be measuring the subjective impression that one set of children has of computer science after taking a required, hands-on programming course, and comparing that to the subjective impression of children who have not taken such a course. I will also be measuring the reflective ability of the children in the Technology class by comparing the self-assessment of their own understanding to what they actually know about computer science.

Definitions

Public interpretation of what computer science is can range from the academic study of computation, for example, learning logic and problem solving, to practical applications, such as actual programming, to anything computer literacy related, like knowing how to research on the internet or how to use a mouse. For the purposes of this work, I will define computer science as an academic study of computation, which may encompass practical applications but does not include computer literacy. Furthermore, we will assume computer programming is specifically the process of creating, writing, coding, scripting, testing, and/or debugging a computer program.

Within this work, research both past and current will call out programming environments in which children do not have to learn a specific programming language in order to create programs. These environments are: Alice, AgentSheets, AgentCubes, App Inventor for Android, Game Maker, Macromedia's Flash™, Scratch, Stagecast Creator, and ToonTalk. As Scratch was the observed environment, I will spend a little time explaining what it looks like. Scratch is a free, online learning environment created in 2009 by MIT's Media Lab for the purpose of teaching beginners how to program. It is a block-based tool with a puzzle-piece-like interface that allows for a visual programming

experience, where blocks correspond directly to program fragments. One may use the building blocks provided to create unique games, stories, and animations online. It also includes an open-source component, meaning that others can download, view, and change the code of previously created projects. Scratch was created with educators in mind, and includes resources, tools, and an online community to aid in teaching programming to students.

I also make mention of the "Technology class" or sometimes the "Programming class." This refers to the class in which I did my observations, surveys and interviews. I use these interchangeably; "Technology class" is how the school refers to the course for all grades, regardless of what they are studying. My observations took place during the sixth grade programming section of this class. Other relevant definitions will be explained in text.

Delimitations, Limitations, and Assumptions

Delimitations. There are a variety of ways to approach research on children in computer science. My research looks solely at mandatory, in class, computer programming as key factors in measurement. All three of these factors must be taken into consideration in turn.

Mandatory. Research on afterschool or summer workshop programs look at children who — either on their own or through parental encouragement — choose to be involved in computer science. Students in such programs may already have more favorable opinions of computer science, which is why they are willing to participate in computer science outside of school hours. My research is conducted within a mandatory environment in order to control for the self-selection inherent in voluntary programs.

In class. As above, by looking at students who are required to take computer science courses as part of their assessed studies, research is controlled for those who would choose it versus those who would not otherwise take a computer science course.

Computer programming. Many kinds of computer science courses are available for study, including robotics, computer science unplugged—which applies a variety of computational ideas to activities without the use of computers—digital modeling and rendering, and computational thinking. There are many kinds of computer science courses available to students, and I conducted research on only computer programming and will therefore limit conclusions to this specific subject within computer science.

Limitations. This is a study of convenience, and has some inherent limitations. My research was conducted in a private, all boys' preparatory school environment, creating an element of self-selection. Tuition for the school is approximately $40,000 a year, and while the school offers need-based financial assistance, it is unknown how many students in the school are on financial aid or how much financial aid is offered on average or overall. In addition, student learning took place in a separately equipped Technology classroom with a fully functional set of computers—enough for every child to have his own workspace—a Smart Board, a projector, and a teacher formally trained in Computer Science Education.

In creating a control sample survey, I failed to include a question regarding gender. The control sample survey was created to parallel the questions in the Technology class survey, which did not include a question of gender, as all students in the latter group were boys. Therefore, we will not have gender-based results across data sets.

Initially, a goal of this study was to compare surveys from the sixth grade Technology class to those collected from the eleventh and twelfth grade Advanced Placement Computer Science (APCS) class at the same school. The APCS class was comprised of students, some of whom took the sixth grade Technology class, and some who did not. I may have been able to see whether or not a more longitudinal pattern appeared, but, unfortunately, the only APCS students to complete the survey had not taken the sixth grade Technology class. My research is therefore limited to a one time, case-control study. Additionally, while survey and interview questions are based on previous studies, I did not independently test the questions to gauge validity and reliability in different contexts.

Assumptions. All survey, interview, and observation data was collected anonymously and with consent. It is assumed that participants answered all questions truthfully, to the best of their knowledge. However, as previously noted, the Technology class students may have been influenced by the expertise, ease, and enthusiasm of their teacher.

Literature Review

Studies that look at computer science vary in depth, specific subject matter and complexity. For the purposes of my paper, I have limited the scope of research to studies that have come out within approximately ten years. I have done so because the advances in the study of computer science have changed significantly in that time. Additionally, I have narrowed the discussion to studies that look at computer programming in order to best compare to my own case study that looks specifically at programming.

A commonality among the reviewed literature is a focus on a specific programming environment that may be appropriate for children, rather than comparisons among them. Researchers studied a variety of programming tools—Alice (Kelleher & Pausch, 2007; Munson et al., 2011; Rodger, Hayes, Gaetjens, Qin, Nelson & Tucker, 2009), AgentSheets/AgentCubes (Repenning, 2012), App Inventor for Android (Grover & Pea, 2013; Wagner et al., 2013), Game Maker (Herrig & Taranto, 2012), Macromedia's Flash™ (Werner & Denner, 2009; Xiao & Carroll, 2007), Scratch (Maloney et al., 2008; Sivilotti & Laugel, 2008; Wilson et al., 2010), Stagecast Creator (Denner, Werner & Ortiz, 2011; Werner & Denner, 2009), and ToonTalk (Tholander, 2005). All of the reviewed literature found at least limited gains in learning, enthusiasm and/or skills, which suggests that the specific environment taught may not matter as much as some researchers speculate.

Perhaps what is more important is to find a way to give personal meaning to whichever programming environment is chosen. As Wagner et al. (2013) put it, "In order to inspire students to study computer science and understand its relevancy to their lives, educators should identify meaningful learning contexts" (p. 6). In order to insert those contexts, educators must be engaged with the content as well. In fact, Grover (2013) argues that children may "struggle with algorithmic concepts...if they are left to tinker in programming environments, or if the learning is not scaffolded and designed using the right problems and pedagogies." It is necessarily important to anchor individual programming environments with proper instruction to encourage conceptual learning rather than rote memorization and/or trial and error; these latter processes do not help students achieve true computational thinking understanding.

Putting the common theme of individual programming tools aside, I categorized the literature into three main groups: development, pedagogy, and student learning, which I will outline below.

Development

A fairly broad way in which to study children's interaction with computer programming is to focus on what is happening developmentally as children begin to program. Cognitive development and social development are the two areas of child development that arose in the reviewed literature. As would be expected, there is a lot of overlap within these two categories, so while I will categorize based on individual studies, I will also explicitly call out which is being discussed. Studies looked at how students are cognitively and socially developing from two distinct standpoints: those that begin with programming instruction first and those that begin with design conceptions.

Programming first. One way to teach computer programming is to jump right into learning specific functionality. Eight articles looked at what happens when instruction begins with teaching basic concepts. The first five I will discuss all look at social and cognitive gains by describing cooperative learning environments where students are required to work in pairs or in groups.

Hwang, Shadiev, Wang, and Huang (2012) looked at first-time undergraduate students. While each student has his or her own project, they were all provided with ways to provide feedback, share their own code—which was open-sourced so that others could download and modify it—and ask each other for help. In order to track gains in cognitive understanding, investigators viewed each student's code from beginning to end of the assignment, as well as information on what other students interjected, where, and

how. The study found that overall, the group gained cognitive abilities but that the lowest performers struggled both cognitively and socially, as they were unwilling to ask for and/or receive the help with their work.

Werner and Denner (2009) looked at pair programming in middle school in order to understand if social pairings allow girls—who they found traditionally struggle with persistence throughout the debugging process—to engage in problem solving cooperatively, thus affecting cognitive outcomes. They found that "by helping each other learn the debugging process, many of these pairs of middle school girls employed the problem-solving steps that will prepare them for computer science courses" (p. 43). In this way, the social development of learning to work effectively in pairs positively influenced cognitive development of persistence and trial and error to reach a solution. However, the researchers also found that the social interaction could have undesired negative results; sometimes a girl would steer her partner away from the more direct route to a solution that she was on and into an entirely different direction, making it much harder to find a solution, if one was found at all.

In a follow-up study, Denner et al. (2011) took a closer look at the cognitive developments through these social interactions, as they continued to research pair programming. In this study, they had girls go through the programming tutorials for Stagecast Creator in paired partners and followed up by giving in-class instruction on the same elements learned. Students were then given a series of different kinds of games to create. Denner et al. (2012) studied each of the games by looking at usability, defined as how well a player can engage with the game, and increased complexity. The latter component was written into the assignment, as each subsequent game included more

level and stage requirements. The researchers concluded that this approach is "promising" (p. 245), but also noted that students had difficulty with planning the design, as they were not given direct instruction on how to do so.

Sivilotti and Laugel (2008) began with a 15-minute lecture on understanding computer science to distinguish the tools of computer science from the application of computer science in practice. They then gave a five-minute tutorial on Scratch, and set middle school children to work on a series of increasingly difficult tasks to work through. Pairs were then given an assignment with specific design element requirements—they were allowed to add any other elements in addition to the requirements. These researchers found a significant cognitive gain in children's learning. However, a limitation of this study is that the results are based solely on participant surveys that asked how much they *felt* they learned rather than studying actual content created.

Wilson et al. (2010) also looked at pairs of students working with Scratch. They worked within traditional school settings and instructed two different groups of elementary school children for one-hour sessions over the course of eight weeks. Both instructors began with programming basics, teaching children a series of skills. In order to measure gains in cognitive abilities, Wilson et al. (2010) tested the children at week three and at week nine, the latter being the week after course completion. They tested on the same set of skills to trace any improvement over time. They did find a rise in test scores over time with one class, but ran into trouble in quantifying scores for the second, as far fewer children were in class to take the second test, due to an upcoming holiday break. They were not, therefore, able to draw as many conclusions as hoped.

Two additional studies that began with teaching programming looked primarily at cognitive development. While both of these studies allowed for interactions between students, they did not require it as with the previous five.

Rodger et al. (2009) worked with students during a week long summer camp using a world creation environment. The workshop was broken up into instructional periods followed by unstructured work time in which participants were allowed to build their own Alice world in whatever way they wanted. This format enabled children to add more complex functions to their projects over the course of the week, while building from basic to more difficult concepts over time. Rodger et al. (2009) then studied each child's world to see what objects were employed and what programming concepts were enabled. They found that cognitively, the majority of campers were able to master at least the basic concepts provided in the instruction and incorporate such ideas into their individual designs. Additionally, they found campers were always asking for more time to work on their individual project, suggesting that more gains in cognitive abilities may have occurred given more time.

