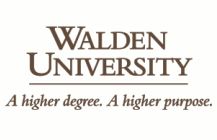

2019

An Examination of Abstraction in K-12 Computer Science Education

Christine Lynn Liebe
Walden University

Walden University

College of Education

This is to certify that the doctoral dissertation by

Christine Liebe

has been found to be complete and satisfactory in all respects,
and that any and all revisions required by
the review committee have been made.

Review Committee
Dr. Wade Smith, Committee Chairperson, Education Faculty
Dr. Danielle Hedegard, Committee Member, Education Faculty
Dr. Narjis Hyder, University Reviewer, Education Faculty

Chief Academic Officer
Eric Riedel, Ph.D.

Walden University
2019

Abstract

An Examination of Abstraction in K-12 Computer Science Education

by

Christine Liebe

MS, Walden University, 2008

BA, Michigan State University, 1989

Dissertation Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Education: Curriculum Instruction and Assessment

Walden University

April 2019

Abstract

Computer scientists have been working towards a common definition of abstraction; however, the instruction and assessment of abstraction remain categorically underresearched. Because abstraction is often cited as a component of computational thinking, abstraction has been summarily likened to a higher order thinking skill. A broad conceptual framework including philosophy, psychology, constructionism, and computational thinking was aligned with the descriptive qualitative design and guided the literature review and data analysis. This qualitative examination of how teachers determine curriculum, deliver instruction, and design assessments in K-12 computer science education provides insight into best practices and variables for future quantitative study. The instructional strategies, objectives, and assessments of twelve K-12 computer science teachers from 3 states were examined in this descriptive qualitative examination of instruction using thematic coding analysis. The majority of teachers had little to no professional development regarding teaching abstraction. All teachers in the study were unsure what student abstraction abilities should be according to grade level. Teachers' understanding of abstraction ranged from very little knowledge to very knowledgeable. The majority of teachers did not actively assess abstraction. Teachers described successfully teaching abstraction through multiple instructional practices and spiraling curriculum. Practical descriptive insights illuminate additional variables to research the instruction of abstraction qualitatively and quantitatively, as well as provide anecdotal instructional successes.

An Examination of Abstraction in K-12 Computer Science Education

Christine Liebe

MS, Walden University, 2008

BA, Michigan State University, 1989

Dissertation Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Education: Curriculum, Instruction, and Assessment

Walden University

April 2019

Dedication

This study is dedicated to the prosperous and compassionate futures all students deserve and that we can all have when all people learn to use computers as a tool for ethical creative problem-solving. I would especially like to honor the volunteer teachers who made this study possible. The teacher participants sacrificed personal time to share their insights and expertise shared amongst demanding schedules, family obligations, graduate studies, and health issues. I am deeply grateful.

Acknowledgments

Many thanks go to my mentors, committee members, and everyone who kept me in their prayers, especially to the Creator. I would especially like to thank Steve Kaufman, Dr. Tracy Camp, Dr. Debra Winston, Dr. Yvette Myrick, Dr. Janice Gardner, Dr. Wade Smith, Dr. Danielle Hedegard, Dr. Narjis Hyder, Dr. Paula Dawidowicz, Dr. Connie Warner, Dr. Sylvia Gholston, Dr. Andree Robinson-Neal and many other colleagues who inspired and encouraged me. Thank you to Walden University. The Don E. Ackerman Fellowship which I was awarded supported this scholarship. I am so grateful for this life. I am grateful to my grandmother, Evelyn Vukin, a tireless classroom teacher, my parents, Tom and Linda Drahnak, who gave their unconditional support, my children, Mara and Pearce McLaughlin, who gave me inspiration, my aunt, Julie Nettere, who stood by me, my two cousins, Laura Fisanick and Dawn Drahnak, who received their doctorates several years ago, the many friends, Dr. Steve Lewis, and colleagues along the way who cheered me on, and Greg Otto, who made me smile.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction to the Study

As computer science gains recognition and evolves as a discipline, the study of *abstraction*, the ability to representationally minimize extraneous detail, is important for student competency (Lau, 2018). Countries such as England, Scotland, Estonia, Finland, Australia, Israel, and Singapore require computer science (CS) courses in secondary school and in some countries CS education is required in middle school and even elementary school (Deruy, 2017). In the United States, Iowa, Arkansas, Nevada, Texas, and West Virginia require that computer science courses are offered and have adopted computer science standards (Code.org, 2017). Many other states, such as Colorado, are in the process of developing and adopting CS standards and have hired state level CS education support specialists (Code.org, 2017). The cities of Chicago and New York City require computer science credits for high school graduation (Code.org, 2017). Virginia has embedded computer science into content standards (Code.org, 2017). Educational trends, such as teaching drag and drop programming and computational thinking, are useful instructional strategies in computer science, and additional curriculum and instruction are necessary to assist students in gaining foundational knowledge required for professional success (Denning, Tedre, & Yongpradit, 2017). The field of computer science education is growing, and CS educational research will help teachers and students around the globe.

Computer science is a deceiving name for a subject regarding using the computer as a tool to express human intelligence and creativity. (Norman, 2006). The use of the

word computer highlights the tool, not the essence of the activity of solving problems and using intelligence (Hazzan, Lapidot, & Ragonis, 2014; Norman, 2006). Just as telescopes are tools for astronomers, particle accelerators assist physicists, and Petri dishes aide biologists, the computer helps humans to solve complex problems (Norman, 2006). Because the field of computer science is new and not well understood, the tool has become associated with the essence of the subject (Norman, 2006). In 1986, Dr. Hal Abelson explained on video that computer science formalizes intuition about the processes of controlling complexity (Norman, 2006). In chess, the rules or procedures of the game can be taught in minutes; however, the concept of the game and the implications of the rules take much longer to master (Norman, 2006). CS has similar concepts and procedures as chess.

Proficiency in computer science requires many thinking skills, such as sequencing, induction, deduction, problem-solving, and creativity (CS10K, 2016; College Board, 2016). The term computer science is used because creating programs involves aspects of the scientific method, as well as creativity and design principles (Hazzan et al., 2014). Many lines of code must be abstracted into representative features to make coding efficient and elegant, thus controlling complexity (Colburn, 2015; Perrenet, 2010). Additionally, computer science requires knowledge of algorithms, organizing and sorting data, navigating the Internet, cybersecurity, as well as a basic understanding of computer hardware and software systems (Brookshear, 2012). Instructors of computer science are also encouraged to guide students ethically in creating technology that extends human respect and compassion (College Board, 2016).

Abstraction is an essential and simultaneously advanced concept consisting of several levels of procedure and conceptual awareness that computer programmers must develop (Armoni, 2013; Colburn, 2000; Hazzan, Lapidot, & Ragonis, 2015). The ability to use abstraction effectively is a teachable skill (Fuller et al., 2007). The knowledge of many concepts, skills, and procedures are important to become proficient with abstraction and with computers. Abstraction is an essential skill that programmers, engineers, and technicians must understand and execute to create efficient and functional computational solutions.

Unfortunately, little research exists that offers computer science instructors in K-12 educational guidance about the age at which students can begin to learn abstraction. Similarly, no research exists offering instructional guidance regarding teaching abstraction. Researching abstraction instruction is challenging because the concept of abstraction is complex and not easily defined (Armoni, 2013; Perrenet, 2010). Wing (2006) introduced the concept of *computational thinking*, of which, abstraction is a subskill. Wing's concept of computational thinking has efficaciously integrated CS in math and science content and has also propagated computer science instruction. The majority of studies examining computational thinking have been conducted with university participants, not K-12 students (Czerkawski & Lyman III, 2015; Grover & Pea, 2013; Lim, Hosak, & Vogt, 2012; Lye & Koh, 2014). Because of the preponderance of postsecondary CS educational research, this study may help inform the instruction applied by K-12 computer science educators. Many of the K-12 studies involve the assessment of student responses to a variety of instructional software programs (Bers,

2010; Bers et al., 2014; Lee et al., 2014; Wang et al., 2014; Reuker et al., 2013). Computer scientists tend to construct software to teach students and then research the effectiveness of the new software on the learning experience. Instead of researching educational performance and variables, such research has focused on demonstrating the viability of software to instruct students. Although challenges exist in defining abstraction and researching the instruction of abstraction, I pursue a practical definition and understanding of abstraction, namely minimizing extraneous detail, and share best practices obtained from K-12 CS teachers in this qualitative examination.

This study may also provide qualitative information about instructional best practices and professional development pathways for teaching computer science in K-12 classrooms. Engineering is a subject with formalized operations that guides people to construct things constrained only by the tolerance of physics (Norman, 2006). According to Abelson, captured on video in 1986, computer science is only limited by human imagination (Norman, 2006). Computer science is not concrete but the product of human imagination. Based on the assumption that computers are an abstraction of human ingenuity, humans are needed to provide essential instruction in computer science (Colburn, 2000). Variables needed for the effective instruction of abstraction may be identified as a result of this qualitative examination. Several studies have qualitatively examined the acquisition of computer coding skills (Denner, Werner, & Ortiz, 2014; Fuller et al., 2007; Wang, Wang, & Liu, 2014) and computational thinking (Daily & Eugene, 2013; Lee, 2010; Lee et al., 2014; Pellas & Peroutseas, 2016). In this qualitative study, I examine how teachers acquire the knowledge, skills, and pedagogical theory that

they use to teach computer science. I also identify variables that future professional development can address and variables that can be quantitatively researched to evaluate the effectiveness of instruction. The qualitative examination of K-12 instruction of abstraction will provide insight into the nature of effective CS teaching for a variety of grade levels. Specifically, in Chapter 1, I provide information on the background, problems, purpose, significance, research questions, nature, definitions, assumptions, scope and limitations, and delimitations regarding this qualitative examination of the instruction of abstraction.

Computer science education has the power to positively impact society, educational systems, classroom systems, and individual students. Training teachers in all content areas to teach computer science is a large but necessary undertaking if people are going to learn to use the computer as the multi-faceted tool it was designed to be, not just a printing or publishing device (Code.org, 2016; Computer Science Teachers Association [CSTA], 2015). Computer science education that allows secondary students to explore and create in collaboration with teachers, using portfolio-based assessment, supported by computer science businesses, will improve the workforce and the economy. At the mega level, humankind can benefit from the enhanced creativity and power over technology that students will learn (Kaufman et al., 2003). Economies will benefit from increased productivity, innovation, a more capable workforce, and increased employment resulting from entrepreneurial endeavors. Educators will be able to assist students in becoming scholar innovators. Students will learn to express their creativity to solve the world's

problems. Computer science education can potentially change thinking on a personal, societal, and global level from victimization due to technology to evolution because of technology. Because abstraction is an essential thinking skill needed to code computers effectively, this study will facilitate effective instruction in computer science, which is ultimately the instruction of the technological equivalent of human creativity and communication.

In this dissertation, I explain why abstraction is a multifaceted concept with many procedural possibilities. Given the pervasiveness of technology; the fact that technology is replacing jobs; the imminent need for Science, Technology, Engineering, and Mathematics (STEM) workers (mainly computer science workers); and the lack of computer science education research guiding K-12 curriculum, instruction, and assessment, abstraction in computer science is a worthy topic to examine. Furthermore, the study of the instruction of abstraction is essentially an interdisciplinary study that bridges psychology, education, mathematics, and computer science. Examining the experience of educators with the concepts and procedures related to abstraction in all grade levels will provide important information for teachers, students, parents, foundations, policy-makers, curriculum developers, software developers, and researchers in several disciplines.

Background

According to computer science experts, we are experiencing a digital information explosion and revolution (Fayer, Lacey, & Watson, 2017). Money, information, documents, voices, and images are transferred wirelessly from country to country and

from device to device, in seconds as digital bits that can last forever (Abelson, Ledeen, & Lewis, 2008). Just as fire can be used for heating and cooking or destruction, the way we use information can enlighten, corrupt, or enslave as evidenced by online doctoral education, hacking in the recent 2016 United States presidential election, and the governmental control of China's Internet (Abelson et al., 2008). Technology has made privacy almost impossible, and laws have not kept up with technological changes (Abelson et al., 2008). There is an urgent need for all democratic citizens to use technology proficiently, intelligently, and ethically; otherwise, the majority of people will continue to be at the mercy of technological advances (Abelson et al. 2008). Teaching students to learn to use an advanced concept, such as abstraction effectively, will help them become empowered users and creators of technology (Fayer et al., 2017). As Fayer et al. (2017) assert, STEM employment is beginning to dominate the new positions being created.

The economy will also benefit from an educational system that prepares students to use computers effectively. Employment for workers with STEM skills was double that of nonSTEM employment between 2009 and 2015, 10.5% to 5.2% for nonSTEM job growth (Fayer et al., 2017). Computer occupations made up 49% of all 8.6 million STEM jobs in 2015, and the need for software developers, systems analysts, network administrators, information and systems managers, computer programmers, computer sales and service representatives exceeds the need for mechanical and civil engineers (Fayer et al., 2017). In 2015, the average STEM job wage was $87,570 double the average wage for nonSTEM jobs, $48,320 (Fayer et al., 2017). The U.S. Bureau of Labor

and Statistics expects over a million openings for computer occupations from 2014 to 2024 (Fayer et al., 2017). In the near future, education will need to better prepare students for computer occupations. In all disciplines, the professional with computer science expertise will have a great impact in their field.

Education must prepare teachers to match the growing demand for STEM and computer science proficient employees. There is a dire need for computer science teachers. The ten fastest growing STEM jobs require a bachelor's degree or higher (Fayer, Lacey, & Watson, 2017). Web developers, computer support technicians, and a variety of other occupations expected to grow by 2024 typically require an associate's degree or less (Fayer et al., 2017). Private coding boot camps are supplying a demand for intensive higher education in computer science that community colleges and universities are failing to provide (Code Fellows, 2019; General Assembly, 2019). Apple, Microsoft, and Google offer free educational support and training to teachers and staff, and sometimes even free computers (Apple, 2019; Google, 2019; Microsoft, 2019). The National Science Foundation (NSF) offers millions of dollars in grants to support computer science teacher development (CS10K, 2017). Cuny (2017), a National Science Foundation program director, estimates there is a need for over 30,000 high school computer science teachers, a figure that doesn't include the need for teachers in middle and elementary schools. The need for CS educators validates CS instructional research, such as this dissertation study.

Nonprofit organizations, such as Code.org, are dedicated to promoting computer science, especially computer coding in K-12 education. However, a nonprofit free CS

education curriculum will not solve the lack of CS courses and teachers in education. Only eight states in the United States have K-12 computer science standards – Washington, Idaho, Missouri, Illinois, West Virginia, New Jersey, Connecticut, and Florida (Code.org, 2017). Thirty-three states plus the District of Columbia count computer science classes towards high school graduation requirements (Code.org, 2017). Most parents surveyed, 93%, want their children to learn computer science, but only 40% of schools offer CS courses (Code.org, 2017). The private sector, foundations, nonprofits, and parents are asking for increases in CS education. Policy-makers are beginning to take heed.

Research on CS Abstraction

The term abstraction in computer science education is being professionally defined and used, although specific research investigating teaching abstraction is limited (Armoni, 2013; Fuller et al., 2007; Grover & Pea, 2013; Perrenet, 2010). Studies have focused on the success of computer coding software for children and how children interact with the software (Fessakis, Gouli, & Mavroudi, 2013; Kazakoff & Bers, 2012; Lee, 2010; Wang et al., 2014). Other researchers included abstraction as a part of computational thinking (Armoni, 2013; Bers, Flannery, Lee, 2010; Sullivan, Kazakoff & Bers, 2013; Wang et al., 2014). Instructional theory regarding teaching abstraction in math may offer solutions to teaching abstraction in computer coding (White & Mitchelmore, 2010). Harlow and Leak (2014) observed elementary students exhibiting beginning abstraction skills in computer coding. However, no qualitative or quantitative studies found so far address the most effective age to introduce abstraction. I offer an

additional exploration of educational research regarding abstraction and similar topics in the literature review of this dissertation.

Abstraction as a Skill and a Concept in Computer Science

Abstraction is a skill used in many disciplines. In computer coding, abstraction is used to modularize and manage complex coding commands (Armoni, 2014; Colburn, 2000). Abstraction skills require the use of induction and deduction. For example, software writers use induction when they have a lot of code that they want to simplify and deduction when they want to choose one coding solution from many possibilities. Minimizing detail while writing code is a classic use of inductive and deductive reasoning, Computer science educators have debated whether abstraction is an innate skill or a skill that can be taught (Armoni, 2014). Armoni reviewed the literature regarding definitions of abstraction, theory on levels of abstraction, research regarding abstraction as a precondition for relevant computer science work, research regarding abstraction as a result of computer science education, and theory on teaching abstraction. Levels of abstraction are categorized by size (large to small), meaning (how to what), and thinking (problem to solution). Armoni's literature review is not extensive, but the research is current and the theory she cited came from respected academic sources. Given the lack of research on the subject of abstraction, Armoni provided a convincing foundation for her conceptual framework.

The Perrenet, Kassenbrood, and Groote (PKG) hierarchy is a significant theoretical framework for abstraction (Armoni, 2014). According to the PKG hierarchy, abstraction takes place on the execution level with the algorithm and the machine (Figure

5). Next abstraction occurs at the program level. Then at the object level, people perceive a program or an algorithm as a thing rather than the complex processes they are. Finally, abstraction takes place on the problem level when people deductively pose a solution, then create code inductively abstracting the code to simplify making the code elegant. The PKG hierarchy is complex enough to address the critical cognitive building blocks needed for producing abstraction in computer science. The illustration in Figure 1, which is an open educational resource, demonstrates graphically how abstraction as a concept and skill progresses from the execution level, addressing the machine and algorithm, to the program level, such as Fortran, to the object level, a computer game (Angry Birds) for instance, to the problem level, a computer game that uses the computer's graphic user interface to have fun and make money.

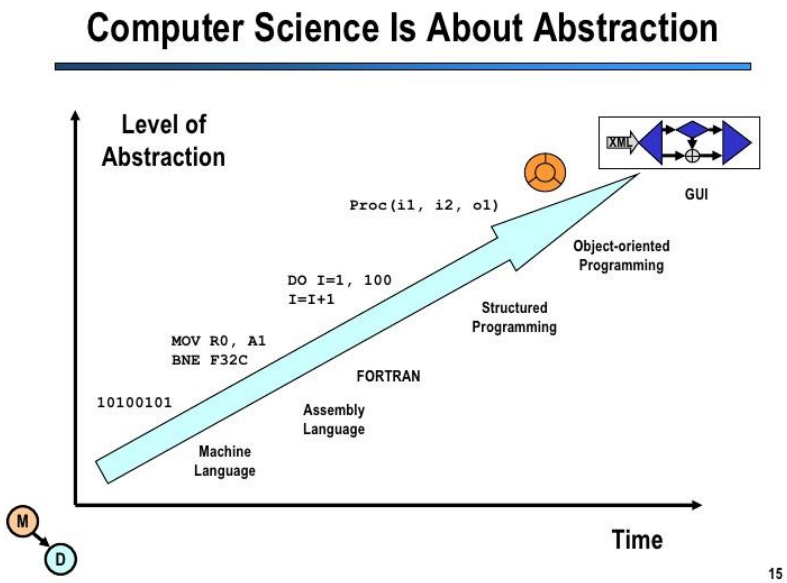

Figure 1. Computer science is about abstraction.

CS teachers must teach students how to think and how to make the computer work in order to teach abstraction. Abstraction skills allow programmers to use induction when they have a lot of code and want to simplify it or when they want to choose one coding solution from many possibilities (Hazzan et al., 2014). Deduction is also a part of abstraction because programmers need to go back and forth in their minds from big picture to small detail iteratively to create elegant code. Bloom's taxonomy, does not adequately provide course designers or instructors with the means to create and evaluate instruction (Fuller et al., 2007). For this reason, it is important to recognize the application of the PKG hierarchy provides a robust definition of abstraction and clear objectives for computer science instruction and assessment. The complexity of both the abstraction concept and the multiplicity of abstraction procedures may ultimately render the PKG Hierarchy too simplistic; however, the PKG hierarchy is concise for educational purposes. To evaluate teachers' experiences and beliefs regarding the instruction of abstraction, I explore the PKG hierarchy, computational thinking, *critical thinking*, and other conceptual and theoretical frameworks. I do not develop theory; instead, I provide a descriptive, robust and informative synthesis of this study regarding the *instruction* of abstraction.

Abstraction in Computer Science Means a Representation

According to Waite (2016), abstractions in computer science are representations that minimize extraneous detail in computer code. Abstractions in computer science are representations, simplifications of larger, more complex code. For the beginning CS educator, it may be tempting to think that teaching students to think abstractly means to

think about things that are not concrete or to simply to use one's imagination. Although the ability to imagine and process ideas is surely foundational for abstraction in computer science, CS abstraction is more complex. Beyond merely thinking and imagining, abstraction is a force that has propelled technology and computers into becoming one of the most necessary aspects of modern life (Abelson et al., 2008). Abstraction is a concept consistently applied in computer science allowing software and technology to become more efficient and easier to program (Colburn, 2000). In Figure 2, the cartoon shows a person expressing their imagination on the screen using the hardware and software at their fingertips. The more CS teachers understand the complexity of abstraction as both concept and skill, the more teachers will deliver effective instruction.

Figure 2. Humorous example of abstraction. Retrieved from https://xkcd.com/676/open educational resource.

Rather than considering abstraction as a teachable skill, some educators might be tempted to allow students to naturally discover abstraction through contextual learning (trial and error) and to assume that abstraction is using thought to create. The "use-modify-create" progression in learning is a typical instructional format in beginning CS courses (Grover & Pea, 2013, p. 40). Indeed, the push to learn to compute through gaming arises from this instructional orientation (Repenning et al., 2015). Learning by doing, or learning contextually, is a natural and important piece of learning human languages (Sanz, 2005). Direct instruction is also necessary for learning human languages (Sanz, 2005). Learning by doing is a tenant of constructionism, predominant CS instructional technique characterized by inquiry-based, collaborative, trial and error learning (Papert, 1980). Students vary in their need for direct and contextual instruction (Sanz, 2005). Although there is no researched correlation between learning human languages and learning computer languages, instructional parallels between the two subjects may assist CS instructors in teaching abstraction. According to Fuller et al. (2007), CS students vary in their learning preferences. Clearer definitions of abstraction and research on the instruction of abstraction facilitate a better understanding of effective instruction, such as direct, contextual, or constructionism.

Colburn (2000) noted that as programming has evolved, the very language of computer programming has become abstracted. According to Colburn, abstraction can be procedural or content oriented, similar to the PKG hierarchy. In the past, programmers considered it a badge of honor to be able to fix unruly programs consisting of binary code (Colburn, 2000). The evolution of Fortran, one of the first computer languages, from

binary code to text was the beginning of the abstraction of programming languages. Unlike math, which requires abstraction to eliminate content, CS uses abstraction to enlarge content (Colburn, 2000); for instance, programmers can define lists, arrays, functions, and variables allowing classrooms, shopping malls, and complex analyses to existing in virtual space. Modern programming allows humans to create more realistic representations of reality without using cumbersome computer commands. Abstraction is the declarative and procedural vehicle that has facilitated the ease and speed of computing (Colburn, 2000). An example of a highly abstracted computer coding language called SNAP is illustrated in Figure 3. SNAP is an example of "drag and drop" computer code that teaches introductory computer science students principles such as recursion and variables which when used correctly demonstrate abstraction. The multi-colored blocks snap into place as they are moved with the mouse on the computer screen. Drag and drop code used in object-oriented programming is representative of much more detailed line code and helps CS students to kinesthetically interact with abstract concepts using the mouse.

Problem Solving and Abstraction

Problems must be analyzed and deconstructed to consider possible solutions. One must deductively determine the main aspect of the problem, then inductively evaluate possible remedies. Both abstraction and problem-solving use deduction and induction, but to different ends. Problem-solving is akin to debugging in computer science, which is identifying why a program is not working and fixing the program. can't begin to offer students instruction on abstraction, another reason to conduct this qualitative study.

Numerical data, as represented in Figure 4, is the foundation of all words and graphical computer representation.

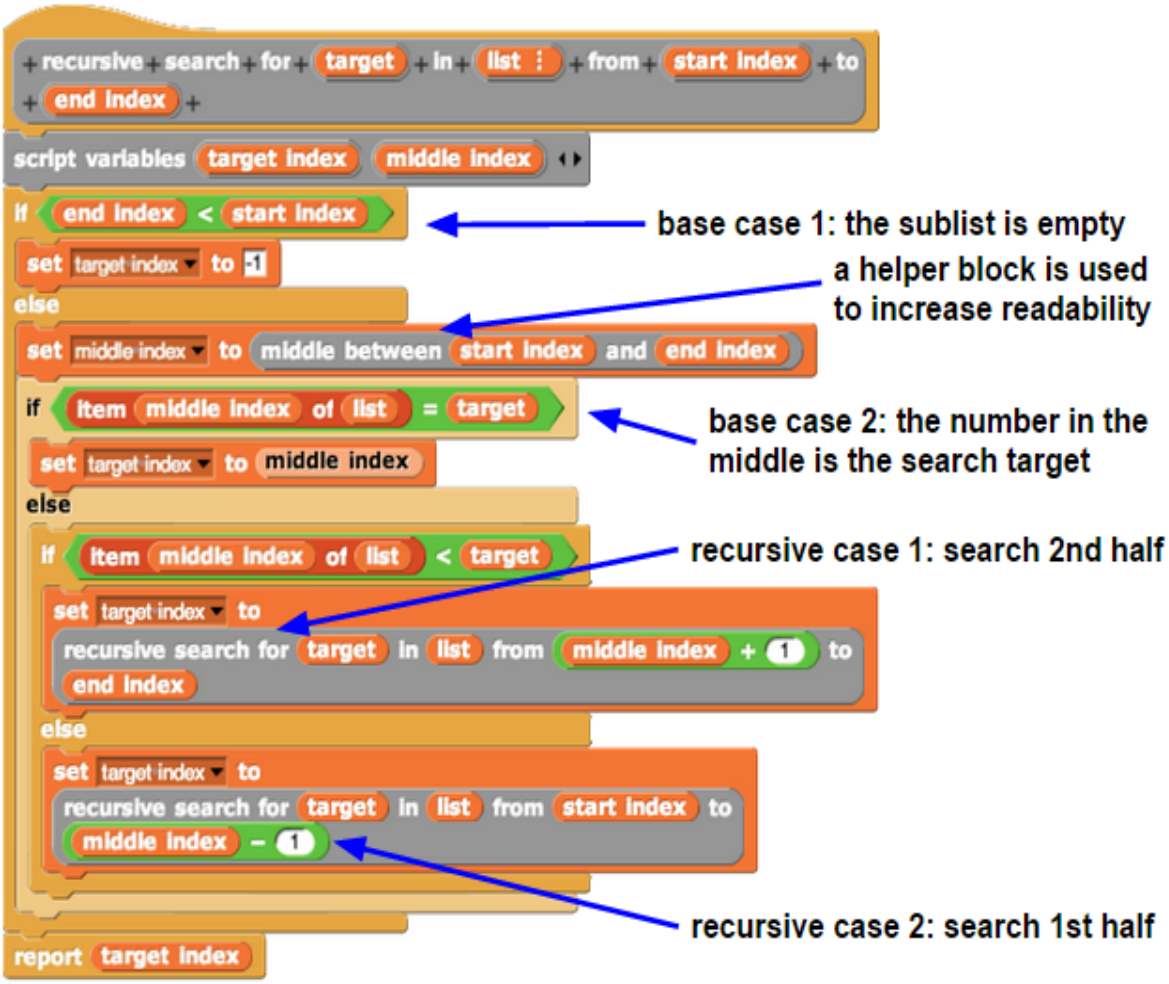

Figure 3. Example of SNAP software, drag and drop code

Abstraction is a symbolic categorization process that allows computer coders to create efficient and effective code. High school students in Greece reported on surveys that along with increased confidence in overall computer science skills after an introductory computer science course; their problem-solving skills also increased (Giannoukos et al., 2013). Abstraction may be taught successfully using problem-solving

instructional strategies. However, teachers who do not thoroughly understand abstraction, The direct link between the abstraction of numerical data and our visual experience of technology is another important reason that CS abstraction warrants research.