Munson et al. (2011) introduced the same environment—Alice—to teach programming basics to a group of high school students during a weeklong camp. They then used a concept exam to test on concrete concepts understood in order to see if any gains were made in programming knowledge from before the camp started to after its conclusion. They found that "Through the Concept Exam, there was evidence to support that...students' content knowledge improved as a result of the workshop" (p. 1847). While there is definite variety in how programming instructions are handled in

dissemination, these studies all look at what happens when it comes before design, if in fact formal design is part of the instruction at all.

Design first. The next set of studies traced cognitive developmental gains by introducing design concepts before programming basics. As Lee, Kolodner & Goel (2011) state, "Design is a cognitive activity" (p. 1). When students think through the design of a game prior to developing it, they must define a problem—what the game will look like—and think about avenues to solve that problem—how to make the game look a certain way—prior to touching a programming environment. Moreover, by thinking through how other children will react to the game and incorporating their feedback into a prototype (Herrig & Taranto, 2012), they are building social development by incorporating how another person may feel into their own actions. Socially and cognitively, "When students design a game for usability, they include features that engage and motivate the user, clear instructions on how to play the game, and play experience that is free of defects" (Denner et al., 2012, p. 241) thus enabling them to make cognitive gains through social interactions.

Two studies looked at implementation of design prior to introducing programming basics. First, Xiao, & Carroll (2007) look at an informal collaboration between students and teachers to design and build a website for health students learning in the online space. Using Moodle—an online course-building platform—groups began by designing what the health course should look like cooperatively prior to learning how to build it. Student participants had varied computer science experience, ranging from beginner to somewhat savvy. While the study does not look at cognitive gains, it gives insight into social development. "By working on real problems, students interpret the

process based on their own understanding of complex social phenomenon. Therefore students may hold different views of the problem and have different approaches to solving the problems" (p. 31). In order to work through these differences, students must cooperate and work on their social development to reach a common decision. The investigators found overwhelming success in social development for these high school students. While the group was meant to complete just one project, they decided—cooperatively—to work on other projects together as well. Though it is not explicitly stated, the possibility exists that these children were able to help each other through Vygotsky's zone of proximal development (1978). Defined, the zone of proximal development is "the distance between the actual developmental level [of a child] as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with capable peers" (Vygotsky, 1978, p. 86). Put another way, a child may be capable of accomplishing more with guided help than on her own.

The second study that looked at design first was Kelleher and Pausch's (2007) study that led to the creation of Storytelling Alice. The researchers worked through design elements that children might want to work with prior to building an environment for them. By involving children in the initial process, they were able to come up with a programming environment that subsequent children found engaging. Working with the second set of children, the investigators were able to talk through design elements first and then give them freedom to learn and explore on their own. They measured cognitive gains by observing the kinds of elements implemented into each project. What they

found was that all of them made at least some cognitive gains (by creating sequential programs), and 87% created more complex programs, displaying higher cognitive gains.

Both design studies had favorable outcomes. A possible problem with presenting design first, however, is that it is more time consuming, and may lose student interest before even introducing programming into the equation. Additionally, it would take more planning on behalf of instructors and if it does not go as planned, it would be difficult to change the course of action to accommodate. This may account for why there are few such studies—and both represented here are from more than five years ago—that begin with design and look at programming after.

Programming and design intertwined. Finally (and most recently), Grover and Pea (2013) looked at a hybrid of programming first and design first through their discourse-intensive model of teaching programming to middle school children using App Inventor—a programming environment where one can create her own web application. They began by introducing some programming basics, but actively encouraged students to interject with questions related to their own phone app design ideas. From these discussions, they were able to introduce concepts and terms organically over the course of the day as they became relevant to student projects. By doing so, investigators not only found gains in student abilities, but they were also able to ground these skills in relevant lessons, which laid the groundwork for skills learned to actually stick.

Pedagogy

Grover (2013) states, "Not all coding experiences are equal, and what kids take away in terms of thinking and problem solving skills depends on the depth of those

experiences. It requires employing pedagogies and curricula that engage intentional thinking around solutions BEFORE any coding happens." Pedagogy refers to the way in which a subject—in this case computer programming—is taught in an educational setting. It looks at the learning environment as a whole—who is teaching and how, whether students are instructed to work together or alone, how the classroom is set up, etc.—as well as specific theoretical designs for implementation in the classroom. Reviewed literature fell into five distinguishable pedagogical categories, with some overlap.

Discourse. Discourse pedagogy lets student questions guide the curriculum. An experienced teacher sets a general framework for students by starting with lectures and/or instructing to the class as a whole. Students are encouraged to ask "what if" questions as they have them and the instructor will adjust the lecture to address and build on these questions. The instructor may also pose hypothetical questions to guide instruction in a certain direction. Two reviewed studies looked at discourse related pedagogy, using two different programming tools (App Inventor and Scratch). One looked at it from a group perspective where all students were involved in the same discussion (Grover & Pea, 2013). The other provided very little upfront guidance and let students ask questions and engage discussion as needed (Maloney et al., 2008). Both studies found that students learned fairly sophisticated programming skills from employment of discourse intensive pedagogy.

Constructionism. Four studies looked at constructivist pedagogy. Constructionism has creation-based learning at its core. Rather than outline a series of problems individually for students to work through, constructionism harnesses the theory

that real-world creation gives students a more engaged interaction with the material. They are able to leave a course with a tangible final project that they can share out with others. This provides a reason to complete coursework rather than just because it is an assignment. Two studies evaluated constructionism techniques for middle school (Denner et al., 2011; Xiao & Carroll, 2007) and one looked at elementary school children (Tholander, 2005). A fourth article exclusively discussed implementation of constructivist programming in the middle school classroom under the auspices that it provides "problem-based learning experience" (p. 28) to help "foster a feeling of independence while at the same time encouraging peer collaboration" (Herrig & Taranto, 2012, p. 31). While Denner et al. (2012) and Tholander (2005) found that children did in fact complete the creation of games through programming, the games were often not very sophisticated. Moreover, in a setting that has minimal instruction for game creation, Denner et al. (2012) discovered that several children had trouble with even some more basic concepts suggesting that it may be better to incorporate higher levels of teacher instruction when utilizing constructionism.

Reciprocal. In some studies, an instructor who does not have a computer science background implements computer programming into the classroom and works with students as they are learning as well. This is what I am referring to when I say "reciprocal pedagogy." I am not referring to "reciprocal teaching," which is a four-pronged strategy used with reading teachers. For the purposes of this literature review, reciprocal pedagogy is in reference to the idea that having an inexperienced teacher at the helm allows students to do more exploration and experimentation, as they cannot always rely on the teacher to provide answers when they get stuck. In addition, it allows students

to be the experts on occasion, thus bolstering their confidence in programming. Four studies looked at reciprocal pedagogy in some form or another.

Wilson et al. (2010) brought researchers into classrooms at two different schools to lead programming instruction while the class teacher—who had no previous experience in computer science—played the role of assistant teacher. However, as the main goal of this particular study was to evaluate the Scratch software for effectively teaching young children about programming, there is almost no discussion of how the reciprocal framework played out between the main classroom teacher and the students.

Four of the studies allude to reciprocal learning without including much about it in their findings. Munson et al. (2011) and Rodger et al. (2009) do not go into the classroom and play the role of head instructor. Rather, both studies look at summer workshops where teachers learn first and then allow for reciprocal work with students later on. Munson et al. (2011) found that while students received feedback and got help, teachers "often served as learning peers rather than mentors to the students during the workshop" (p. 1847). The study found that "there is evidence to support that student attitudes improved with respect to the information technology field from the beginning to the end of the summer workshop (ibid)." The researchers also found "evidence to support that both teachers' and students' content knowledge improved as a result of the workshop" (ibid). However, the study does not address the students' assessment of their own ability, just a generic impression of the field in general. Rodger et al. (2009) study concludes with plans for follow-up once teachers returned to the classroom, however, I was unable to locate supplemental research that spoke to if/how the reciprocal learning worked once the school year began.

Xiao & Carroll (2007) discuss the idea of middle school teachers as both facilitators and learners as new technologies emerge, and conclude that this kind of reciprocal learning is particularly beneficial for teachers to stay competent in teaching the most updated lessons. This study, however, looks exclusively at an informal online environment as a "complement" (p. 33) to the classroom, rather than a traditional classroom setting; it is unclear how the model would translate into a formal learning environment.

Finally, Maloney et al. (2008) bring in an undergraduate and graduate support team to play a mentoring role for the children in the "clubhouse" who are learning to use Scratch (2008). Because "mentors had little or no experience programming and were new to Scratch," the model "empowered youth, allowing them to sometimes switch roles and teach a mentor something new in Scratch" (p. 3). This provided a "more equitable relationship that turned both mentees and mentors into learners" (p. 5). While this study did not look at classroom teaching, it did find that children were able to demonstrate many programming concepts despite the lack of formal lesson plans and experienced teachers. These studies suggest that reciprocal learning may have at least some pedagogical benefits in computer programming, which becomes very important when considering the dearth of teachers who are already skilled in—or indeed have any previous exposure to—computer programming.

Peer assistance. Peer assistance goes beyond allowing students to work with each other and actually requires it. Assignments are given to two or more students, and they must work together to find the solution. The idea behind peer assistance is similar to that of Vygotsky's (1978) zone of proximal development, in which children are able to

do more working together than each would be able to do on his own. Three studies discussed peer assistance in some detail.

Hwang et al. look at thirteen undergraduate students with no previous computer programming experience. They gave in-class lectures and required students to work cooperatively in a variety of ways—dividing up tasks, providing feedback for each other's code, writing code in relay, and open-sourcing code so that students could browse and build on each other's work. They found that this format "motivate[s] students learning and active participation" (2012, p. 151). Additionally, they found that most students perceived cooperative learning to be useful. This is a promising technique, however, as this study looks at undergraduates in Taiwan, we must be careful when trying to relate it directly to middle school children.

Sivilotti and Laugel (2008) also started off with a detailed introductory lecture for middle school students. Then, working in pairs, students are given a series of tasks to complete. These tasks get progressively more difficult and students must work together to find the proper solutions. This study found success in improving children's interest in programming as well as computer science, which may be related to the peer pairing, but a correlation is not drawn.

The third study is more in depth on both how peer assistance translates to the middle school level as well as addresses how peer pairing works to motivate children to persist (Werner & Denner, 2009). Through the presentation of four transcripts, results show the positive and negative ways middle school girls interact while programming together. Additionally, they found that children gained confidence from their experience working together, although they do not compare these students to students working alone.

Peer assistance seems to have some positive effects on confidence and learning to program. However for students who prefer to work alone, this forced interaction may be frustrating. Even for students who may not mind working in pairs, having ample time to work alone may be necessary in order for them to obtain full enjoyment of computer programming (Wilson et al., 2010). In addition, students who struggle the most with concepts and tasks may fall behind as their peers push ahead and no one is left on the same level to help them (Hwang et al., 2012).

Teacher scaffolding. Scaffolding is another Vygotskian concept whereby the instructor provides purposeful guidance to allow students to work through problems and come to solutions on their own. Two articles discussed scaffolding directly (Repenning, 2012; Tholander, 2005). The Repenning article is not a study, although it references previous studies that "systematically investigate the interaction of pedagogical approaches and motivational levels" to broaden student interest and participation in computer programming. This article stresses the importance of direct instruction and scaffolding "across different school contexts, gender, and ethnicity" (p.40). This lends support to scaffolding as pedagogy to implement. Repenning stresses that "guided discovery" is an equally favorable motivational tool for both boys and girls. However, this article does not discuss how much students are actually learning. Tholander looks at the relationship between instructor and student and found a central source for development within the programming framework arose from scaffolding (2005). While this is also a positive statement, Tholander also discusses the difficulty in finding common understanding between student and tutor; without this common perspective, students became easily confused.

Student Learning

While pedagogy views how computer programming is *taught*, it does not necessarily address what students are learning. Certainly, programming is a skill in and of itself that is learned by children as young as elementary school-aged (Wilson et al., 2010; Maloney et al., 2008). But one of the reasons it is so important to implement computer science into schools is because of the intersubjectivity inherent therein. Students learn skills that are relevant in other areas of schooling—as well as out in the world—regardless of whether or not they end up choosing computer science as a field of study in college or professionally. Different studies noted a few examples of this.

First, a few studies found that the use of arithmetic within programming environments provided contextualization to help improve students' understanding of math and numbers (Fletcher & Lu, 2009; Maloney et al., 2008; Wilson et al., 2010). This provides the meaningful context needed to apply mathematical skills rather than just learn them through rote memorization such as is taught in many math courses.

A second area of conversation in the articles—and perhaps the easiest place to see how programming strengthens other subjects—is in the area of computational thinking. Several studies looked directly at computational thinking as a way for programming to heighten other subjects (Denner et al., 2011; Fletcher & Lu, 2009; Grover & Pea, 2013; Rodger et al., 2009; Repenning, 2012; Sivilotti & Laugel, 2008; Werner & Denner, 2009). Computational thinking (CT) is a term coined by computer scientist Seymour Papert in the 1990s. It refers to a set of problem solving skills and techniques that individuals use to write programs that create the base of common computer applications. CT relies upon the ability to understand human behavior as well as the ability to create

and adjust systems. Computational thinking can be applied to other subjects, such as math, science, language arts, social studies, art, and history. Rodger et al. give some good examples of how CT intersects with these other fields of study. For instance, a programming environment can be used as an inquiry-based design tool to serve as the base for projects such as "a food web with items such as producers, consumers and decomposers, [where designers] have to investigate the impact of humans and/or pesticides in their system" (2009, p. 272-273). Introducing children to these foundational concepts gives children the ability to insert a computational way of thinking into other areas of study.

Related to the idea of computational thinking, the most often referenced programming learning concept by far was problem solving (Denner et al., 2011; Frost et al., 2009; Grover & Pea, 2013; Herrig & Taranto, 2012; Hwang et al., 2012; Tholander, 2005; Werner & Denner, 2009; Wilson et al., 2010; Xiao & Carroll, 2007). This is not surprising, as "problem-solving" has generic enough positive connotations that can be applied to any field of study while still being a key component of computer programming. In their outline of basic level objectives for K-12 computer science, Frost et al. (2009) make it clear that problem solving is a creative medium and a basic component of programming; the other studies seem to agree with this ground-level tenet.

To understand how problem solving may or may not be working, several studies look at the *process* of programming. Denner et al. (2011) asked children to create a series of different types of games, while Grover & Pea (2013) and Herrig & Taranto (2012) let the environment guide the questions and the children came up with the solutions. Hwang et al. (2012) looked at different stages in an online cooperative

learning environment. All four studies then looked at what programming elements were incorporated into their games to identify various ways and complexities to come up with solutions. Tholander (2005) took children through a series of tasks independently and then set them in an unstructured programming environment to see how successfully they incorporated the simple tasks into a complicated solution to create interesting games. Similarly, other studies focused on how problem solving skills are gained by examining programs created and looking at what programming elements were incorporated (Wagner et al., 2013; Wilson et al. 2010), although they do so without parameters set out first. One study that looked at how problem solving is being used during programming experiences relied on observations and recorded sessions to pinpoint how students work through problems to get to a solution (Werner & Denner, 2009). This was a different approach that succeeded in obtaining more in depth knowledge of the pair programming experience, but it did not formally code problem solving usage.

We are starting to see some research studying the intersection of child development and computer programming; in this literature review, I have outlined three areas in which this research could be categorized: development, pedagogy, and student learning. However, current research is still somewhat sparse, and there are still some holes that need to be addressed. For instance, student perception has not been systematically dealt with in previous studies.

In my thesis, I will talk about this area of cognitive development that current research does not fully address: the effect of hands-on computer programming on student assessment of their own ability, i.e., self-efficacy. It was Bandura (1977) who developed a framework for modeling the way that self-efficacy—that is, perception of one's own

ability—affects performance. This framework was found to be predictive in a series of task-based tests, concluding that

> self-efficacy was a uniformly accurate predictor of performance on tasks varying in difficulty with different threats regardless of whether the changes in self-efficacy were produced through performance accomplishments (89% congruence) or by vicarious experience alone (86% congruence). (p. 16)

"Performance accomplishments (p. 5)," which refers to personal experience in performing a task, and "vicarious experience (p. 7)," wherein a child's experience is based on watching another perform a task, are the strongest two of Bandura's four principle sources of self-efficacy. While the power of self-efficacy as a means of predicting performance is equally strong in either case, personal experience was shown to be a stronger *generator* of self-efficacy. Bandura writes, "experiences based on performance accomplishments produced higher, more generalized, and stronger efficacy expectations than did vicarious experience (p. 15)." This framework plays an important part in my research, as I address the enhancement of children's perceptions of self-ability via hands-on experience.

There are a few studies that talk about children's perception of computer science indirectly (Kelleher & Pausch, 2007; Mason, 2009, Rodger et al., 2009, Werner & Denner, 2009; Wilson, et al., 2010). Kelleher and Pausch ask this hypothetical question:

> If you walk into a classroom of middle school girls in the U.S. and ask how many of them want to learn to program, few hands are likely to go up. If you ask how

> many of them want to learn to make animated movies
> like those from Pixar and Dreamworks, you are likely to
> get a very different and very positive response. (p. 60)

This hints at the idea of perception by speculating about the girls' desire to learn based on how they feel about the word "program" specifically versus what they would be doing in the chosen programming environment. Similarly, Rodger et al. mention that students "have little idea of careers in computer science other than the media images of the non-interactive geek (most likely male) sitting in his cubicle all day" (2009, p. 271). Both discussions end there without further elaborating on the subject of perception. Werner & Denner go a little further in their introduction by discussing the overall perception of IT work as socially isolating and competitive as a turnoff for girls (2009), but what they actually cite are previous studies on *women's* understanding of the field rather than *children's*. Arguably, there is a disconnect between what a child understands and what an adult thinks, even if it is a young adult of college age.

Mason looks at Bandura's 1986 explanation for gains in cognitive development as linking to one's perception of his or her self-efficacy, but she does not directly research the question of self-efficacy and perception in the design measurements of her thesis, rather she limits her research to student interest and motivation (2009).

Wilson et al. (2010) evaluate elementary school children's enjoyment of computer programming classes based on a three-point visual scale (cartoon faces are used to represent the children's feelings) and relate those feelings to their programming abilities. This tentatively links feeling to self-efficacy, but again does not elaborate further.

While there is not much written indirectly on perception, with regards to computer science, even fewer studies look at perception from a more direct standpoint (Drobnis, 2010; Herling, 2011; Hwang et al., 2012; Munson et al., 2011; Sheehan, 2003). Those that do, generally focus on perceptions of college age students (Herling; Hwang, et al., 2012) or elementary school children (Sheehan, 2003). Two studies looked at high school students' perception of computer science (Drobnis, 2010). The first study had as its focus high school girls' attitudes about computer science based on who was present in their Introductory to computer science course (female only versus mixed gender) in a math and science magnet high school where all of the students at the school are required to take the Introduction to Computer Science course. While it is interesting to look at a group of students who are all taking the same course for comparisons, the group dynamic is still skewed, as students must have an aptitude for math and science to be accepted into the magnet program. The students in the case study may already have different viewpoints on computer science than those in the general student population. In addition, the research presented in Drobnis' (2010) work indicates that self efficacy between the groups of introductory students did not vary greatly.

The second study (Munson et al., 2011) looked at the change in high school students' perception of computer science, specifically in regards to information technology. Students took pre- and post- workshop surveys to capture any attitude changes. Investigators found statistically significant changes (in the positive direction only). However, discussion group conversations suggested that attendees believed that "students who would attend this workshop were consistent with common stereotypes, e.g., nerdy, boring. Upon completion of the workshop, they were surprised that the majority

of the students attending were similar to themselves" (p. 1846). This makes it unclear to determine perception of exactly what—ideas of information technology, the people who are involved in computer science, students who may be interested in computer science or some combination of these—is changing.

Changing perceptions in either student category, whether elementary school or high school/college, may be problematic. College-aged students may be interested in taking one computer programming course, but they are unlikely to change their major to a computer science focus or even continue to take more courses after doing so (Rodger et al., 2009). Elementary school children may be too young to actually do much original computer programming; they will merely perform routine actions from instruction or previous experience (Tholander, 2005). This may make it difficult for them to fully comprehend the abstract concepts behind what computer science entails; without a concrete understanding of computer programming, their perceptions of whether or not they feel positively about computer science could easily waver.