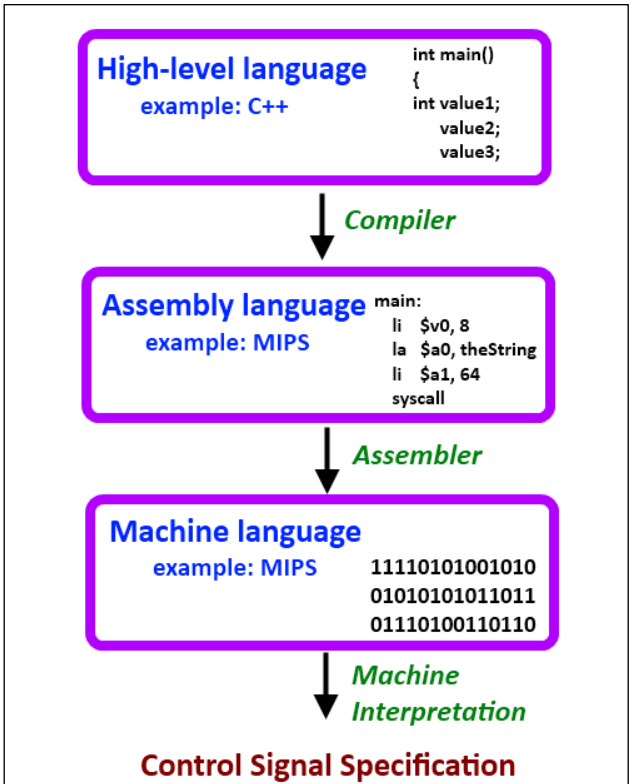

Figure 4. Levels of computer languages.

Problem Statement

There is a significant lack of educational research guiding computer science instruction in K-12 and higher education. Although computer science educators agree that abstraction is a necessary and crucial computer programming skill, they are unsure how to teach abstraction (Armoni, 2013; Brennan & Resnick, 2012; Perrenet, 2010). Even less educational research exists that addresses abstraction in computer science education. Both qualitative and quantitative studies of computational thinking provide insight into

teaching abstraction, although computational thinking is a concept in development and includes a variety of critical thinking skills seemingly better assessed by qualitative research (Armoni, 2013; Brennan & Resnick, 2012; Lye & Koh, 2014; Perrenet, 2010). Researchers examined teaching computational thinking in elementary school; however, none specifically addressed the instruction for abstraction in computer coding (Bers, 2010; Bers et al, 2013; Denner, Werner, & Ortiz, 2012; Lee, 2010; Lee et al., 2014; Lye & Koh, 2014; Wang, Wang, & Liu, 2014). Although researchers have concluded that certain types CS instruction like constructionism, use-modify-create, and object-oriented software assist in the development of computational thinking and computer coding skills, no specific research informing best practices for teaching CS abstraction exists. Therefore, this study focuses on K-12 computer science instruction for abstraction qualitatively examining the teaching experience.

Purpose of this Study

The purpose of this descriptive qualitative inquiry is to examine teachers' experiences determining curriculum, delivering instruction, and designing *assessments* regarding the topic of abstraction in computer science. The instruction of abstraction is the primary phenomenon of this study. Teaching experience is defined as the constructivist experience of teachers using their background knowledge to create curriculum and teach *objectives* successfully (Connelly & Clandanin, 1988; Merriam & Tisdell, 2016; Pappert, 1993).

Research Questions

This study is guided by one general research question: How do teachers decide what effective instruction for teaching abstraction in computer coding is? Additionally, three primary research questions shape the nature of this study,

Research Question 1: What types of instruction do K-12 teachers find most effective for teaching abstraction in computer coding?

Research Question 2: How do teachers determine objectives and competencies for teaching abstraction in computer coding?

Research Question 3: How do teachers assess student abstraction skills in computer coding?

Additional questions regarding the teaching experience are expected to arise during the semistructured interview process (Patton, 2002; Yin, 2014). As I began to gather interview data, I began thematic coding analysis and inductively used "emic" or in vivo structures to code interview data (Maxwell, 2013; Patton, 2002). Miles, Huberman, and Saldana (2013), highly recommended coding, qualitative data analysis, concurrently with data collection. Precoding structures arising from theoretical constructs are called "etic" structures (Maxwell, 2013; Patton, 2002). Using a precoding method based on theoretical constructs indicates a primary deductive method of coding (Patton, 2002). The danger in using a precoding structure deductively is that researcher bias might cause the analysis to be skewed (Maxwell, 2013). For this reason, I needed to focus on creating illustrative descriptions of teaching experiences. It is possible that all aspects of their lives and their experience as teachers may be valuable information. However, the

research focuses on curriculum development, instructional strategies, and assessment methods of abstraction in computer coding. The primary strategic advantage in creating a precoding structure from teacher interviews is that it helps maintain a focused study and not get overwhelmed by extraneous data.

Conceptual Framework

Piaget's reflective abstraction and cognitive development theory (1950, 2014), Papert's constructionism theory (1980), Wing's computational thinking theory (2006), and Vygotsky's (1986) zone of proximal learning were the primary theories used to conceptualize the literature related to the research questions. Learning is inextricably connected to instruction. Piaget's (1950, 2014) theories on reflective abstraction and the development of cognition were used to understand the possibility of teaching abstraction to all grade levels as well as developing the categories of instructional approaches used to teach abstraction. Vygotsky's (1986) zone of proximal learning was used to frame the relationship of the teacher and the student, evaluate collaborative learning, developing critical thinking, and developing advanced thinking skills like abstraction needed for successful computer programming. Papert's (1980) constructionism and Wing's (2006) computational thinking theories were used to understand current and enduring approaches to computer science instruction. Because abstraction involves both deductive and inductive thinking, the theories of Bloom (1956) and Marzano and Kendall (2007) informed research on critical thinking, problem-solving, and computational thinking. Additionally, theory from Fichte (as cited in Whistler, 2016) regarding abstraction and potentiation, Floridi's (2011) theory that computers facilitate epistemological

development, and Gobbo and Benini's inforg theory (2012) regarding the abstract essence of the human-computer relationship provide deeper understanding about the complex nature of the concept of abstraction that teachers must understand in order to teach. A more detailed analysis can be found in Chapter 2.

The development and nature of thinking bridges into psychology which is why Piaget's (2001, 1980) and Vygotsky's theories informed Research questions 1 and 2. Cognitive research demonstrates that 30% to 35% of adolescents reach the formal operation stage (defined by Piaget) in which the cognitive ability to abstract occurs - some adults never reach the formal operation stage (Armoni, 2012; Kramer, J., 2007). Recent literature challenges abstraction as a cognitive process developing later in the teenage years or as an adult (Braithwaite et al., 2016; Novack et al., 2015; Rittle-Johnson & Schneider, 2014). Piaget later recanted the ability of young children to learn abstraction calling the thinking process reflective abstraction. Nevertheless, Piaget's stages of cognitive development and reflective abstraction provide a context for examining abstraction skills from elementary to post-secondary grades, as well as from low to high abstraction levels (Armoni, 2012; Kramer, J., 2007). Finally, Vygotsky's (1986) zone of proximal learning theory informed Research Question 3 and may explain multiple pathways for children in learning and expressing abstraction.

Papert's (1980) constructionism and Wing's (2006) computational thinking theories, which informed Research Questions 1, 2, and 3, illuminate the ways and primary content needed to shape thinking for learning computer science, as well as the ways students demonstrate computer science knowledge. Constructionism is a prevalent

theory in CS education and has resulted in student-led guided inquiry instructional strategies (Armoni, 2013: Bers et al., 2014; Denner, Werner, & Ortiz, 2012; Fessakis, Gouli, & Mavroudi, 2013; Harlow & Leak, 2014; Kazakoff & Bers, 2012; Lee, 2010; Papert, 1980; Wang, Wang, & Liu, 2014). Computational thinking has been used to guide curriculum and learning experiences resulting in thinking that uses computers to solve problems (Anton & Barany, 2013; Bers, 2010; Bers et al, 2014; Lee, 2010; Lee et al., 2014; Lye & Koh, 2016; Pellas & Peroutseas, 2016; Bucher, 2016; Sanford & Naidu, 2016; Zhong et al., 2016; Czerkawski & Lyman, 2015; Shell & Soh, 2013; Wing, 2006). Both constructionism and computational thinking can be used to inform assessment choices. Papert (1980) adapted constructivism in computer science calling the concept constructionism, meaning the construction of knowledge to create objects in the world. Acquiring mastery of computer coding concepts and procedures are both necessary aspects of learning abstraction (Zendler & Klaudt, 2012).

Nature of the Study

The basic qualitative descriptive inquiry is an effective research strategy for initial investigation in educational subjects with many variables (Merriam & Tisdell, 2016). This study explores how teachers generate the knowledge of their instruction, curriculum, and assessments, and how they construct meaning regarding the teaching of abstraction in computer science via constructivism (Patton, 2002). In this study, 12 teachers were interviewed twice. Initially, the study required each teacher to submit five artifacts of student coding exhibiting abstraction or the progression towards abstraction. Although mitigating factors, which will be explained in Chapter 3, necessitated the change in data

collection to include two teacher interviews and analytic researcher memos after each interview. Because teachers have a variety of instructional goals for teaching abstraction, different students, ages of students, learning environments, and curricula, there are a plethora of variables that cannot be controlled. Computer science courses may be taught as a sub-discipline of Science, Math, or Technology because many states do not have curricular requirements for CS. Additionally, CS teachers use many different computer software programs to teach coding. The K-12 annual assessments in most states do not test computer science competencies. A lack of quantitative data, consistent curriculum, and teacher case variation necessitate investigating the instruction of abstraction using a basic qualitative format.

Definition of Terms

Abstraction is the use of a variety of algorithms that shorten, hide, and simplify computer code creates elegant and efficient computing. (Armoni, 2013).

Computational thinking is applying computational solutions to computer coding or computational devices to solve problems (Wing, 2006).

Critical thinking is the ability to process information in a variety of ways including synthesis, analysis, and metacognition (Faccione, 1996).

Instruction is the activity and delivery of experiences designed to affect learning (Ambrose et al., 2010).

Assessment is the activity and process of evaluating learning (Ambrose et al., 2010).

Objectives are specific skills and tasks designed to produce a certain learning effect (Biggs & Collis, 2014).

Standards and Competencies are broad learning goals, skills, and thinking abilities that guide specific course objectives (Biggs & Collis, 2014).

Recursion is an iterative algorithmic process that simplifies and shortens computer code (Brookshear et al., 2012).

Variables are representative symbols that simplify computer code and represent larger concepts (Brookshear et al., 2012).

Event handlers are algorithmic processes that simplify computer code (Brennan & Resnick, 2012).

Assumptions

Assumptions are worldviews or beliefs that can bias a researcher's observations and interpretations (Corbin & Strauss, 2015). There are three major categories of assumptions in this study regarding teacher honesty, the importance of abstraction, and abstraction as a thinking skill in student coding. It is assumed that teachers provided honest answers and comments in the interviews. This study also implies that in accordance with computer science education, abstraction is a necessary skill (Armoni, 2013; Brennan & Resnick, 2012). The order of the importance of computer science thinking skills, such as sequencing, using persistence, and implementing conditionals to learn about computer science is unknown. Abstraction is not a necessary skill to begin learning computer science; however, abstraction is necessary to produce effective elegant

computer code (Armoni, 2013). As the field of computer science education evolves, the validity of assumptions is important to reflect upon and evaluate (Denning et al., 2016).