> As Drobnis notes in her dissertation,
>
> in a study about computer attitudes, it was found that eleven and twelve year olds perceive interests in computers as a positive trait. However, by age fifteen, girls who are considered computer enthusiasts are viewed by their male and female peers as being lonely individuals who have entered into a "male domain." (2010, p. 13)

It is clearly important to reach students before this attitude changes. As we have seen within the body of previous studies, much of the current research is centered around after

school or summer camp sessions. The research I present in this thesis will differ by looking directly at children who have been formally taught computer programming within the classroom on a mandatory basis, rather than a voluntary one. The current body of literature focuses on take-away learnings from children's computer programming experiences; what they are able to do at the end of the day is often the main focus of the studies. My research will slightly shift this focus by trying to understand children's cognitive abilities to comprehend tangibly what computer science and computer programming are—as it relates to perceptions—rather than focus on what they have created. In addition, I will build upon current research by focusing on the perceptions sixth grade children have post-programming experiences and compare them to those who have not been formally taught programming within the school setting. I will be looking both at their ideas and biases of computer science as a field of study as well as their self-confidence in their own abilities.

Methods

Subjects

The main participants of this study were 15 sixth grade boys in a private all boys' preparatory school in the New York City area. Research took place in the mandatory Technology classroom, in which students participate in class three sessions in a two-week cycle. All children worked from separate desktop computer consoles and had a mixture of direct instruction, individual work time, collaborative and/or free work time, and sharing involved in most class sessions. All 15 children had been at the school the previous year, where they were introduced to the Technology course—along with very

basic programming—through a robotics course. For this study, students were observed using Scratch over a five-month period, during which they used the tool to create and share original video games during class. A follow-up assignment asked students to "remix" (e.g. change elements of programming to) a game found in the open-source Scratch community to present to the class. The children ranged in age from eleven to twelve years old.

The control group for this study was comprised of 20 sixth grade girls and boys from two different schools, one located in the New York City area, and the other in Portland, Oregon. The New York City school is a mostly Black (77%) public K-8 school wherein approximately 80% of students are eligible for free or reduced-cost lunches; enrollment for the school includes nearly 700 students. Study participants were anonymous volunteers from an after-school program in the school building. The Portland, Oregon school is a focus option public school with a mostly White (77%) student population of approximately 460 students. The sixth to eighth grade middle school has about 25% of its student population qualifying for free or reduced lunches. Study participants were anonymous volunteers from a math class, during school hours. Children in the control were all in sixth grade, and none of them had previously taken a programming course at school, as no Technology courses are offered in either school. The children ranged in age from 11 to 13 years of age.

Instrumentation

Observations. I observed the Technology classroom over the course of five months. The children were aware of my presence in the classroom as an observer, but I did not participate in class discussion or by answering questions. Due to the volume level of the classroom, I did not attempt to record in class. Rather, I took notes while class was in session and transcribed them afterward.

Surveys. I provided a survey to both children in the control group as well as the Technology class. Both surveys were based on questionnaires developed by a research study aimed at getting Israeli high school girls more excited about studying Computer Science in school (Eidelman, et al., 2011). The surveys were comprised of multiple choice (e.g. *Do you know how to do computer programming? Yes, No, or Unsure)*, sliding scale (e.g. *I would like to study computer science in high school*, where 1 is *No*, 2 is in the middle without specific demarcation, and 3 is *Yes*), and checkboxes, where students picked as many statements they believed were true from a group of seven to nine choices (e.g. *Computer science is...Fun, Boring, Interesting*, etc. See Appendix A). While the surveys differed slightly from Technology classroom to control group, they were designed to obtain information about student perception of computer science.

Student interviews. Interviews were conducted one-to-one in a subsection of the cafeteria. Each child was asked if he was willing to participate in an anonymous interview and gave verbal consent prior to moving to the separate area. The additional space was used both because it was quieter and in order to keep children from overhearing each other's answers. All children whom I asked to interview consented, giving a total of 13 students interviewed over the two class periods. The other two

children were not in attendance during the classes I was present. Interview questions were based on the survey described above, which had already been taken. The purpose of the student interviews was to compare measurements of children's perceptions (e.g. *I know what computer science is* on a three point sliding scale of *Yes* to *No*) to stated knowledge (e.g. In answer to the question: *What is computer science?)*. There were nine questions total on the survey, though I did ask some children an additional two follow-up questions. I stopped asking one of the original nine questions (*What's your favorite thing you've done in this class?*) a few interviews in after learning that it was irrelevant, given that the entire unit was based on one overarching theme. In one interview, I did not ask one of the questions (*Do you like this class?*), and in one more interview I did not ask the follow-up (*Why/Why not?*). This was due to the time constraints of each class session. Each interview took approximately10 to 15 minutes to perform.

Teacher interview. I interviewed the sixth grade Technology teacher—who is also the head of Technology at the school—and asked a total of seven questions. The purpose of the interview was to understand the pedagogical background and evolution of the sixth grade course as it has taken shape over the past nine years he has taught it in this institution. An additional purpose to the interview was to find out what the sixth grade students are taught about computer science as a discipline (e.g. *Do you define computer science for your sixth grade students?*), which students had taken Robotics the year before—all of them—and whether or not the sixth grade Technology class is required, which it is.

Procedures

Surveys were collected via an anonymized online Google Doc survey. Students were given an anonymized survey administered in the classroom via Google Docs. Questions were chosen to measure the students' competence of their knowledge of computer science, understand their feelings about computer science, and parse their feelings about computer science as a future option. All data were collected and stored in a private, anonymized spreadsheet. Some survey questions—such as "I think computer science is...nerdy" and "I think computer science is for nerds" were purposely duplicated to verify consistent attention of the test subjects. Findings confirmed consistency.

Information collected from the student interviews was transcribed into a database verbatim. With the help of a focus group, I arrived at a rubric for coding the short answer, free-form responses objectively. I then provided the data and rubric to two impartial volunteers to calibrate and receive feedback. Based on information collected from their feedback, I adjusted the rubric for clarity around the definitions of computer science and computer programming and age appropriate understanding of these concepts for a sixth grade child. I then sent the revised rubric to an additional six reviewers for scoring. For each point in the rubric, feedback from the group of reviewers was only factored in if it met a standard of statistical significance, unless otherwise noted.

Information from the teacher interview provided background on students in the sixth grade Technology class. I did not code or use it for data analysis, but it may serve as background for assumptions in the discussion section.

Statistical Analysis

Surveys for the Technology class and the general population groups were conducted using Google Forms that collect data into a private spreadsheet. Data were organized into 34 discrete variables corresponding to the 34 questions on the survey. Summary values (mean, standard deviation) were computed with spreadsheet functions. Using SPSS I applied Welch's independent variable t-test to each variable pair, selecting for further analyses only those pairs where the p value was under 0.1 (i.e., less than a 10% chance that the difference in the samples could be accounted for by noise).

Prompt	Control μ	Exp. μ	p (Welch's t-test)
I know what computer science is.	1.85	2.214	0.062
I know ways that computer science is used in the world.	1.75	2.857	0
Would like to have conversations with f&f about CS.	1.95	1.5	0.0977
I think computer science is problem solving	1.2	1.857	0.0452
I think computer science is frustrating	1.2	1.857	0.0452
I have met people that use computer science in their jobs.	1.5	2.143	0.0692
I think computer science is boring.	1.3	1	0.0828
My school gives me the chance to learn about computer science.	1.4	3	0

Figure 1

Also using SPSS, I computed bivariate Pearson's correlation and accompanying *p* values for all 342/2unique pairs of variables and separated out those correlations with significant *p* values for further inspection. I repeated this procedure for each data set in

isolation (the Technology group and the control) and also for a combined data-set with an additional, synthesized binary variable "Taken computer science" which allowed me to examine correlation between attendance in Technology and any of the other variables.

Additionally, in order to investigate the delta between relationships in the control and relationships in the experiment, I generated a report that includes all control-only, experiment-only and experiment-and-control correlations that met my criteria for significance ($p < 0.05$). Along with each correlation I included a standardized interpretation of the Pearson's r score (very strong positive, moderately strong negative, etc.) ("Pearson's r Correlation – A Rule of Thumb," n.d.) to allow for easy comparison.

In order to extract quantitative results from the short answer responses I created a tri-valued rubric for each question (see Appendix C). As described in *Procedures*, volunteers were used to assign values for each child's responses. I computed summary statistics for each tri-valued variable and arrived at a 95% confidence interval summarizing the focus group's opinion for each question, for each child, and also for the focus group's opinion for each question averaged over all children. For example, I found that the focus group would agree with 95% certainty that the average value for "Do you like the class?" was between 1.3 and 2 on a 0,1,2 scale—indicating that the coders felt the students *did* enthusiastically enjoy the class (see appendix for legends of 0,1,2 scales).

Results

Calibration

As mentioned in *Procedures*, I intentionally inserted some duplication in the survey to try to capture whether the students were being consistent in their answers. Pairs

of questions such as "I think computer science is for nerds" and "I think computer science is...nerdy" were designed to catch points where students were losing attention. However, we found that answers to these questions were highly correlated, hinting that the students were answering the questions carefully and consistently (Figure 2).

Questions	Mean	Std.Dev
I think computer science is nerdy	1.294	0.708
Computer science is for nerds.	1.294	0.708
I think computer science is boring, 1	1.176	0.567
I think computer science is boring, 2	1.118	0.471

Figure 2

Analysis of student perception. Data shows that "I know what computer science is" and "I know ways that computer science is used in the world" are both significantly higher in the experiment (see figure 1) than the controls. This shows that these students have confidence in their own knowledge of what computer science is. Furthermore, higher scores in the experiment group were present for "I think computer science is problem solving," and "I have met people that do computer science in their jobs," indicating a possible higher level of engagement with the subject because it shows that students who have met computer scientists may be able to make the connection between the jobs held and the problem solving nature of such positions. Students in the experiment group were also *more* likely to respond that computer science was frustrating and *less* likely to respond that it is boring, both of which could be explained by the hands-on experience of the experiment group.

In investigating the potential effects of these perceptions, I found that children who responded that computer science is "fun" appear to be more likely to pursue

computer science in the future—planning to take classes and/or have a computer science related job—than those who replied that it is "interesting." This is supported by strong correlation (see Figure 3) between the responses to "I think computer science is fun" and the variables "I am likely to take computer science classes" and "I would like to study computer science in high school," versus the more tepid correlation between "I think computer science is interesting" and "I have a good understanding of computer science." Anecdotally, students in the experiment group who responded to the short answer portion of the study used the word "fun" many times in describing their experiences with the Technology course. Concretely, the short answer portion found that these students enjoyed the class overall and in many cases exhibited great enthusiasm about the experience (see Appendix C).

Questions	Pearson r	p value
I am likely to take computer science classes. vs I think computer science is fun	0.554	0.001
I would like to study computer science in high school. vs I think computer science is fun	0.354	0.04
I have a good understanding of computer science. vs I think Computer Science is interesting.	0.274	0.117

Figure 3

Analysis of student knowledge. In the short answer portion of the study I attempted to capture a holistic picture of whether the Technology students attained a real understanding of what computer science and programming actually mean. The results, coded by a group of volunteers, indicate that the students did indeed gain a level of understanding, with a bias toward a more concrete understanding of computer science centered on application (programming). In response to the "what is computer science"

question, the coded responses indicated, with 95% confidence, that the students' responses show a substantial understanding of computer science, ranging from more vague statements that computer science "is for making computers do things" to some more concrete statements about computer science being the application of programming to make computers do *specific* things. The responses to "what is programming" support the general trend toward concreteness where we see with high confidence that student answers indicate a specific understanding of programming as procedure for creating concrete artifacts such as "websites," "applications," "apps," etc. This makes sense, as programming is the main activity in the class, but it is interesting to see that—without being given a concrete definition by the teacher —the students gather that what they are doing in the class with the simplified, visual Scratch tool is analogous to what working "programmers" do in their jobs.

Perceptions of computer science and socialization. I asked students whether they agreed that "people who work in computer science do a lot of work by themselves, and I don't want to do that." Responses were positively correlated with responses to "I think there are many jobs in computer science that I'd like," as well as with "I think computer science is for nerds," and negatively correlated with "some day I'd like to have a job in computer science" (Figure 5). This seems to indicate that perceptions of the isolation and antisocial behavior are partly to blame for some students not viewing computer science as a viable career path. A possible solution is hinted at by the positive correlation between "I think computer science is something I could do with my friends" and "I have taken computer science before"—perhaps exposure to computer science is required to overcome the stereotypes.

Questions	Pearson r	p value
I don't think there are a lot of jobs in cs i'd like. vs I think cs workers work by themselves and I don't want that.	0.688	0
Computer science is for nerds. vs I think cs workers work by themselves and I don't want that.	0.461	0.006
Have you studied computer science before? vs I think computer science is something I could do with friends.	0.321	0.064

Figure 4

Discussion

Summary of Findings

The goal of this study was to examine differences between the sixth grade students' perceptions of computer science based on students who have taken a basic hands-on computer programming course versus sixth grade students who have not had such a course. Specifically, I wanted to investigate the Technology class students' perceptions of their own abilities and how those perceptions relate to the tangible knowledge that they took from the course. Furthermore, I wanted to look at student ideas of computer science as a discipline (e.g. Is it boring, interesting, fun, etc.) and future interest in computer science (e.g. Are students planning on taking computer science courses in the future, do they want a job in computer science, etc.).

Previously cited studies have indicated that students without a computer science background tend to find the field daunting, too difficult, nerdy, and/or anti-social. Many of these studies focus on students at the upper middle school (eighth grade), high school, and college levels. By finding a slightly younger demographic to study, I wanted to begin a conversation based on two ideas.

First, I wanted to explore if such perceptions were already forming or solidified for sixth grade children. This might give an idea of when it is developmentally appropriate to begin concrete conversations regarding computer science. As we have seen in much of the previously cited literature, even young children may begin to understand the logical foundation that computer science is built upon, as well as understand some basic concepts of how to perform programming tasks. It behooves us to encourage ideas about computer science and computer scientists that break the mold of stereotypes as early as children can feasibly understand in order that they do not limit themselves academically.

Second, if the stereotype was present in the control group, I wanted to look into how easily it might then be broken when students are required to participate in a computer programming course. By experiencing the course firsthand, my hypothesis was that students might be able to overcome stereotypes around computer science and computer scientists.

My findings indicate that students in the control group—who were less likely to have a computer science background as well as less likely to have a concrete understanding of what computer science is—were significantly more likely to think of computer science as boring than students in the Technology class (Figure 1). In fact, not one of the Technology students indicated that he believed computer science is boring. This may suggest two things: 1. Sixth grade students who are not predisposed to computer science may already have certain stereotypical ideas of computer science, and 2. Students who study computer programming can overcome some of the negative beliefs their non-computer science counterparts have. Further, the Technology class had more

definitive affirmations around questions that looked at the tangible nature of performing computer programming tasks (e.g. "I know ways that computer science is used in the world," "I think computer science is problem solving," "I think computer science is frustrating"). This finding indicates that—while not always positive in nature—the Technology class students did in fact have malleable ideas of computer science in comparison to those who have not taken a computer programming course (figure 1).

Across the combined data analysis of both groups I found that students who had studied computer science before, unsurprisingly, exhibited more perceived understanding of computer science. Concretely, in my findings, previous exposure to computer science was positively correlated with "I have a good understanding of computer science," "I know what computer science is," "I think computer science is problem solving," and "I know ways that computer science is used in the world." This data suggests that one way to adjust perceptions of computer science is to expose students to the practice of computer programming. Looking specifically at the responses to "I know what computer science is," the control group response showed $\mu=1.87$ and $\sigma=0.64$ (on a three-point scale), whereas the Technology group responded with $\mu=2.27$ and $\sigma=0.46$ (on a three-point scale). As seen above, students in the Technology class were much more likely to believe they know what computer science is than students in the control group. This indicates that taking the computer programming course increased student confidence in their knowledge of computer science. In terms of motivating students to pursue computer science, and making computer science a realistic career goal, student's perception of their own ability might be more important (especially at this young age) than their actual knowledge of the topic. This is an area for future examination.

Encouragingly, it seems that this improved assessment of ability goes hand in hand with an increased propensity for seeing computer science as a viable path. In comparing the control group with the Technology class, there was a strong positive correlation between "I would like to study computer science in high school" and "I know what computer science is," "I am likely to take computer science classes," and "I think computer science is fun." This supports the idea that perception can be linked to a willingness to pursue computer science. Specifically, results show that taking the Technology class in sixth grade might contribute to the students' desire to take computer science courses in the future. The importance of perception leads me to conclude that mandatory computer science at a young age—specifically as taught through computer programming— is an important step toward overcoming negative biases, both social and self-imposed (based on negative self-assessment of ability).

Perception is one thing, but I was also curious as to whether the students in the experiment group actually had a more developed understanding of what computer science *is* than the control group. By comparing the Technology class surveys to their short answers, I found that—on average—students had a substantial (age appropriate) understanding of computer science and computer programming (Figure 6). This indicates that not only were the Technology students more confident in their perceived understanding than the control group, but they also actually had basic knowledge of computer science and computer programming. This knowledge may explain their affirmative responses to the question "I know what computer science workers do in their jobs" and thereby explain their increased tendency to see computer science as a viable career path.

Prompt:	[redacted]	[redacted]	[redacted]	Encountered problems (frustration, confusion)	[redacted]
95% conf (bottom)	0.541	1.016	1.069	0.373	1.313
95% conf (top)	1.613	2.163	1.956	1.448	2.093
Mean	1.077	1.589	1.513	0.91	1.703
StdDev	0.268	0.286	0.222	0.269	0.195

Figure 5

My observations of the Technology class highlighted how important it is for student learning that children are in an open environment where they are required to experiment and allowed to make mistakes. The teacher instituted a "Try three things" policy that meant if a student encountered a problem, he would have to try three things prior to asking for help from the instructor. The student would need to tell the teacher what three things he tried, thus opening up a discussion where teacher could scaffold learning by asking specific questions about student choices and encouraging exploration that could lead the student to a solution. This focused scaffolding allowed students to make leaps in their programming—and logical thinking.

Further, the technology students' social engagement with each other kept the class from feeling boring (as indicated by the zero positive responses to "Computer science is boring" survey question). Students were able to actively interact with each other on a regular basis, and they often grew loud and enthusiastic about such interactions. This kind of interaction allowed students to engage in scaffolding with each other, often helping each other learn something they could not figure out individually, as seen in Appendix E.

Conclusions Drawn by Results

From this study, we can conclude that students who take the Technology class have different perceptions of computer science from the control group. I was interested to see that the students had a significantly higher response when prompted by the question: "Do you know what computer science is?" and "I would like to have conversations about computer science with friends and family." From this I conclude that the Technology course does indeed increase students' confidence in their knowledge of computer science. These perceptions also indicate that students who take the class obtain a more realistic understanding of computer science than those who have not. For instance, taking the Technology course resulted in positive correlations to "I know ways that computer science is used in the world," "I think computer science is problem solving," and "I think computer science is frustrating." While these are not all positive ideas of computer science, they are realistic and based on actual understanding of computer science rather than a generic stereotype. This is in stark contrast to the control group, in which several findings indicated that students believed the stereotype of a lone person working on problems that are too hard for the average person to understand.

Further, students who have taken the Technology class also know what computer science is. This means that there may be a relationship between the knowledge of what computer science is and a realistic perception of computer science. This change in perception is useful in that it may help students to overcome the cultural biases and stereotypes that potentially cause low enrollment in computer science, as observed in such studies as Taub et al.'s (2009).

In addition, having an understanding of computer science may contribute to wanting to study computer science in the future, thinking computer science is fun, interesting, and/or problem solving. Again, the concrete understanding of computer science allows students to think differently and often more favorably about computer science than those who do not have that basic understanding.

Recommendations for Further Research

Interestingly, I expected to find a strong correlation between taking the Technology course and the social aspect of computer science (e.g. "I think computer science is something I could do with friends"), but this did not show up as significant in the Technology surveys. Further, only one Technology student cited social reasons for liking the class ("I like that we are able to work together sometimes and help each other out"). On the other hand, the control group showed several positive correlations with "I think computer science is something I could do with friends:" "I know what computer science is," "I know ways that computer science is used in the world," "I have a good understanding of computer science," and "Have you studied computer science before?" Further research should be done to better understand what is happening here.

It would also be beneficial to follow up with the sixth grade Technology group with longitudinal studies to see how their understanding, perceptions and interest of computer science evolves over time. Knowing how their ideas of computer science change over time could give insight into the fluidity of perception and whether or not having basic knowledge is enough to keep students interested in computer science over time. Related, further studies should look at different grade levels to see if age is a significant factor in findings. In addition, my study did not control for gender. As

gender is an issue in the field of Computer Science, it would be important to factor it in for future studies.

While I looked specifically at perception and knowledge of computer science, it would be advantageous to have a more in-depth look at what students have learned, as well. Comparisons between what they know *about* computer science and what computer science skills they have learned could provide additional findings.

Works Cited

Bandura, A. Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review,* 84(2), 191-215.