Scope and Delimitations

The purpose of this inquiry is to examine teachers' experiences determining curriculum, delivering instruction, and designing assessments regarding the topic of abstraction in computer science. Identifying and describing teachers' experiences with abstraction can provide useful information for other teachers, educational researchers, software developers, policy-makers, and additional professionals involved in CS education. Because teachers are the primary facilitators of education, investigating their experiences will inform future studies.

This inquiry is bounded by specific aspects of the teaching experience, namely curriculum, instruction, and assessment, in order to identify useful teaching strategies and variables for future research. Additionally, using purposeful sampling and limiting the participants to four elementary, four middle school, and four secondary teachers, leads to fewer variables, but make the data richer.

Transferability of the findings from this study may inform the instruction of abstraction across the United States. Recruiting teachers from a variety of states aides in the transferability of results. The knowledge gained from comparing and contrasting the experiences of teachers in a variety of grade levels may provide teachers with ideas about appropriate grade-level instruction. Insights from this study may also help policy-makers and educational researchers write grade-level appropriate standards.

Limitations

Potential weaknesses in this research proposal can be attributed to the qualitative researcher and the small qualitative sample. Human bias from my perceptions (as the researcher) of the literature review, the theoretical framework, precoding structures, and perceptions of validity, and naiveté regarding threats to validity limit this study. Also, the small number of participants required for a thorough qualitative study naturally limit generalizability (Stake, 2006). Hopefully, this research will generate future quantitative research that will utilize greater population sizes and more objective variables.

Significance

Better computer science instruction results in more people in all professions having the technical knowledge to solve computational problems (Abelsen, Ledeen, & Lewis, 2008). An informed digitally literate citizenry may be able to be more creative with technology and make wise choices about technology for future generations. Metaliteracy, a concept proposed by Jacobsen & Mackey (2013), encourages the use of critical thinking and metacognition in the development of literacy needed to navigate digital text and sources. As technology continues to increase the scope of education, teachers need guidance, such as metaliteracy, about teaching technology, specifically how to offer effective computer science education with specific and appropriate cognitive objectives. Aligned curricula for computer science are in development, and there is a need to understand what type of instruction is most useful (CSTA, 2015; CS10K, 2015). Some predefined curricula from Code.org, Khan Academy, and others are taught at the

elementary, middle school, high school, and college levels because of a lack of educational resources and a lack of trained teachers.

This study will inform the developing concept of computational thinking by providing insight about when and how students learn abstraction (a component of computational thinking). Computational thinking is the dominant theory guiding computer science curricula and suggests a complimentary instructional approach to critical thinking. Abstraction is one component of computational thinking. This study will also help inform the development of computer science curricula.

Summary

The process of teaching like the process of computer science can be reduced to input and output. If abstraction, an essential skill needed for computer programming is the output, what types of instruction at what ages provide the optimal output or evidence of learning? Computer science education is a developing field (Wagner, 2013). By examining the perspectives of teachers in the field, this study will help to identify factors of effective instruction of abstraction that can be quantitatively studied, as well as additional educational variables, such as grade-level appropriate instruction. The appeal of computational thinking is that it is a catchphrase for a necessary and straightforward idea that people need to use computers more practically in all disciplines. The reality of computational thinking is that it is a vast subject, and we have only begun to uncover the many ways of thinking that require development to use computers effectively. Examination of the instruction of abstraction will further our understanding as educators about teaching computer science effectively. In Chapter 2, I comprehensively explore the

existing gap in literature affirming a qualitative examination of abstraction in K-12 instruction.

Chapter 2: Literature Review

If all people learn to use computers as the tools they were meant to be, the notion of digital literacy will expand from simply navigating software, such as Microsoft Excel, and evaluating Internet sources to programming computers and designing technology that solves human problems (Abelson et al., 2008). Programming computers, as opposed to operating computers, facilitates human creativity and knowledge. For multiple reasons, such as the pervasiveness of abstraction in today's technology, as well as the complexity of the subject the instruction of abstraction in computer science may be difficult for teachers to develop or broach. Abstraction is a necessary aspect of being a competent computer programmer, but because the subject of abstraction has been poorly defined and researched, the instruction of abstraction lacks guidance (Armoni, 2013). Perhaps because abstraction is embedded in multiple layers of technology explaining abstraction may appear overwhelming.

Colburn (2000) mentioned computers are essentially abstractions of human thought, expanding content and capability. Because the subject of the instruction of abstraction leaves many questions, this study will help to illuminate current pedagogy and future research. The purpose of this descriptive qualitative inquiry is to examine teachers' experiences determining curriculum, delivering instruction, and designing assessments regarding the topic of abstraction in computer science. There may be different pathways to learning abstraction, similar to the theoretical concept proposed by Fuller et al. (2007). In this literature review, I compare and contrast abstraction with computational thinking and critical thinking. I also show how there are implications for

teaching abstraction from research on the instruction of computational thinking and critical thinking. A discussion of philosophical and theoretical constructs regarding abstraction provides a context for the research of this complex topic.

Literacy Search Strategy

I primarily used the ERIC and Sage databases, as well as Google Scholar, to search keywords limiting the review to abstraction in computer science, instruction of abstraction in computer science, and dissertations and peer-reviewed articles published between 2013 and 2017. CT research usually indicates abstraction as a component of computational thinking. Critical thinking also contains aspects of abstraction, namely deduction and induction (Kong et al., 2014; Marzano & Kendall, 2007). Therefore, I used computational thinking and critical thinking studies for the bulk of this literature review. The following are the main keywords used to generate the literature review: *computer science coding + children, computer science instruction + children + coding, computer science + language acquisition, computer science assessment + children, computer science principles + instruction + age, computer coding + age, computer coding + elementary, dissertations + computer science instruction, comparing coding with different ages, computer coding developmental age, and computer coding teaching vocabulary.*

I found 116 relevant peer-reviewed scholarly publications from thousands of studies were found using the keywords: instruction, computer science, abstraction, math, computational thinking, and critical thinking. Seven of the 116 relevant publications

offered only commentary, curricular or instructional suggestions, and literature review; the remaining publications were research studies.

Thirteen studies addressed how students learn abstraction in computer science (Armoni, 2013; Carbonaro, Szafron, Cutumisu, & Schaeffer, 2010; Colburn & Shutte, 2007; Cooper, Perez, Rainey, 2010; Csneroch & Math, 2015; Guzdial, 2011; Katai, Toth, & Adjani, 2014; Perrenet, 2010; Reuker et al., 2013; Saeli et al., 2012; Shirazi et al., 2013; Wang, et al., 2014; Weintrop & Wilensky, 2014). Perrenet's (2010) use of surveys and interviews provided data on college students' understanding of the diverse level of abstraction. Fessakis, Gouli, and Mavroudi, (2013) and Harlow and Leak (2014) investigated how elementary students learn abstraction and computational thinking via video observation. Armoni (2013) surveyed high school students and found they were capable of basic levels of abstraction.

After presenting at the 2017 Computer Science Teachers Association conference, colleagues from England alerted me to the presence of two dissertation studies on abstraction in computer science, one finished and one in progress. I conducted dissertation searches using the Walden University library and all of the keywords listed previously but did not find any dissertations on the instruction or learning experience of CS abstraction. Teague (2015) conducted a dissertation mixed method study of undergraduate Information Technology (IT) students and applied their mastery and experience of learning abstraction to Piaget's learning theory. In one semester, novice programmers demonstrated proficiency with the sensorimotor and preoperational reasoning but did not achieve proficiency in concrete operational stage thinking. Teague

noted that the most mature Piagetian stage, formal operational reasoning, was not considered in depth in her study and concluded difficulty in the development of abstract thinking limited novice programmers from achieving programming skills. Waite, Curzon, Marsh, and Sentence (2016) recommended using visual instructional strategies, such as graphic organizers, concept maps, and storyboards for teaching abstraction to young learners. Waite is currently working on her dissertation in which she is studying the instruction of abstraction in elementary computer science education. Teague (2015) and Waite et al. (2016) illustrated that related research has focused on student learning and not the teaching experience of abstraction.

Few researchers have investigated teaching in computer science, and their research has not examined teaching abstraction. Researchers in only one study addressed computer science teachers providing descriptive statistics from surveys to ascertain their pedagogical content knowledge (Saeli, Perrenet, Jochems, Zwaneveld, 2012). Many STEM teachers are not teaching computer science and consequently not teaching CS abstraction. None of 38 science teachers who won the Presidential Award for Excellence in Science Teaching surveyed included computer coding in their courses (Hakverdi-Can & Dana, 2012). Scant research does not provide helpful information about CS teachers. Instructors as students were the subjects of one study that indicated they were satisfied with an online AppInventor introductory course (Hsu & Ching, 2013). As the push for training CS teachers advances, CS educational research is being collected.

Beginning in 2004, CS educational and computer science experts at the Exploring Computer Science (ECS) Project developed teacher professional development modules

designed for K-12 educators focused on expanding AP Computer Science teaching. Margolis et al. (2011) created the ECS Project after reporting on gross racial and gender inequities in computer science education. ECS teacher training has been successfully implemented in Los Angeles and Chicago public schools (Ryoo et al., 2014). ECS teacher training curriculum emphasizes including problem-solving and critical thinking, specifically teaching the analysis of abstraction. ECS researchers report that the top three CS instructor practices include connecting computing with equity and everyday issues, encouraging collaboration, and using guided inquiry to facilitate metacognition and computational thinking (Ryoo et al., 2014). ECS teachers asked more questions about knowledge acquisition and analysis, and fewer questions about application and evaluation. Abstraction is an important aspect of teaching CS computer science, and additional research into the understanding of teachers' experiences will facilitate better teaching practices.

Cooper, Perez, and Rainey (2010) recommended that the role of the teacher in the process of learning abstraction should be studied. It is essential to understand if the output, or student learning which most studies examine, is happening. Alternately, by addressing learning input, or instruction, researchers can guide teaching best practices. Precedent exists from this literature review for interviewing and surveying teachers to obtain information regarding their teaching experience of abstraction in computer science. Philosophical and theoretical viewpoints also inform current understanding of the instruction of abstraction.

Abstraction and Philosophy

In order to understand the breadth of context for abstraction existing in computer science, I begin this literature review with an examination of theory and conceptual frameworks from philosophy, psychology, and computer science. According to Flick (2013), theory and conceptual frameworks can be used to illuminate participant perspectives (p. 48). Famous philosophers such as Kant, Hegel, and Fichte expounded on the nature of abstraction, existence, and thought. Philosophers have debated how reason evolves as an abstraction of thought allowing humans to transcend experience (Whistler, 2016). According to Fichte (as cited in Whistler, 2016), abstraction of thought occurring from induction (observation to theory) and subsequently deduction (theory to confirmation of reason) is either potentiation or depotentiation. Abstracting can be used to pull out the essential nature of something and expose it, making it more potent, or abstraction can be used to delve deeper into the nature of something deconstructing it, depotentiating it, and to make it less "potent". The conventional use of abstraction in computer science is like Fichte's potentiation, whereby abstraction preserves the essential nature of the computer program (Gobbo & Benini, 2012). Instructors may find that assisting students in honing their metacognitive skills will assist the student in thinking about thinking (metacognition), leading to thinking about coding efficiently because coding essentially represents thought. Therefore, philosophy may be a useful subject assisting teachers and students in understanding abstraction in computer science.

Teaching abstraction may benefit from discussions about the nature of the human-computer relationship. Ben-Ari (2001) pointed out that most students do not have an

explicit model of the computer. A computer is not an animal that is a soft-tissue being like us and thinks like a human. A computer metaphorically has a different body than a human. However, a computer does have structure and function, very similar to language that has words, syntax, and grammar. In fact, computer programming "symbolically represents algorithms as numbers" basically hiding information, the very act of abstraction (Gobbo & Benini, 2012, p. 4). Abstraction can be found in both human thoughts and subsequently in computer programming. The computer requires programming code to function, thereby becoming an ontological extension of thought (Ben-Ari, 2001; Gobbo & Benini, 2012). Computers represent and express our thoughts, thereby seeming human.

Abstraction becomes more complex because the inforg, or human-computer interaction, as an object produces more levels of organization and explanation, or levels of abstraction (Gobbo & Benini, 2012). Coding is a part of the human-computer interaction, e.g. choosing printing options on a printer and accessing cloud-based services (Gobbo & Benini, 2012). Computers have always been tools to extend the thinking process and knowledge. Computers also facilitate epistemological development or the development of knowledge (Floridi, 2011). As computers become easier for humans to program, Fichte's 19th-century notion that abstraction produces reason indifferent to the self seems to have manifested in the form of the human-computer inforg. The level of abstraction may be as simple as an app on a phone or as complex as a robot that learns how to help older adults. Both computer examples require elegant and efficient computer code, and both are extensions of human creativity serving human needs. The complexity

of human and computer interactions makes the concepts and skills needed to understand, teach, and learn abstraction challenging.

Abstraction and Psychology

Piaget (1950, 2014) asserted that the development of abstract thinking, which he defined as the ability to realistically imagine a problem and a solution, occurs around age 11. The instructional implications for abstraction would be that teachers focus on knowledge acquisition and algorithmic procedure, like memorizing math facts and computational procedures, until middle school when students have learned the coding process and can think of ways to apply and use both deductive and inductive reasoning. It might be too much to assume that elementary students could demonstrate the independent application of abstraction in computer coding. Elementary students might be able to model abstraction at the Perrenet, Kassenbrood, and Groot (PKG) execution level, or algorithm level, but not independently demonstrate abstraction (Perrenet, Groote, & Kassenbrood, 2005). More research is required to understand student learning and capacity for abstraction.