Bureau of Labor Statistics. U.S. department of labor, occupational outlook handbook, 2012-13 edition. Retrieved April 6, 2013, from http://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education,* 58(1), 240-249.

Drobnis, A. W. (2010). *Girls in computer science: A female only introduction class in high school* (Doctoral dissertation, George Mason University).

Eidelman, L., Hazzan, O., Lapidot, T., Mattias, Y., Raijman, D., & Segalov, M. (2011). Mind the (gender) gap: Can a two-hour visit to a hi-tech company change perceptions about computer science? *ACM Inroads, 2*(3), 64-70.

Fletcher, G. H. L., & Lu, J. J. (2009). Human computing skills: Rethinking the k-12 experience. *Communications of the ACM,* 52(2), 23-25.

Frost, D., Verno, A., Burkhart, D., Hutton, M., & Houston, I. S. D. (2009). A model curriculum for k–12 computer science level I objectives and outlines. *CSTA Curriculum Committee.*

Guzdial, M., & Robertson, J. (2012). Levels of abstraction: Pre-teens and career choices. *Communications of the ACM*, 55(12),12-13.

Grover, S. (May 28, 2013). OPINION: Learning to code isn't enough. *EdSurge.* Retrieved June 24, 2013 from https://www.edsurge.com/n/2013-05-28-

opinion-learning-to-code-isn-t-enough

Grover, S., & Pea, R. (2013). Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. *SIGCSE '13, March 6-9,* Denver, CO.

Herling, L. (2011). *Hispanic women overcoming deterrents to computer science: A phenomenological study.* (Doctor of Education, University of South Dakota).

Herrig, B., & Taranto, G. (2012). Being a game changer. *Technology and Engineering Teacher, 72*(3), 27-31.

Hwang, W., Shadiev, R., Wang, C., & Huang, Z. (2012). A study of cooperative computer programming learning behavior and its influence on learning performance. *Paper Presented at the Conference of E-Learning,* Kidmore End. 150-157.

Kelleher, C. & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM, 50*(7), 58-64. doi:10.1145/1272516.1272540

Lamb, A. & Johnson, L. (2011). Scratch: Computer programming for 21st century learners. *Teacher Librarian*, 38(4), 64-68.

Lee, C., Kolodner, J. L., & Goel, A. K. (2011). Guest editorial-creative design: Scaffolding creative reasoning and meaningful learning. *Educational Technology & Society, 14*(1), 1-2.

Maloney, J., Peppler, K., Kafai, Y. B., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with scratch. *SIGCSE'08, March 12-15,* Portland, OR. 367-371.

Mason, J. C. (2009). *Exploratory case study: Experience with a game programming assignment in an introductory computer science classroom.* (Master of Science, *University of Calgary)*

Munson, A., Moskal, B., Harriger, A., Lauriski-Karriker, T. & Heersink, D. (2011). Computing at the high school level: Changing what teachers and students know and believe. *Computers & Education, 57*(2), 1836-1849. doi:10.1016/j.compedu.2011.03.005

Pearson's r Correlation – A Rule of Thumb. Retrieved May 21, 2013, from http://faculty.quinnipiac.edu/libarts/polsci/statistics.html.

Repenning, A. (2012). Programming goes back school. *Communications of the ACM, 55*(5), 38-40.

Rodger, S. H., Hayes, J., Gaetjens, L., Qin, H., Nelson, D., & Tucker, R. (2009). Engaging middle school teachers and students with alice in a diverse set of subjects. *SIGCSE '09,* Chattanooga, TN.

Sheehan, R. (2003). Children's perception of computer programming as an aid to designing programming environments. *Proceedings of the 2003 Conference on Interaction Design and Children (IDC '03),* New York, NY. 75-83. doi:10.1145/953536.953548

Sivilotti, P. A. G., & Laugel, S. A. (2008). Scratching the surface of advanced topics in software engineering: A workshop module for middle school students. *SIGCSE'08, March 12-15,* Portland, OR.

Taub, R., Ben-Ari, M., & Armoni, M (2009).The effect of CS unplugged on

middle-school students' views of CS. In *ACM SIGCSE Bulletin* (Vol. 41, No. 3, pp. 99-103). ACM.

Tholander, J. (2005). Children's perspectives in a game programming discourse. *Journal of Interactive Learning Research, 16*(1), 51-82.

Turkle, S. (2012). *Alone together: Why we expect more from technology and less from each other*. NY, Basic Books.

Vygotsky, L.S. (1978). *Mind in society*. Cambridge, MA: Harvard University Press.

Wagner, A., Gray, J., Corley, J., & Wolber, D. (2013). Using app inventor in a k-12 summer camp. *Proceedings of the 44th ACM Technical Symposium on Computer Science Education,* Denver, CO March 6-9. 621-626.

Werner, L., & Denner, J. (2009). Pair programming in middle school: What does it look like? *Journal of Research on Technology in Education, 42*(1), 29-49.

Wilson, A., Connolly, T., Hainey, T., & Moffat, D. C. (2010). Evaluation of introducing programming to younger school children using a computer game making tool. *Psychology of Programming Interest Group 2010 Workshop,* Madrid.

Xiao, L. & Carroll, J. (2007). Fostering an informal learning community of computer technologies at school. *Behaviour & Information Technology, 26*(1), 23-36.

Appendices

Appendix A
Student survey questionnaires

Table A1
Control group survey questions

* Required

Your Age *

_____

Your grade *

_____

Your school

_____

Does your school offer computer science classes for sixth graders? *

- Yes
- No
- Unsure

Have you studied computer science before? *

Either in or out of the classroom.

- Yes
- No
- Unsure

I know what computer science is. *

	1	2	3	
Disagree				Agree

I know ways that computer science is used in the world. *

	1	2	3	
Disagree				Agree

I am likely to take computer science classes. *

	1	2	3	
Unlikely				Likely

I would like to study computer science in high school. *

	1	2	3	
Disagree				Agree

I would like to have conversations with my friends and family about computer science. *

	1	2	3	
Disagree				Agree

Table A1 continued

Someday, I would like to have a job that uses computer science. *

1 2 3

Disagree ○ ○ ○ Agree

I think computer science is... *

check all that apply

- ☐ Interesting
- ☐ Boring
- ☐ Fun
- ☐ Nerdy
- ☐ Problem solving
- ☐ Something I could do with my friends
- ☐ Something I could do alone
- ☐ Frustrating
- ☐ None of these

Check all statements that you agree with. *

- ☐ I believe that I am smart.
- ☐ I believe that I can achieve any goal I set.
- ☐ I have met people that use computer science in their jobs.
- ☐ I have a good understanding of computer science.
- ☐ My school gives me the chance to learn about computer science.
- ☐ I think Computer Science is interesting.
- ☐ None of these

Check all statements that you agree with. *

- ☐ I think computer science is too hard.
- ☐ I don't think there are a lot of jobs in computer science that I would like.
- ☐ I don't understand computer science.
- ☐ I think computer science is boring.
- ☐ My school does not give me a chance to learn about computer science.
- ☐ Computer science is for nerds.
- ☐ I think people who work in computer science do a lot of work by themselves, and I don't want to do that.
- ☐ None of these

Table A2
Technology class survey questions

* Required

Your Age *

_____

Your grade *

_____

I know what computer science is. *

1 2 3

Disagree ○ ○ ○ Agree

Do you know how to do computer programming? *

○ Yes
○ No
○ Unsure

I know ways that computer science is used in the world. *

1 2 3

Disagree ○ ○ ○ Agree

I am likely to take more computer science classes. *

1 2 3

Unlikely ○ ○ ○ Likely

I would like to study computer science in high school. *

1 2 3

Disagree ○ ○ ○ Agree

I would like to have conversations with my friends and family about computer science. *

1 2 3

Disagree ○ ○ ○ Agree

Someday, I would like to have a job that uses computer science. *

1 2 3

Disagree ○ ○ ○ Agree

Table A2 continued

I think computer science is... *

check all that apply

- [ ] Interesting
- [ ] Boring
- [ ] Fun
- [ ] Nerdy
- [ ] Problem solving
- [ ] Something I could do with my friends
- [ ] Something I could do alone
- [ ] Frustrating
- [ ] None of these

Check all statements that you agree with. *

- [ ] I believe that I am smart.
- [ ] I believe that I can achieve any goal I set.
- [ ] I have met people that use computer science in their jobs.
- [ ] I have a good understanding of computer science.
- [ ] My school gives me the chance to learn about computer science.
- [ ] I think computer science is interesting.
- [ ] None of these

Check all statements that you agree with. *

- [ ] I think computer science is too hard.
- [ ] I don't think there are a lot of jobs in computer science that I would like.
- [ ] I don't understand computer science.
- [ ] I think computer science is boring.
- [ ] My school does not give me a chance to learn about computer science.
- [ ] Computer science is for nerds.
- [ ] I think people who work in computer science do a lot of work by themselves, and I don't want to do that.
- [ ] None of these

Appendix B
Interview questions

B1
Student interview questions

1. What is computer science?
2. What is computer programming?
3. What's your favorite thing you've done in this class?
4. What do you like about this class?
5. What don't you like about this class?
6. Do you like this class? Why/why not?
7. What's something you've learned in this class?

B2
Teacher interview questions

1. What pedagogy do you try to implement into your sixth grade class?
2. Why is the sixth grade course structured the way it is?
3. What works? What doesn't work? What would work better?
4. What did you try prior to this year that you are no longer doing? Why did you abandon these elements?
5. Is the sixth grade Technology course required?
6. How many of the sixth graders come in with an existing programming background?

Appendix C
Student interview responses

Table C1
Adjusted grader response rubric

Concept	0	1	2
[redacted]	No real understanding of what computer science is.	Some understanding of what computer science is.	Good understanding of what computer science is.
[redacted]	No real understanding of what computer *programming* is.	Some understanding of what computer *programming* is.	Good age level understanding of what computer *programming* is.
[redacted]	Student does not engage with the material.	Student has some engagement with the material.	Student has a high level of engagement with the material.
Encountered problems (frustration, confusion)	Encountered no problems with the material	Encountered some problems with the material	Encountered a lot of problems with the material
[redacted]	Did not like the class	Indifferent	Like the class

Table C2
Adjusted graded student responses

Prompt:	[redacted]	[redacted]	Encountered problems (frustration, confusion)	[redacted]
95% conf (bottom)	0.541	1.016	1.069	0.373	1.313
95% conf (top)	1.613	2.163	1.956	1.448	2.093
Mean	1.077	1.589	1.513	0.91	1.703
StdDev	0.268	0.286	0.222	0.269	0.195

Appendix D
Survey results

Table D1
Technology class survey results Pearson's correlation

	A	B	C	D
1	Questions	Pearson r	p value	
2	I know what computer science is. vs I would like to study computer science in high school	0.537	0.048	
3	I know ways that cs is used in the world. vs I would like to talk with f&f about cs.	-0.651	0.012	
4	I know ways that cs is used in the world. vs I think computer science is interesting	0.645	0.013	
5	I know ways that cs is used in the world. vs My school doesn't give me a chance to learn about cs.	-0.679	0.008	
6	I am likely to take computer science classes. vs I would like to study computer science in high school.	0.729	0.003	
7	I am likely to take computer science classes. vs Someday I would like to have a job that uses computer science.	0.641	0.014	
8	I am likely to take computer science classes. vs I think computer science is interesting	0.585	0.028	
9	I am likely to take computer science classes. vs I don't think there are a lot of jobs in cs i'd like.	-0.614	0.02	
10	I would like to study computer science in high school. vs Someday I would like to have a job that uses computer science.	0.709	0.004	
11	I would like to study computer science in high school. vs I think computer science is fun	0.651	0.012	
12	I would like to study computer science in high school. vs I think computer science is too hard.	-0.665	0.01	
13	I would like to study computer science in high school. vs I don't think there are a lot of jobs in cs i'd like.	-0.842	0	
14	I would like to study computer science in high school. vs None of these	0.718	0.004	
15	I would like to talk with f&f about cs. vs None of these	0.691	0.006	
16	Someday I would like to have a job that uses computer science. vs I don't think there are a lot of jobs in cs i'd like.	-0.71	0.004	
17	I think computer science is interesting vs I think computer science is problem solving	0.548	0.043	
18	I think computer science is interesting vs I think computer science is frustrating	-0.73	0.003	
19	I think computer science is fun vs I think computer science is something I could do alone	0.65	0.012	
20	I think computer science is fun vs I think computer science is frustrating	-0.548	0.043	
21	I think computer science is fun vs I don't think there are a lot of jobs in cs i'd like.	-0.548	0.043	
22	I think computer science is fun vs None of these	0.73	0.003	
23	I think computer science is nerdy vs Computer science is for nerds.	0.782	0.001	
24	I think computer science is frustrating vs I think Computer Science is interesting.	-0.559	0.038	
25	I think computer science is frustrating vs I think computer science is too hard.	0.559	0.038	
26	None of these vs Computer science is for nerds.	0.679	0.008	
27	I think Computer Science is interesting. vs I don't think there are a lot of jobs in cs i'd like.	-0.559	0.038	
28	I think Computer Science is interesting. vs Computer science is for nerds.	-0.548	0.043	
29	I think Computer Science is interesting. vs None of these	0.645	0.013	
30	I think computer science is too hard. vs I don't think there are a lot of jobs in cs i'd like.	0.559	0.038	
31	I think computer science is too hard. vs None of these	-0.645	0.013	
32	I don't think there are a lot of jobs in cs i'd like. vs None of these	-0.75	0.002	
33	Computer science is for nerds. vs I think cs workers work by themselves and I don't want that.	0.645	0.013	
34	I think cs workers work by themselves and I don't want that. vs None of these	-0.548	0.043	
35				

Table D2
Technology class summary statistics

	Question	Mean	Std.Dev.
2	Your Age	11.786	0.41
3	Your grade	6	0
4	I know what computer science is.	2.214	0.41
5	Do you know how to do computer programming?	2.571	0.623
6	I know ways that computer science is used in the world.	2.857	0.35
7	I am likely to take more computer science classes.	2.214	0.773
8	I would like to study computer science in high school.	2.143	0.833
9	I would like to talk with my friends and family about cs.	1.5	0.627
10	Someday I would like to have a job that uses computer science.	1.857	0.639
11	I think computer science is interesting	2.429	0.904
12	I think computer science is boring	1	0
13	I think computer science is fun	1.571	0.904
14	I think computer science is nerdy	1.429	0.821
15	I think computer science is problem solving	1.857	0.99
16	I think computer science is something I could do with my friends	1.5	0.824
17	I think computer science is something I could do alone	1.571	0.904
18	I think computer science is frustrating	1.857	0.99
19	None of these	1.143	0.515
20	I believe that I am smart.	2.571	0.821
21	I believe that I can achieve any goal I set.	2.429	0.904
22	I have met people that use computer science in their jobs.	2.143	0.99
23	I have a good understanding of computer science.	1.714	0.958
24	My school gives me the chance to learn about computer science.	3	0
25	I think computer science is interesting.	2.286	0.958
26	None of these	1	0
27	I think computer science is too hard.	1.714	0.958
28	I don't think there are a lot of jobs in cs that I would like.	1.857	0.99
29	I don't understand computer science.	1.429	0.821
30	I think computer science is boring.	1	0
31	My school does not give me a chance to learn about cs.	1.143	0.515
32	Computer science is for nerds.	1.286	0.7
33	I think cs workers work by themselves and I don't want that.	1.571	0.904
34	None of these	1.857	0.99

Table D3
Control group survey results Pearson's correlation

Questions	Pearson r	p value
Your Age vs I have a good understanding of computer science.	0.454	0.044
Your Age vs I think computer science is boring.	0.516	0.02
School offers CS classes for 6th graders? vs My school gives me the chance to learn about computer science.	0.585	0.007
Have you studied computer science before? vs I know what computer science is.	0.489	0.029
Have you studied computer science before? vs I think computer science is something I could do with friends	0.545	0.013
Have you studied computer science before? vs I have a good understanding of computer science.	0.663	0.001
I know what computer science is. vs Someday I would like to have a job that uses computer science.	0.522	0.018
I know what computer science is. vs I think computer science is something I could do with friends	0.486	0.03
I know what computer science is. vs My school does not give me a chance to learn about cs.	-0.633	0.003
I know ways that computer science is used in the world. vs I think computer science is something I could do with friends	0.49	0.028
I know ways that computer science is used in the world. vs I have a good understanding of computer science.	0.641	0.002
I know ways that computer science is used in the world. vs My school does not give me a chance to learn about cs.	-0.581	0.007
I am likely to take computer science classes. vs Someday I would like to have a job that uses computer science.	0.678	0.001
I am likely to take computer science classes. vs I think computer science is fun	0.642	0.002
I would like to study computer science in high school. vs Would like to have conversations with f&f about CS.	0.535	0.015
I would like to study computer science in high school. vs Someday I would like to have a job that uses computer science.	0.471	0.036
I think computer science is interesting vs I think computer science is boring	-0.577	0.008
I think computer science is interesting vs I think computer science is frustrating	-0.577	0.008
I think computer science is interesting vs I think Computer Science is interesting.	0.522	0.018
I think computer science is boring vs I think computer science is frustrating	0.444	0.05
I think computer science is boring vs I think computer science is too hard.	0.444	0.05
I think computer science is boring vs My school does not give me a chance to learn about cs.	0.454	0.044
I think computer science is problem solving vs I think computer science is something I could do alone	0.509	0.022
I think computer science is problem solving vs I think computer science is too hard.	0.444	0.05
I think computer science is something I could do with friends vs I have a good understanding of computer science.	0.63	0.003
I have met people that use computer science in their jobs. vs None of these	0.63	0.003
I have a good understanding of computer science. vs My school gives me the chance to learn about computer science.	0.764	0
I have a good understanding of computer science. vs I think Computer Science is interesting.	0.504	0.023

Table D3 continued

I have a good understanding of computer science. vs My school does not give me a chance to learn about cs.	-0.48	0.032
My school gives me the chance to learn about computer science. vs I think Computer Science is interesting.	0.553	0.011
I think Computer Science is interesting. vs I don't understand computer science.	-0.452	0.045
I think computer science is too hard. vs I don't think there are a lot of jobs in cs I would like.	0.454	0.044
I think computer science is too hard. vs I think cs workers work by themselves and I don't want that.	0.509	0.022
I don't think there are a lot of jobs in cs I would like. vs I think computer science is boring.	0.572	0.008
I don't think there are a lot of jobs in cs I would like. vs I think cs workers work by themselves and I don't want that.	0.892	0
I don't think there are a lot of jobs in cs I would like. vs None of these	-0.48	0.032
I don't understand computer science. vs I think cs workers work by themselves and I don't want that.	0.491	0.028
I think computer science is boring. vs Computer science is for nerds.	0.608	0.004
I think computer science is boring. vs I think cs workers work by themselves and I don't want that.	0.642	0.002
My school does not give me a chance to learn about cs. vs None of these	-0.48	0.032

Table D4
Control group summary statistics

Question	Mean	Std.Dev.
Taken CS?	0	0
Your Age	11.9	0.624
Your grade	6	0
School offers CS classes for 6th graders?	2.15	0.726
Have you studied computer science before?	1.7	0.954
I know what computer science is.	1.85	0.654
I know ways that computer science is used in the world.	1.75	0.766
I am likely to take computer science classes.	2.05	0.74
I would like to study computer science in high school.	2.1	0.768
Would like to have conversations with f&f about CS.	1.95	0.865
Someday I would like to have a job that uses computer science.	1.75	0.622
I think computer science is interesting	2.5	0.866
I think computer science is boring	1.2	0.6
I think computer science is fun	1.4	0.8
I think computer science is nerdy	1.2	0.6
I think computer science is problem solving	1.2	0.6
I think computer science is something I could do with friends	1.5	0.866
I think computer science is something I could do alone	1.6	0.917
I think computer science is frustrating	1.2	0.6
I think computer science is none of these	1	0
I believe that I am smart.	2.2	0.98
I believe that I can achieve any goal I set.	2.6	0.8
I have met people that use computer science in their jobs.	1.5	0.866
I have a good understanding of computer science.	1.6	0.917
My school gives me the chance to learn about computer science.	1.4	0.8
I think Computer Science is interesting.	1.9	0.995
None of these	1	0
I think computer science is too hard.	1.2	0.6
I don't think there are a lot of jobs in cs I would like.	1.7	0.954
I don't understand computer science.	1.4	0.8
I think computer science is boring.	1.3	0.714
My school does not give me a chance to learn about cs.	1.7	0.954
Computer science is for nerds.	1.3	0.714
I think cs workers work by themselves and I don't want that.	1.6	0.917
None of these	1.6	0.917