Reflective abstraction, coined by Piaget (cited in Mudrikah, 2016), has been used to organize math instruction and provide insight into instructional best practices for teaching abstraction in computer science. Capetta and Zolman (2013) recommended using peer instruction, reflective thinking exercises, and instructor dialogue to stimulate reflective abstraction in calculus students. Open-ended questions and story problems were found to influence the development of abstraction among Thai 4th grade math students (Promraksa, Sangaroon, & Inprasitha, 2014). African high school students

exhibited more creativity in solving math problems when instructed with a learning process emphasizing doing, reflecting, thinking, and applying concrete experience, reflective observation, abstract visualization, and active experimentation (Chesimet, Githua, & Ng'emo, 2016). Cognitive disequilibrium, another Piagetian concept, could also be used to encourage abstraction given that cognitive disequilibrium initiated critical thinking in over 400 college students surveyed (Cole & Zhou, 2014). Collaborative, inquiry-based instruction emphasizing metacognition and critical thinking development appear to be effective instructional practices for teaching abstraction in math.

Vygotsky and Teaching Critical Thinking through Interpersonal Learning

When exposed to examples of inductive and deductive reasoning, children learn the concepts of abstraction, critical thinking, and computational thinking. Vygotsky's (1986) interpretation of how thought develops through language is an important reminder to teach vocabulary and concepts similar to how children learn language - verbally, interpersonally, and repetitively. Vygotsky's zone of proximal development helps to explain how children could be learning without being able to produce evidence of such learning. According to Vygotsky (1978), "the actual developmental level characterizes mental development retrospectively, while the zone of proximal development characterizes mental development prospectively" (pp. 88-87). Following Vygotskian theory, students learn more than they are capable of expressing and students learn best socially.

Computer science and critical thinking research demonstrate the validity of the zone of proximal learning. Over 85 medical school faculty surveyed agreed that critical

thinking was an ability that could be learned and required interpersonal interaction, insinuating that abstraction would also require collaborative learning (Rowles et al., 2013). Interpersonal or collaborative learning focusing on controversial topics has been shown to increase critical thinking in honors college students (Cargas, 2016). Another application of interpersonal learning involves mentors. Middle school computer science students in a New York City after-school robotics program learned how to build robots from adult mentors, persistence, STEM instruction, and critical thinking skills (Groome & Rodriguez, 2014). Even if children might be too young for direct instruction, indirect instruction via stories or demonstrations, even anthropomorphizing computers could be a way for them to absorb and model the vocabulary and conceptual means to think abstractly, critically, and computationally.

Vygotsky's inner speech, or "talking" to oneself, internal dialogue, begins around age 7 (Flavell et al., 1997). The development of inner speech, the beginning of metacognition, is necessary for abstraction, critical thinking, and computational thinking (Mahn, 2012). Elementary students may be able to begin to understand abstraction, which is one reason why this study includes elementary teachers. Huang et al. (2016) illustrated how math instruction for multiplication could be simplified and made more efficient for middle and high school students using collaborative learning and metacognition. CS educators recommend paired programming for teaching computer science (Porter et al., 2013). Speech helps develop thought; research shows that computer science instruction highlighting collaborative learning and paired programming validates Vygotsky's theory (Harlow & Leak, 2014; Mahn, 2012). In a large quantitative study of 525 high school

students, Asku and Korkulu (2015) provided evidence that critical thinking instruction is correlated with math competency and students must have a positive attitude to be successful in math. If the instruction that focuses on developing an inner dialogue, metacognition, and critical thinking facilitates math competency, similar instructional techniques could help students learn abstraction.

Although collaborative social learning may not be entirely correlated with learning critical thinking, computational thinking, and ultimately abstraction, incorporating verbalizing thoughts might be helpful. Peer-led team learning (PLTL) and critical thinking gains were not correlated in a study conducted with undergraduate biology students; however, PLTL was an instructional strategy positively related to increased self-efficacy and social skills (Synder & Wyles, 2015). Additionally, critical thinking was not correlated with student social presence in Korean online courses, possibly indicating that actual voice or speech may indeed be a necessary critical thinking component (Costley, 2015). Undergraduate students who learned to detach and listen were effectively engaged in critical thinking and group decision-making (Dwyer et al., 2014). Process-oriented guided inquiry learning (POGIL) is an instructional technique originating in 1994 designed for chemistry education (Hu, et al., 2016). POGIL protocol advises using small group collaborative learning where teachers act as facilitators asking questions to stimulate students to construct meaning, solve problems, and develop critical thinking. In surveys, 32 CS secondary and college educators indicated strong agreement with POGIL instruction improving student engagement, interpersonal skills, active learning, and CS learning outcomes. Actively engaging, verbalizing, with other students

or with the instructor, may be important instructional activities that facilitate learning abstraction.

Situating Abstraction within Computational Thinking

Brennan and Resnick (2012) proposed a framework of thinking that includes abstraction as a subskill. As stated previously, various levels of abstraction must be applied to make computers express thought through design. The specific levels of abstraction are subjective. Brennan and Resnick, professors at MIT, have offered a framework for computational thinking that includes computational concepts (sequences, loops, parallelism, events, conditionals, operators, and data), computational practices (being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing), and computational perspectives (expressing, connecting, questioning). Brennan & Resnick's framework for computational thinking, concepts – practices – and perceptions, is akin to the PKG hierarchy (Figure 5). Computational practices are similar to the execution and program levels of abstraction. Object and problem levels require an understanding of computational concepts. Computational perspectives are also necessary for the object and problem levels of abstraction. Although Brennan & Resnick's framework has not been used as much as Wing's more simplified definition of computational thinking, the delineation of multiple ways of thinking needed for computational thinking provides further understanding of a working definition of abstraction.

Computational thinking (Wing, 2006) has taken on multiple meanings, and is the most noted theoretical framework and rationale for many studies (Anton & Barany, 2013;

Bers, 2010; Bers et al., 2014; Czerkawski & Lyman, 2015; Lee, 2010; Lee et al., 2014; Lye & Koh, 2016; Pellas & Peroutseas, 2016; Bucher, 2016; Sanford & Naidu, 2016; Shell & Soh, 2013; Zhong et al., 2016). According to Wing (2006, 2008), head of the computer science department at Carnegie Mellon University, computational thinking requires abstraction and automation.

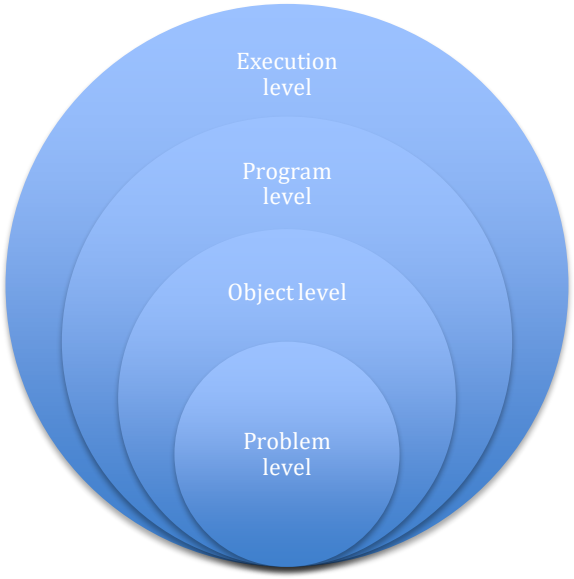

Figure 5. PKG Hierarchy

Although computational thinking (CT) has become pervasive in computer science education, it is also a way of thinking necessary in the 21st century (Cooper, Perez, & Rainey, 2010). Computational thinking enshrines abstraction as one of its primary components (Grover & Pea, 2013). Cooper, Perez, and Rainey (2010) attribute the development of abstraction to the externalization of human knowledge interfacing with computers, a similar interpretation of Floridi's (2008, 2011) works. According to Floridi (2008), human beings are turning into information organisms being increasingly

dependent on computer information in an infosphere (p. 2). Computational thinking will help students determine the difference between computers and humans, especially with the advent of artificial intelligence (essentially an abstraction of human intelligence) and machine learning. Because abstraction is a necessary aspect of CT, this proposed study may provide valuable insight into the instruction of both computational thinking and abstraction.

Comparison of Abstraction, Computational Thinking, and Critical Thinking

Academic articles begin with an abstract or a condensed summary of the broad body of knowledge. In a sense, a car, a microwave, and a computer are tools that we operate by understanding abstraction and not with knowledge of the complex mechanism and coding of the machines (Brookshear, 1997). Applying levels of abstraction enables people to program complex computer operations that would otherwise make computer programming overwhelming (Brookshear, 1997). Similarly, student computer programmers can follow the procedures indicating abstraction, but may not fully understand the concept of abstraction enough to create technological innovation.

According to Dale and Walker (2007), abstraction as a model allows programmers to remove extraneous detail and make the code more efficient. Computer science educators are in the process of defining abstraction (Armoni, 2013; Brennan & Resnick, 2012; Fuller et al. 2007). The thinking skills of induction and deduction are two common ideas in their definitions. Induction and deduction are also critical thinking skills defined by Marzano and Kendall (2007) as specifying and generalizing roughly equivalent in Bloom's taxonomy as analysis, synthesis, and evaluation. By comparing

and contrasting the definitions of abstraction given by the computer science education scholars listed above as situated in the critical thinking taxonomies of Marzano and Kendall and Bloom, essential elements of learning abstraction are identified.

Wing (2006), Brennan and Resnick (2012), Armoni (2013), and Fuller, et al. (2007) offer varying definitions of abstraction sometimes included in computational thinking. To further complicate the issue, Armoni offered a synthesis of important scholarly constructs of abstraction in which he avoids defining abstraction and instead utilizes the PGK hierarchy to support a framework for teaching abstraction. The PGK hierarchy describes four levels of abstraction. First, the execution level involves expressing abstraction thinking through the algorithms needed to run computers. The next level is the program level, which requires applying algorithms to a variety of programs, essentially making computers do similar things with different programs. The next level of abstract thinking involves perceiving an algorithm as an object allowing computer programmers to simplify code and make it elegant. Finally, abstraction on the problem level expresses the solution via computer. Armoni further simplified abstraction by adding that it is understanding of the process and problem in size from large to small (and vice versa) as well as in meaning from how to what (and vice versa).

Abstraction is mentioned in the thorough synthesis of learning taxonomies by Fuller, et al. (2007), although the specific taxonomy they developed for computer science courses is based on Bloom's taxonomy (because of its widespread prevalence in computer science education research) and indirectly addresses abstraction. On one side of the taxonomy are activities for producing (apply and create) and on the other side are

cognitive domain activities for interpreting (remember, understand, analyze, and evaluate). The taxonomy is based on the fact that reading computer code, understanding code, and writing code are two different processes, similar to reading and writing a language. Abstraction is one of the skills required for the production activities of applying and creating; abstraction is utilized in all the categories of interpretation. Therefore, this taxonomy does not explicitly recognize abstraction and does not adequately offer a means to evaluate student abstraction skills. Nevertheless, the taxonomy of learning computer science does suggest something novel regarding critical thinking and learning abstraction.

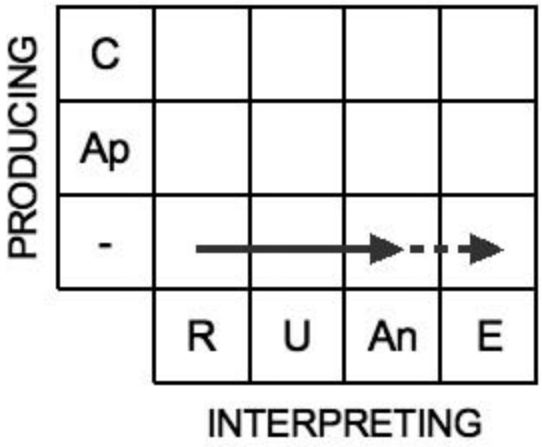

Figure 6. The pathway of a student who attains only theoretical competency (Fuller, et al. 2007).

Fuller et al. (2007) posited that students use multiple pathways for producing and interpreting computer coding to attain higher order thinking. Many subjects are learned by interpreting and analyzing; however, computer coding also requires practicing and applying knowledge. Figure 6 illustrates how students learn computer coding by remembering (R), understanding (U), analyzing (An), and evaluating (E), purely through

cognitive channels. Figure 7 shows how other students learn computer coding through remembering (R), understanding (U), applying (Ap), and creating (C).

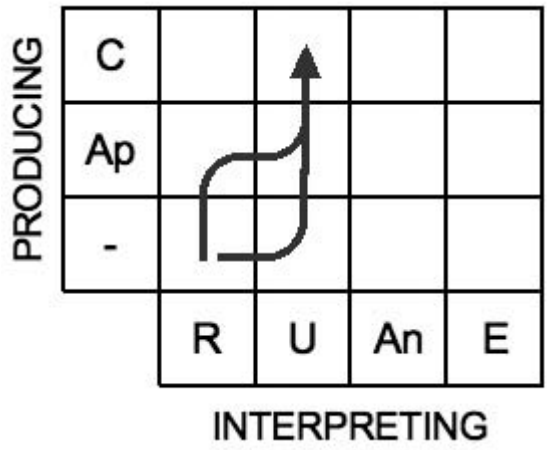

Figure 7. The pathway of a student who attains only practical competencies (Fuller, et al., 2007).

English literature teaches students how to analyze and critique, but rarely are students required to write (create) a novel. In computer science students must produce software; therefore, the simplicity of Bloom's taxonomy which focuses on conceptual understanding does not adequately support computer science course design or measurement of course objectives which also requires procedural understanding. Because Bloom's taxonomy is used so readily, educators often assume that the higher levels of Bloom's, i.e., analyzing, synthesis, and evaluation create a better learning experience. The beauty of the taxonomy for learning proposed by Fuller, et al. (2007) is that they recognize multiple pathways for developing critical thinking and achieving proficiency in computer science illustrated in Figure 8.

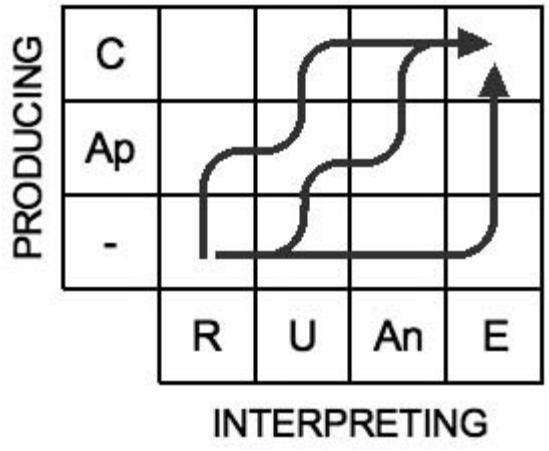

Figure 8. The goal "Create or Evaluate" can be attained through multiple pathways (Fuller, et al.).

The other theoretical constructs for learning abstraction, computational thinking, and critical thinking, do not recognize students might have varying cognitive pathways for achieving CS proficiency (Armoni, 2013; Brennan & Resnick, 2012; Marzano & Kendall, 2007; Wing, 2006).

Critical thinking defined by philosophy usually emphasizes the "nature or quality of thought"; whereas, critical thinking defined by psychology stresses cognitive processes (Atabaki et al., 2014). In a complex taxonomy of learning including the cognitive, affective, psychomotor, and self-system domains, Marzano and Kendall (2007) identified abstraction as the process of generalizing and specifying akin to Bloom's analyzing, synthesizing, and evaluating. Stating that inferences can be both inductive and deductive, Marzano and Kendall (2007) have defined generalizing as retroduction, a process more like induction but requiring both induction and deduction during the process. Additionally, generalizing involves pattern recognition, the ability to focus on specifics,

identifying connections that explain patterns (Marzano & Kendall, 2007). Focusing on the words induction and deduction may help inform the instruction of abstraction.

The theoretical constructs suggested by Armoni (2013), Wing (2006), and Bloom's taxonomy (Fuller et al., 2007) as well as the producing/interpreting taxonomy (Fuller et al., 2007), concentrate solely on the cognitive domain of learning although Fuller et al. (2007) admit that the affective domain is a critical part of computer science education because students are expected to create professional soft skills for evaluating ethical behavior, evaluating the ethical implementation of technology, and facilitating clients. Brennan and Resnick (2012) and Marzano and Kendall's (2007) theoretical constructs include the affective domain. Marzano and Kendall's new taxonomy of learning also includes the self-system maintaining that personal beliefs and metacognition are the most important aspect of achieving critical thinking.

Turkish researchers found a positive correlation between self-confidence in reasoning ability and critical thinking after testing 400 K-12 teachers, providing testimony to the probable positive relationship between the self-system and development of critical thinking (Emir, 2015). Affective and emotional domains associated with the self-system also influenced critical thinking acquisition in Russian, advanced English Language, Science, and Social Studies courses (Vanicheva, Kah, & Ponidelko 2016). Kwan and Wong (2014) found in studying over 900, ninth grade, Hong Kong, humanities students that critical thinking resulted from the interaction between cognitive learning strategies and student motivational beliefs. No clear correlations between learning styles and critical thinking were found in college nursing students; however, researchers

recommended more studies (Andreu et al., 2015). Ultimately, the affective domain and self-system theoretical constructs may be more valuable for the instruction in all disciplines; whereas, the cognitive domain theoretical constructs may be more relevant for assessment.

Regarding structure, the taxonomies of Armoni (2013), Fuller et al. (2007), and Marzano and Kendall (2007) are most convincing because each synthesizes the work of multiple theories, definitions, and learning taxonomies. Marzano and Kendall elaborated on their new taxonomy for learning in a book. Wing (2006), a highly respected professor of computer science and head of her department at Carnegie Mellon, offered her definition of computational thinking in an opinion editorial piece, which minimizes her definition's robustness. Brennan and Resnick (2012) do not provide a literature review for their expanded definition of computational thinking, instead simply offer anecdotal qualitative evidence from student interviews and artifacts. To gain a thorough sense of the complex nature of learning abstraction, the viewpoints of researchers regarding critical thinking, computational thinking, and abstraction are all relevant, some more illuminating and credible than others.

Abstraction is a skill used in many disciplines and is a skill used in computer coding to modularize and manage complex coding commands. Abstraction skills allow computer coders to use induction, for example, when they have a lot of code and want to simplify it or when they want to choose one coding solution from many possibilities. According to Faccione & Gittens (2016) deduction is ideological reasoning or top-down thinking. Deduction is also a part of abstraction because students debug their programs

and examine their computer programs' goals compared to the actual program function and execution. Induction or bottom-up empirical reasoning is needed when wading through the code to find pieces of code from other software languages or published programs that could be used to accomplish the goal of the program. Bloom's taxonomy, although simple and seemingly abstract itself, does not adequately provide course designers or instructors with the means to create and evaluate instruction (Fuller, et al., 2007). For this reason, it is important to recognize the salient qualities of Armoni (2013), Brennan and Resnick (2012), Fuller et al. (2007), and Marzano and Kendall (2007). By focusing on production, interpretation, the affective domain, the self-system, as well as the cognitive domain, computer science educators can create more effective learning opportunities and provide students with better feedback through assessment.

Constructionist Instruction

Additional theoretical constructionist frameworks provide more context for the instruction of abstraction in computer science. After defining the specific cognitive learning objectives in CS education, abstraction, and computational thinking, educational policymakers and researchers can consider common theoretical CS frameworks to evaluate the instruction of abstraction. Papert (1980), a strong proponent of computational thinking, was an advocate of elementary children learning through creating and directing computers. Both math and reading can be simultaneously taught using computers, which is why now reading, writing, arithmetic, and algorithms are being touted as the four R's (Czerkawski & Lyman III, 2015).

Primarily, researchers use constructionism from Papert (1980) to describe the natural tendency students and instructors engage in when learning and teaching coding (Armoni, 2013: Bers et al., 2014; Denner et al., 2012; Fessakis et al., 2013; Harlow & Leak, 2014; Kazakoff & Bers, 2012; Lee, 2010; Wang et al., 2014). Computational thinking is becoming a theoretical framework. Computer science education researchers also utilize (Bers, 2010; Lee et al., 2014; Lye & Koh, 2014). Vakil (2014) added Freirean pedagogy to constructionism in his qualitative research teaching AppInventor in a middle school after-school program for disadvantaged urban students. Vakil's approach illustrates how current and known pedagogy can be combined with CS educational frameworks.

As CS educators become more familiar with existing pedagogy, such as POGIL, and teachers learn to understand constructionism and computational thinking, more overlapping instructional theory will undoubtedly emerge. Kivunja (2014) proposed changing educational pedagogy based on Vygotsky and social constructivism to embrace social connectivism, critical thinking, and digital literacy necessary for 21st-century workforce success. Computer science education that includes explicit abstraction instruction will ultimately facilitate both the acquisition of critical thinking skills and advanced computer science skills.

Instructional Implications for Abstraction

Addressing the pervasive need for CS curricula and resources, CS researchers developed and tested software as a means of legitimate instruction. Tangible software, software that requires the manipulation of objects to create code, has been shown to

increase sequential thinking and even computational thinking, even in kindergartners (Bers, 2010; Kazakoff & Bers, 2012; Wang et al., 2014; Zhong et al., 2016). Gaming software is another type of instructional software, and studies have shown promise in providing software that stimulates computational thinking (Carbonaro et al., 2010; Lee et al., 2014). Additional studies utilizing gaming and robotic software also suggest that such instructional methodology is effective in engaging students, teaching problem-solving, and introducing them to computational thinking (Denner et al., 2012; Grout & Houlden, 2013; Kaleliegoulu & Goulabar, 2014; Pellas & Peroutseas, 2016; Reppening, 2016). Although many of these studies were conducted in after school or summer camp programs, gaming software instruction was correlated with improved motivation, engagement, and computational thinking, especially with female and minority K-12 students, (Daily & Eugene, 2013; Denner et al., 2012; Grout & Holden, 2013; Pryzbylla & Romeike, 2014; Sanchez et al., 2011; Vakil, 2014;). Middle school and high school girls who identified confidence and interest in problem-solving also had a correlated interest in all STEM courses; girls with interest in creativity and design had a correlated interest in computer science and engineering (Cooper & Haeverlo, 2015). Gaming and robotics software appear useful in making computer science fun, attractive to learn, and improving student retention.

Additionally, several studies on CS instruction and computational thinking imply best practices for teaching abstraction. Many standard instructional practices, such as utilizing Universal Design for Learning (UDL) are recommended for teaching computer science and increasing computational thinking (Israel et al., 2015). Tung, Lin, and Lin

(2013) shared a curriculum module for introductory CS students using scaffolding and an algorithm plagiarism detector (providing instant technological feedback), which students found helpful when surveyed. Applying universal design for learning and global immersion therapy, Israel et al. (2015) found that elementary, middle school and college students could be successful in learning computer science. Recognizing that computer science requires visual intelligence, using visualization and encouraging students to draw or writing code using human language, called pseudo code, has been shown to facilitate visualization capabilities (Baloukas, 2009; Shane & Sherman, 2014; Arnoux & Finkel, 2010; Fouh, Akbar, & Shafer, 2012; Ozurt, 2015). Csernoch et al. (2015) indicated that using dance, music, and theater to teach introductory computer science to college students improved test scores, grades, and retention.

Another instructional tactic deemed helpful in generating computational thinking was the immersion into microworlds, such as Unity or Second Life (Jenkins, 2015; Reuker et al., 2013). Interestingly, using kinesthetic instruction and sketching, improved the acquisition of two-dimensional spatial design and computational thinking (Youssef & Berry, 2012). Chang (2014) noted that the visual programming software, Alice, is better suited to alleviate stress and improve confidence with low-performing introductory computer science students than Scratch, insinuating that some instructional software is better for learning object-oriented programming, a programming paradigm designating objects as classes of data in fields with specific procedures (Uysal, 2016). According to Uysal (2016), novice programmers had difficulty learning Java, an object-oriented programming language, due to cognitive load theory. Object-oriented programming may

not be necessary for learning abstraction although more research would help to prove this point. (Gobbo & Benini, 2012). Instructional best practices, such as scaffolding, providing instant feedback, applying universal design for education, engaging multiple intelligences, providing visual and spatial intelligence training, and encouraging creativity and imagination may also be useful to foster abstraction abilities.

Use of the Internet, rubrics for critical thinking, and instructor training are also indicated from critical thinking research at the college and university level. In higher education, computer science students who utilized common aspects of the Internet such as GoogleMaps, apps, and other web services, were more engaged and had better grades (Lim, Hosak, & Vogt, 2012). When college engineering instructors use critical thinking rubrics, they teach more critical thinking (Ralston & Bays, 2015). Also, providing instructional development seminars regarding the use of critical thinking rubrics in college engineering courses was correlated with improved student cognition and affective engagement (Adair & Jaeger, 2016; Haynes et al., 2016). Unfortunately, African college instructors often do not use cooperative learning to assist in the development of critical thinking because they are not trained to do so (Malatji, 2016). Questioning taxonomies focused on evaluative thinking and metacognition are additional teaching practices that can facilitate the instruction of abstraction (Buckley et al., 2015; Festo, 2016; Lihui et al., 2015). Connecting with the Internet, using rubrics for abstraction, and supporting K-12 CS instructors with professional development in using abstraction rubrics might facilitate better instruction of abstraction in computer science.

Moreover, research indicates that collaborative learning environments, interdisciplinary instruction, and ipsative portfolio-based assessment provide effective learning experiences for computational thinking. As stated previously, abstraction is an important element of computational thinking (Wing, 2006). In a qualitative study of third-grade elementary students, Harlow and Leak (2014) determined that memes were propagated during constructionist CS instruction when teachers offered suggestions or guidance. When a student found a solution, he or she communicated the solution with other students allowing them to share in learning, thus propagating a meme. Writing, Science, and English as a Foreign Language are subjects successfully paired with CS instruction facilitating computational thinking (Alsamani & Daif-Allah, 2015; Chang, 2014; Kafai & Burke, 2015; Merricks & Henderson, 2013). Assessment using portfolios, similar to writing, is recommended although surveys and quizzes are being developed to assess execution skills and programming knowledge (Sanford & Naidu, 2016; Zhong et al., 2016). Critical thinking assessments benefit from utilizing standards from multiple disciplines (Liu, Frankel, & Roohr, 2014). Abstraction assessments can similarly be informed from a variety of disciplines, such as critical thinking, Science, and Math. It appears that although there are not many studies on the instruction of abstraction, guidelines like using collaborative constructionist learning environments that allow students to gain CS skills and knowledge in a variety of ways will assist the attainment of computational thinking and subsequently in abstraction.