Table D5
Combined survey results Pearson's correlation

Questions	Pearson r	p value
Taken CS? vs Does your school offer cs for sixth graders?	0.601	0
Taken CS? vs Have you studied computer science before?	0.658	0
Taken CS? vs I know ways that cs is used in the world.	0.655	0
Taken CS? vs I think computer science is problem solving	0.381	0.026
Taken CS? vs I think computer science is frustrating	0.381	0.026
Taken CS? vs My school gives me the chance to learn about computer science.	0.789	0
Your Age vs I think computer science is boring.	0.461	0.006
Does your school offer cs for sixth graders? vs Have you studied computer science before?	0.565	0.001
Does your school offer cs for sixth graders? vs My school gives me the chance to learn about computer science.	0.761	0
Does your school offer cs for sixth graders? vs My school doesn't give me a chance to learn about cs.	-0.398	0.02
Have you studied computer science before? vs I know what computer science is.	0.509	0.002
Have you studied computer science before? vs I know ways that cs is used in the world.	0.631	0
Have you studied computer science before? vs I have met people that use computer science in their jobs.	0.37	0.031
Have you studied computer science before? vs I have a good understanding of computer science.	0.415	0.015
Have you studied computer science before? vs My school gives me the chance to learn about computer science.	0.713	0
Have you studied computer science before? vs I think Computer Science is interesting.	0.35	0.043
Have you studied computer science before? vs My school doesn't give me a chance to learn about cs.	-0.42	0.013
I know what computer science is. vs I know ways that cs is used in the world.	0.476	0.004
I know what computer science is. vs I am likely to take computer science classes.	0.392	0.022
I know what computer science is. vs I would like to study computer science in high school.	0.373	0.03
I know what computer science is. vs Someday I would like to have a job that uses computer science.	0.47	0.005
I know what computer science is. vs I think computer science is boring	-0.421	0.013
I know what computer science is. vs I believe that I can achieve any goal I set.	-0.35	0.042
I know what computer science is. vs My school doesn't give me a chance to learn about cs.	-0.584	0
I know ways that cs is used in the world. vs I have met people that use computer science in their jobs.	0.387	0.024
I know ways that cs is used in the world. vs I have a good		

Table D5 continued

I know ways that cs is used in the world. vs I have a good understanding of computer science.	0.358	0.038
I know ways that cs is used in the world. vs My school gives me the chance to learn about computer science.	0.658	0
I know ways that cs is used in the world. vs My school doesn't give me a chance to learn about cs.	-0.637	0
I am likely to take computer science classes. vs I would like to study computer science in high school.	0.514	0.002
I am likely to take computer science classes. vs Someday I would like to have a job that uses computer science.	0.665	0
I am likely to take computer science classes. vs I think computer science is fun	0.554	0.001
I am likely to take computer science classes. vs I don't think there are a lot of jobs in cs i'd like.	-0.442	0.009
I would like to study computer science in high school. vs I would like to talk with f&f about cs.	0.456	0.007
I would like to study computer science in high school. vs Someday I would like to have a job that uses computer science.	0.575	0
I would like to study computer science in high school. vs I think computer science is fun	0.354	0.04
I would like to study computer science in high school. vs I think Computer Science is interesting.	0.436	0.01
I would like to study computer science in high school. vs I think computer science is too hard.	-0.441	0.009
I would like to study computer science in high school. vs I don't think there are a lot of jobs in cs i'd like.	-0.573	0
I would like to talk with f&f about cs. vs I have met people that use computer science in their jobs.	-0.371	0.031
Someday I would like to have a job that uses computer science. vs I don't think there are a lot of jobs in cs i'd like.	-0.51	0.002
Someday I would like to have a job that uses computer science. vs None of these	0.436	0.01
I think computer science is interesting vs I think computer science is boring	-0.417	0.014
I think computer science is interesting vs I think computer science is something I could do alone	0.387	0.024
I think computer science is interesting vs I think computer science is frustrating	-0.61	0
I think computer science is interesting vs I think Computer Science is interesting.	0.503	0.002
I think computer science is interesting vs I don't think there are a lot of jobs in cs i'd like.	-0.351	0.042
I think computer science is interesting vs I don't understand computer science.	-0.354	0.04
I think computer science is boring vs I think computer science is boring.	0.363	0.035

Table D5 continued

I think computer science is boring vs My school doesn't give me a chance to learn about cs.	0.451	0.007
I think computer science is fun vs None of these	0.461	0.006
I think computer science is nerdy vs None of these	0.419	0.014
I think computer science is nerdy vs Computer science is for nerds.	0.531	0.001
I think computer science is problem solving vs I think computer science is something I could do alone	0.403	0.018
I think computer science is problem solving vs I believe that I am smart.	0.384	0.025
I think computer science is problem solving vs My school gives me the chance to learn about computer science.	0.384	0.025
I think computer science is something I could do alone vs I believe that I can achieve any goal I set.	0.358	0.038
I think computer science is something I could do alone vs I think Computer Science is interesting.	0.35	0.042
I think computer science is frustrating vs I think computer science is too hard.	0.403	0.018
None of these vs Computer science is for nerds.	0.419	0.014
I have met people that use computer science in their jobs. vs I have a good understanding of computer science.	0.361	0.036
I have met people that use computer science in their jobs. vs My school gives me the chance to learn about computer science.	0.378	0.027
I have a good understanding of computer science. vs My school gives me the chance to learn about computer science.	0.4	0.019
My school gives me the chance to learn about computer science. vs I think Computer Science is interesting.	0.41	0.016
My school gives me the chance to learn about computer science. vs My school doesn't give me a chance to learn about cs.	-0.449	0.008
I think Computer Science is interesting. vs I don't think there are a lot of jobs in cs i'd like.	-0.35	0.043
I think Computer Science is interesting. vs I don't understand computer science.	-0.394	0.021
I think Computer Science is interesting. vs None of these	0.45	0.008
I think computer science is too hard. vs I don't think there are a lot of jobs in cs i'd like.	0.497	0.003
I think computer science is too hard. vs None of these	-0.376	0.028
I don't think there are a lot of jobs in cs i'd like. vs I don't understand computer science.	0.348	0.044
I don't think there are a lot of jobs in cs i'd like. vs I think computer science is boring.	0.395	0.021
I don't think there are a lot of jobs in cs i'd like. vs I think cs workers work by themselves and I don't want that.	0.688	0

Table D5 continued

I don't think there are a lot of jobs in cs i'd like. vs None of these	-0.581	0
I don't understand computer science. vs None of these	-0.376	0.028
I think computer science is boring. vs Computer science is for nerds.	0.456	0.007
I think computer science is boring. vs I think cs workers work by themselves and I don't want that.	0.482	0.004
My school doesn't give me a chance to learn about cs. vs None of these	-0.41	0.016
Computer science is for nerds. vs I think cs workers work by themselves and I don't want that.	0.461	0.006
I think cs workers work by themselves and I don't want that. vs None of these	-0.477	0.004

Table D6
Combined summary statistics

Question	Mean	Std.Dev.
Taken CS?	0.412	0.492
Your Age	11.853	0.549
Your grade	6	0
Does your school offer cs for sixth graders?	2.5	0.697
Have you studied computer science before?	2.235	0.972
I know what computer science is.	2	0.594
I know ways that cs is used in the world.	2.206	0.832
I am likely to take computer science classes.	2.118	0.758
I would like to study computer science in high school.	2.118	0.796
I would like to talk with f&f about cs.	1.765	0.807
Someday I would like to have a job that uses computer science.	1.794	0.631
I think computer science is interesting	2.471	0.882
I think computer science is boring	1.118	0.471
I think computer science is fun	1.471	0.848
I think computer science is nerdy	1.294	0.708
I think computer science is problem solving	1.471	0.848
I think computer science is something I could do with friends	1.5	0.849
I think computer science is something I could do alone	1.588	0.911
I think computer science is frustrating	1.471	0.848
None of these	1.059	0.338
I believe that I am smart.	2.353	0.936
I believe that I can achieve any goal I set.	2.529	0.848
I have met people that use computer science in their jobs.	1.765	0.972
I have a good understanding of computer science.	1.647	0.936
My school gives me the chance to learn about computer science.	2.059	0.998
I think Computer Science is interesting.	2.059	0.998
None of these	1	0
I think computer science is too hard.	1.412	0.809
I don't think there are a lot of jobs in cs i'd like.	1.765	0.972
I don't understand computer science.	1.412	0.809
I think computer science is boring.	1.176	0.567
My school doesn't give me a chance to learn about cs.	1.471	0.848
Computer science is for nerds.	1.294	0.708
I think cs workers work by themselves and I don't want that.	1.588	0.911
None of these	1.706	0.956

Appendix E
Sample observation notes from the technology class

January 28, 2013
Participants: Three 6th grade boys, A, B, and C
Setting: Technology classroom at an all boys preparatory school in New York City. The students are working on the final stages of making individual games using Scratch on their computers. Boy A and Boy B sit side by side at their own computers, working on a game of their own design. Boy C is across the aisle at other row of computers and does not come over until later. Boy A and Boy B decide to try out each other's games. The interaction took about five minutes total.

Boy A: (*to Boy B who is starting the game*) Do the instructions. (*Boy B clicks on instructions where a figure and some text comes onto the screen.)* No, I need to make that guy red, still. (*Boy B begins playing the game. He moves the player up and down along the side of the screen, but is unable to shoot any of the oncoming enemies.*) You have to go off the wall to shoot.

Boy B: (*Throws hands up in frustration.)* Oh come on, your game is....

Boy A: Maybe I'll change that.

(*Boy A and Boy B turn their attention to Boy A's screen. Boy A begins playing Boy B's game. Boy C saunters over to the two boys.)*

Boy A: Your intro is pretty good, actually. Your game is very hard. (*All three watch as Boy A continues to play for about two minutes.)*

Boy C: I'm doing a disability. It's so cool.

Boy B: Really? How? Can I see?

Boy C: I'm still kind of working on it, but yeah. (*Boy B and Boy C go to look at Boy C's game. Boy A quits playing Boy B's game and starts working on making his intro character red.)*

Assessment: While the assignment the children are working on centers around creation of their own individual game, the students interact with each other in order to improve their own projects. They do this in two ways. First, by playing each other's games, they are learning what works and does not work for an audience and can then modify their games to cater to what their classmates like; seeing the game in practice makes for a more interesting game. In this scenario, Boy A saw his game being played by Boy B and was able to adjust his game based on what he observed. Second, through the "zone of proximal development," students can learn by viewing what their classmates are working on, and figuring out how to implement it into their own games. Boy B can learn from what Boy C is doing in his own game, while Boy C may be able to better understand what he has learned to do through modeling it for his classmate. In this observation, both kinds of learning are present.