Literature Justifying the Inclusion of Elementary Teachers

Because contradictory evidence in recent literature exists regarding the age at which students can learn abstraction, teachers may also be confused about how and when to teach abstraction. Similarities between learning abstraction in math and computer science provide a basis of comparison for CS education which lacks research (Colburn, 2000). Teague (2015) found in accordance with Piagetian theory, that novice college programmers did not exhibit the ability to produce abstraction, and it logically follows that K-12 students probably would not be able to produce abstraction. However, recent research in elementary cognitive development in mathematics regarding abstraction suggests that elementary students can learn declarative and procedural knowledge (Braithwaite et al. 2016; Kazak, Wegerif, & Fujita, 2015; Novack et al., 2015; Rittle-Johnson & Schneider, 2014; Szucs et al., 2014). Novack et al. (2016) observed that third-grade students learned a procedure, like a computer algorithm, using an abstract gesture, a kinesthetic movement, for a mathematical concept. The students were given a mathematical grouping 4 + 3 + 6 and shown to use a V movement with their arm for 4+3, so the V pattern + 6 = 6 + V pattern, the commutative property in mathematics. Novack et al. (2015) replicated the work of previous researchers. Computer science instruction research using tangible software maintains kindergarten and elementary children can learn algorithmic concepts and procedures, even computational thinking (Bers, 2010; Bers et al., 2014; Lee, 2010; Kaleliegoulu & Goulabar, 2014; Wang et al., 2014). Also, national CS standards instruct teachers, even in elementary school to teach abstraction (CSTA, 2019). According to the previously mentioned mathematical studies, beginning

levels of abstraction seem to be attainable in elementary grades. A qualitative examination of elementary, middle school and high school teachers' interpretations of the definition and instruction of abstraction will inform inconsistencies in research regarding the instruction of abstraction.

Abstraction skills in elementary students may develop through nonformal, possibly conceptual pathways, versus formal, or procedural pathways earlier than theorists, such as Vygotsky and Piaget have proposed (Braithwaite et al., 2016). Researchers in the Netherlands concluded after evaluating the online math performance of over 50,000 4th through 6th grade students (aged 8 – 12) that students who learned through nonformal pathways, for instance by perceptual grouping of numbers or opportunistic selection of numbers in an equation to solve, made more errors when taught to follow formal procedures or syntactic parsing of numbers based on formal operations. Abstraction in computer science similarly requires formal and nonformal cognitive operation. Interviewing elementary computer science teachers as well as secondary CS teachers will help to inform the research on formal and nonformal cognitive development in abstraction across disciplines.

Additional research confirms that elementary conceptual and procedural knowledge, such as abstraction, can be acquired relying on contextual interpersonal instruction. Rittle-Johnson and Schneider (2014) concurred that conceptual and procedural mathematical knowledge development in elementary school children is bi-directional and iterative, matching the findings of Fuller et al. (2007) regarding students demonstrating multiple pathways to learn computer science. Szucs et al. (2014)

established that nonformal cognitive processes of executive function, phonological processing, verbal awareness, visual-spatial short-term working memory, and spatial ability were more important than formal "number sense" for nine-year-old mathematical cognitive development. Even, dialogic abstract language facilitates the performance of concrete patterning tasks for preschoolers (Kazak, Wegerif, & Fujita, 2015). Language, discourse, pair programming, and the development of memes facilitate computational thinking in elementary students (Fessakis et al., 2013; Harlow & Leak, 2014). The information from CS elementary teachers will augment the growing body of research regarding the nature and extent of instruction guiding conceptual and procedural cognitive development.

K-12 teachers have similar instructional goals for teaching abstraction but different students, ages of students, learning environments, and curricula. Students have a wide variety of computer science experiences in all grades. Computer science courses may be taught as a sub-discipline of Science, Math, or Technology because many states do not have curricular requirements for Computer Science. Additionally, computer science teachers use many different computer software programs to teach coding. Most states k-12 annual assessments do not test computer science competencies. The wide array of variables in computer science education substantiates qualitative investigation. A lack of quantitative data, content development, consistent curriculum, and teacher preparation necessitate investigating the instruction of abstraction. No studies thus far, have interviewed teachers nor sought to triangulate the teacher experience through two teacher interviews and student artifacts. Therefore, this study examines the K- 12

teaching experience of abstraction in computer science for curriculum development, instructional practices, and assessment preferences.

Summary and Conclusions

No completed studies found have specifically focused on the teaching experience of instructing abstraction. Abstraction is primarily situated as a sub-skill of computational thinking even though abstraction is a more complex concept that requires research for both teaching and learning. The majority of research in the past five years has used computational thinking as the theoretical framework (Wing, 2006, 2008). Research from the instruction and learning of critical thinking and abstraction in Math, Science, and STEM courses at the secondary and university level implies that instruction for abstraction would benefit from collaboration, scaffolding, interpersonal learning, question taxonomies, critical thinking rubrics, and real-world applications, such as the Internet. Research from elementary and secondary computational thinking and computer science education suggests that abstraction might be taught successfully using tangible software, constructionist inquiry-based collaborative learning, and gaming software. Recent studies in teaching elementary math indicate that elementary students can learn abstraction, contrary to Piagetian theory. Including elementary teachers in this study adds a layer of complexity, but ultimately facilitate greater pedagogical awareness of effective CS abstraction instruction. The lack of specific research regarding abstraction, the need for computer science teachers, and the lack of research regarding their professional development and pedagogical orientation validate the need for the proposed study. In

Chapter 3, I delineate the specific methodology for this qualitative examination of the instruction of abstraction in K-12 computer science education.

Chapter 3: Research Method

Abstraction is a concept and a process in computer science education that is worthy of investigation especially because computer science programming is being introduced more often in preschool and elementary school. Computer science educators need research to guide pedagogy. As shown in the previous chapter, the instruction of abstraction in K-12 computer science merits study. The purpose of this descriptive qualitative inquiry is to examine teachers' experiences determining curriculum, delivering instruction, and designing assessments regarding the topic of abstraction in computer science. In this chapter, I describe this basic descriptive qualitative study highlighting the interviews K-12 computer science teachers. Although including elementary teachers in this study adds more complexity, the inclusion of elementary teachers as participants enriches and informs curriculum development, instruction, and assessment for K-12 computer science education. Not only do the perceptions of secondary teachers inform the instruction of abstraction, but the perceptions of elementary teachers also help inform future variables for studying grade-level appropriate instruction of abstraction. Future quantitative studies could look at correlations between the use of variables and iteration in programming by grade level if variables and iteration (programming skills) are indicated as strong factors in this qualitative examination of abstraction. In this section, I outline the research design, participant sampling, recruitment, data collection, and data analysis strategies.

Research Design and Rationale

The primary objectives of this qualitative descriptive study are to generate ideas, suggestions, and practical instructional strategies on the subject of abstraction for CS teachers. The field of computer science education, and especially abstraction in CS, lacks research. Qualitative examination of this research subject can provide variables for further quantitative study as well as contextual analysis. The examination of instructional pedagogy for teaching abstraction in computer coding is guided by the research questions:

Research Question 1: What types of instruction do K-12 teachers find most effective for teaching abstraction in computer coding?

Research Question 2: How do teachers determine objectives and competencies for teaching abstraction in computer coding?

Research Question 3: How do teachers assess student abstraction skills in computer coding?

This study was not designed to generate theory regarding learning abstraction or teaching abstraction. Instead, the study was designed to provide educators, researchers, and curriculum developers' practical knowledge about the teacher experience. Practical guidance and suggestions for K-12 CS instructors will ultimately also benefit students.

Qualitative inquiry is an effective research strategy for initial investigation in subjects with many variables (Creswell, 2007). Specific variables for future quantitative research were uncovered in this study. Moreover, this study provided insight into a variety of computer science education topics requiring more research, such as

determining grade-level appropriate objectives, instructional best practices, curriculum and standards, assessments, age-appropriate instruction, and professional development. According to Stake (2010), qualitative research subjectively provides insight into subjects that are complex. Teaching is an inherently complex human to human interaction. Because the field of computer science education is new and little research exists regarding the instruction of abstraction by grade level, qualitative research will provide a more complete understanding of the educational experience, the human experience of teaching.

In this basic qualitative descriptive study, I employed an interpretive/constructivist perspective to glean practical information that will aid current teaching pedagogy. Because the purpose of this inquiry is to examine teachers' experiences determining curriculum, delivering instruction, and designing assessments regarding the topic of abstraction in computer science, the most common form of qualitative research design, basic qualitative, was appropriate (Merriam & Tisdell, 2016). I interviewed 12 teachers (grades K-12) twice and write researcher memos after each interview thus triangulating the data (Yin, 2014). The first interviews yielded data regarding all three research questions. The second interviews also addressed all three research questions and provide more in-depth data.

Due to the specific nature of inquiry related to the instruction of abstraction, the more general aspect of a multiple case study was inappropriate. Yin (2014) recommended multiple case studies for the investigation of how a situation arises when context binds cases. Stake (2006) suggested using a multiple case study format when cases are closely

linked together. With the research questions posed for this study, case study research would uncover more data than recommended when the boundaries of the experience are not clear, and diary studies are useful for examining the intrapersonal experience. Case study qualitative research is thus unsuitable for the nature of this study.

Phenomenology, or the "meaning, structure, and essence" of teachers' experience, might have been an appropriate qualitative approach for this research study; however, the primary locus of abstraction exists in the student mind requiring student interviews, and interviewing teachers would not provide access to the students' internal experience nor yield critical data from teachers (Merriam & Tisdell, 2016; Patton, 2002, p. 104;). Ethnographical qualitative studies investigate individual people or cultures (Stake, 2010). In this study, I examined the teaching experience of CS K-12 educators who were not in the same classrooms, same buildings, nor even the same geographical locations. Therefore, an ethnography was also not an appropriate qualitative approach. I sought practical information for teachers, not deep personal information required by other types of qualitative inquiry. Because in this study I searched for commonalities, differences, and variables for future study, no theory was generated (Charmaz, 2014; Patton, 2002). Hence, the research did not employ a phenomenological nor a grounded theory approach.

Role of the Researcher

I was an outside investigator in this research project. I conducted interviews with teachers and analyzing the interview experience with researcher memos. As the sole researcher, I designed the experiment, recruited the participants, interviewed participants, and analyzed the data. My only professional relationship to teachers was as the former

Computer Science Content Specialist for the state department of education supporting computer science teachers in Colorado. I did not have this relationship with teachers from other states. I did not have any direct relationships with the teachers' students.

Biases

My assumptions arose from my participation in Advanced Placement (AP) computer science instructor training, teaching elementary students computer coding, learning Scratch and AppInventor computer coding, and developing online and hands-on computer coding classes for college and elementary students. I assumed that teachers had some experience with computer coding or worked professionally with computer software or hardware. Teachers who are new to computer science might not have much understanding of the definition of abstraction, and if they do, they only understand abstraction as a procedural or algorithmic skill. Computer science professionals who transferred into teaching will understand abstraction and be able to teach it but may have more trouble developing assessments. Teachers may use more direct instruction than is necessary, according to constructivist theory, to teach abstraction in computer coding. Many teachers rely on free online modules, such as Code.org, to teach their students rather than teaching students themselves. Some teachers are learning computer science along with their students. Teachers may lack comprehensive understanding of brain development in relation to computer science and instructional best practices to foster age-appropriate learning. National standards and instructional material are becoming more available, but teachers do not have research that helps them develop consistent, effective

instruction and accurate assessments of learning (CSTA, 2019). Some students will out-perform their teachers in their understanding and execution of abstraction.

Methodology

In this section, I describe and provide a rationale for the selection of participants, instrumentation, data collection procedures. After describing procedures for recruiting and obtaining consent, I outline the procedures for data collection. Sufficient evidence of procedures and details provide subsequent researchers with enough information to recreate this study. Furthermore, this section includes a comprehensive data analysis plan and examines ethical practices, as well as issues of trustworthiness.

Participant Selection Logic

Twelve K-12 computer science teachers with two or more years of teaching experience or prior private sector computer science experience, four from elementary, middle school, and high school comprised a purposeful sample for this study. The inclusion of multiple grade levels helped to provide information about curriculum and grade level appropriateness of curriculum. I specifically asked teachers when I am recruited participants if they had two or more years of teaching experience or prior private sector computer science experience. Abstraction is an advanced concept in computer programming, and it is possible that new computer science teachers will not have heard of abstraction. Therefore, including new computer science teachers could provide little useful data. Purposeful sampling allowed for specific information from experienced teachers (Merriam & Tisdell, 2016, p. 96). K-12 computer science teachers were recruited, with a preference for four elementary, four middle school, and four high

school teachers. However, convenience sampling superseded typical purposeful sampling due to recruitment efforts which I describe in Chapter 4. The purpose of beginning this qualitative examination of abstraction among all grade levels is that teachers from kindergarten to college is to involve teachers and students with a wide range of coding experience. Teachers in all grade levels often differentiate instruction. It was anticipated that there may be commonalities regarding teaching abstraction that would help teachers recognize student experience and deliver more effective differentiated instruction.

Participant Sampling

I used snowball sampling to find participants who are currently teaching computer science and sought four elementary, four middle school, and four high school teachers. I attempted to find 15 teachers in case some participants opt outed of the study. Although saturation is reached in qualitative studies when participants begin to share the same information repetitively, minimizing the number of participants yields data that maximizes the chance of finding significant themes rather than superficial observations (Cleary, Horsfall, & Hayter, 2014; Merriam & Tisdell, 2016). Qualitative researchers benefit from smaller numbers of participants, and researchers can gain highly relevant data from homogeneous participant groups (Cleary et al., 2014). Over the past several years, I have compiled a list of CS teachers who have expressed interest in participating in my research. I had access to email listservs nationally through the Computer Science Teachers Association in my position with the Colorado Department of Education Computer Science Content Specialist. Fusch and Ness (2015) suggested that qualitative data saturation is reached when coding themes become repetitive. Qualitative data that is

rich, providing many themes, and thick, providing a great deal of material, is most likely to reach qualitative saturation. Although saturation can be reached with as few as 6 qualitative participants or as many as 20, as a novice researcher, I decided to use the middle number of 12 participants anticipating that the data becomes saturated.

I recruited teachers who had a background as professional computer scientists in some capacity before becoming teachers, or teachers who have taught computer science for at least two years. Although teaching experience is not necessarily correlated with student proficiency, students tend to benefit from more experienced teachers (Madsen & Geringer, 2014). Experienced CS teachers and former CS professionals were purposely chosen as participants to increase the likelihood that they are familiar with abstraction, a complex concept. Teachers who volunteer were asked to submit a resume. Because the purpose of this study is to gather practical information about the most effective instructional methods, new CS teachers or teachers who were not previously computer science professionals will not be able to offer the best information. Therefore, I solicited seasoned computer science professionals who are teachers and CS veteran teachers of two years or more.

Instrumentation

The semistructured interview questions that were used for this research were developed by the researcher and evaluated by three experts. Dr. Sylvia Gholston, Dr. Stephanie Hartman, and Jane Waite, Ph.D. Candidate, evaluated the instrument which was revised based on their suggestions. Waite, from the United Kingdom, is currently also working on her dissertation in computer science education examining the instruction

abstraction in elementary schools. Waite has already interviewed 30 students, four teachers, and conducted surveys with several hundred teachers. The instrument for this study can be found in Table 1, Appendix A.

I interviewed 12 computer science and technology K-12 teachers using a semi-structured format once during a one-month period. According to Rubin and Rubin (2005), semi-structured interviews allow for focused data collection and questions that probe for elaboration, clarification, evidence, sequence, and more. I used the questions in each interview to guide the interview, provide consistency and focus, and allow me to pose follow-up questions which can be found in Appendix C. In Appendix B, the interview questions are aligned with this study's research questions. Demographic questions provided a context for the participants and do not align with research questions.

Semistructured open-ended questions facilitated useful data (Fusch & Ness, 2015). By asking open-ended questions regarding determining objectives and outcomes, delivering instruction, evaluating instruction, and developing assessments, I kept the interviews focused on the experience of teaching abstraction. If teachers were unfamiliar with abstraction, I asked them how they provided instruction for the theoretical constructs of computational thinking and critical thinking. If the teachers were unfamiliar with computational thinking and critical thinking, then I inquired about their instructional approach to teaching computer science. Interview questions will be provided data for all three research questions.

I conducted the second round of interview questions one month after the first interviews providing a format for more deeply investigating research questions. In

follow-up interviews, research questions were also semi-structured but designed to address prominent themes from initial interview data. Collecting semi-structured data provided reliability and simultaneously ensured that the data collection process allowed me to explore significant themes. The second interview facilitated data saturation which occurs when participants begin offering similar answers or repeating information (Fusch & Ness, 2015). The two interviews plus analytic memos comprised three sources of research data.

The third aspect of qualitative data collection consisted of researcher memos. Triangulating qualitative data is a way to elucidate multiple aspects of phenomena (Stake, 2010). Additionally, triangulating qualitative data increases reliability and trustworthiness (Merriam & Tisdell, 2016). Researcher memos are used to develop themes related to possible theory development (Saldana, 2013). In this study, I used researcher memos to examine interview topics and questions posed by participants. Analytic memos assisted in the development of variables, which could be used to study the effectiveness of instructional practices quantitatively.

Procedures for Recruitment, Participation, and Data Collection

Teachers with two or more years of experience teaching computer science were recruited from the Computer Science Teacher Association (CSTA), national teacher contacts, and national computer science listservs. Because abstraction is an advanced computer coding concept, interviewing teachers with two or more years of experience was hoped to provide the best data. Teachers were contacted by email and sent the consent form (Appendices A, B, and D).

Participation

Once teachers emailed or called and indicate they were interested in participating in the study, they were asked to submit the adult consent form. All participants were offered one week to examine the consent form and return it. Participants were informed they may opt out of the study at any time by simply contacting the researcher. Ideally, all data collection occurred in one month. After each interview, I wrote researcher memos, inputted the interview transcript and memo into NVivo software, and examined the data for themes. After the second interviews, I repeated the same procedure. Upon university acceptance of the completed dissertation, the researcher emailed all participants the dissertation research.

Data Collection

I collected interview data (notes and audio files) for two months. Three strategies that helped me organize the data were digital, analytic, and interpretive (Yin, 2014). I collected all data digitally and operated an almost paperless data collection. Interviews were conducted via Skype and recorded. Interview notes were typed during interviews or directly after interviews from written notes. The teacher interviews conducted in person, if any, were digitally recorded. All digital interview files were saved in NVivo. All interview documents, researcher memos, and researcher memos were stored on NVivo software which helped in identifying thematic coding.

The audio interviews and typed notes were stored on external hard drives. Teacher interview documents and researcher memos were also digitally stored on the external hard drives. Paper notes and analyses along with the hard drives used will be

kept in a locked cabinet in my home office for five years after the approval of this dissertation. After teachers completed their interviews, I sent them a thank you letter explaining the future expected completion of the study. Upon acceptance of the dissertation, participants will be sent a summary of the final dissertation via email.

Data Analysis Plan

Interviews and analytic memos were managed and qualitatively coded using NVivo software. I used a thematic coding approach to identify, analyze, and report patterns in participant experiences (Gibbs, 2010; Vaismoradi, Turunen, & Bondas, 2013). Thematic coding is flexible and appropriate for novice researchers yet potentially yields rich descriptive qualitative data (Fereday & Muir-Cochrane, 2006). During the thematic analysis of qualitative data, I utilized primarily inductive emic data analysis producing descriptive themes (Vaismuradi, Turunen, & Bondas, 2013). After inductively developing themes, I compared and contrasted the theory and literature from Chapters 1 and 2 with the identified themes facilitating richer understanding and data analysis, as well as exploring directions for additional research (Cho & Lee, 2014). To this end, the qualitative analysis mimicked qualitative content analysis in that themes were inductively developed, and theory was used to deductively identify secondary themes.

Because the researcher must consistently read and reread data in the thematic coding process, I also submitted analytic researcher memos that aided in uncovering significant themes and provided the reflection necessary to develop thematic codes (Vaismoradi et al. , 2013). After descriptive themes from the data were developed

inductively, deductive analysis of the themes completed the thematic coding analysis (Lewins & Silver, 2006).

Logic models and comparisons of theory and literature with inductive themes comprised the second phase of the thematic coding analysis. I established a nonlinear logic model as a strategy for interpreting and categorizing my data (Yin, 2014). Yin (2014) suggested that logic models can be used to describe complex phenomenon, such as instruction, that occur in several dimensions simultaneously. The comparison of theory and literature to inductive themes was not used to generate theory, but rather provided a richer descriptive understanding of teaching phenomena (Fereday & Muir-Cochrane, 2006). Deductive analyses of inductively generated qualitative themes provided contrast aiding in the understanding and development of secondary themes (Cho & Lee, 2014).

NVivo software was used to facilitate the organization, coding, and analysis of data (Lewins & Silver, 2006). I am most familiar with NVivo, and it was the easiest software to learn and navigate quickly. Interviews were conducted on Skype, recorded and saved into NVivo. TRINT transcription services transcribed interviews. NVivo integrates audio files and enabled me to record and evaluate memos. Computer-assisted qualitative data analysis software (CAQDAS) can help researchers organize, code, analyze, and represent qualitative data (Miles, Hubberman, & Saldana, 2014). The type of CAQDAS best suited for a study depends on the nature of the data recorded, the technology requirements and expertise of the researcher, and the goals of presenting research. Some programs, like Excel, provide both qualitative and quantitative functions. I briefly compared twelve popular CAQDAS, such as Atlas.ti, QDA Miner, and several

free software options. I choose three, which might be beneficial in my case study research including NVivo 10, HyperResearch, and Dedoose. CAQDAS programs are tools that can aide only aide but not replace researchers in analyzing data (Yin, 2014). Despite advice from Yin (2014), who recommends not using any software in case study research because the data is generally too diverse, and regarding data storage, data analysis, and data presentation NVivo was the best software for this research study because audio files, transcriptions, and researcher memos were able to be evaluated for common themes, primarily because a variety of different documents, pdf's and audio files (interviews) that can be entered and coded qualitatively.

Issues of Trustworthiness

In order to assess the credibility, transferability, dependability, and confirmability of this research, I analyzed this research design by applying the validity matrix suggested by Maxwell (2013). A validity matrix is a useful tool that helps ensure alignment of research questions with research methodology. Using the validity matrix I aligned the information that I needed to find with data to be collected. Next, I aligned the plan for analyzing data. The information needed would arise from teachers' experiences of the instruction, curriculum, and assessment of abstraction. Data would come from teacher interviews and researcher memos. Data would be analyzed using logic models, thematic coding analysis, thematically coding data to the conceptual framework, and thematically coding data to literature. In Table 2, I illustrate aligning the threats to validity using the validity matrix with strategies and rationales designed to mitigate threats.

Table 2

Validity Matrix Mitigating Threats

Validity Threats	Possible Strategies to Mitigate Threats	Strategy rationale
Concern about anonymity. Focus on abstraction might overwhelm or intimidate teachers. Novice interviewing may produce poor data.	Offer teachers fake names and temporary email addresses. Let teachers know all of their experience is important. Practice interviewing. Develop nonthreatening scripts.	Create safety and rapport. (Miles, et al., 2014)
Must be vigilant writing memos to ensure quality.	Program phone to with memo writing reminders.	Keep myself and the project organized.

Any threat to rapport or safety can compromise qualitative data (Maxwell, 2013). Scripted introductions to interviews reassuring participants of their right to engage in any degree and assure them of confidentiality are crucial for creating safety and rapport (Miles et al., 2014). In the scripted introductions, teachers were informed of the means by which their personal information will be safeguarded and protected digitally and ethically. Assuring the confidentiality of responses should encourage teachers to provide valid responses. Using open-ended nonjudgmental interview questions helped to create safety and rapport with participants, yielding more credible and valid data. The use of researcher memos after each interview provided a reflective tool allowing for the analysis of descriptive themes and possible researcher bias. Communicating to participants that all aspects of their responses and data they share will be ethically safe-guarded, promoted standards for robust qualitative results.

Ethical Procedures

Participants were contacted by phone and by email. First participants were contacted by email. If they did not respond to the email indicating a desire to participate in the study or not, I called them if I have their phone number. When I called them, I informed them about the study using the language in the adult consent form and asked them if they would like to participate. If teachers indicated a desire to participate in the research study, I asked them to email me the required forms. In the email, teachers were informed about the study and the steps they were required to undertake including submitting a signed adult consent form. Appendix D shows the email template teachers received. Initially, prospective teacher participants were informed that they would be asked to interview for two one-hour sessions (in person or via Zoom. I scheduled the interviews after school hours and on weekends with teachers. The two interviews were scheduled two to three weeks apart. The purpose of the second interview was to ask follow-up questions from the first interview. Additionally, because teaching requires some reflection, the second interview captured additional thoughts or observations about abstraction that teachers noticed after the first interview.

Several steps safeguarded the confidentiality of participants' data. First, teacher participants were assured that their experience and information would be respected and remain confidential both in writing via email and verbally in each interview. In order to share the results of the study, quotations from the interviews may be necessary. Participants were informed that if quotations from interviews are cited, their identity will remain confidential. I used alphabetical letters as pseudonyms for teacher participants.

Descriptive data was collected from teacher participants, but their school and location will remain confidential.

Adult consent forms, teacher interview documents and audio files, and researcher memos were saved digitally and backed up on two external hard drives. I used access codes on my computer and will keep the backup drive in a locked safe in my home office to preserve confidentiality and maintain ethical standards. I made sure that interview transcripts and consent forms transferred via email are encrypted and saved on secured hard drives. All emails and duplicate files were deleted.

Transferability

Teachers are used to self-evaluation and often welcome professional development opportunities (Cajkler et al., 2015). Considering that the answers participants provided were confidential thus caused no threat personally or professionally, answers to interview questions are most likely credible. Recruiting teachers from various locations across the United States and who teach a variety of grade levels, aided in the transferability of research conclusions.

Dependability

After exploring theoretical and conceptual frameworks in the previous chapter from philosophers, psychologists, and CS educational experts, it was certainly be part of my bias as a researcher developing themes to be influenced by theoretical and conceptual frameworks. The complexity of the concept and skills required to produce abstraction, as well as the newness of the subject, warrant a thorough examination, including theoretical and conceptual frameworks (Stake, 2010). By comparing themes from participant

interviews with themes from my analytic memos, I observed my researcher biases. As themes began to emerge, I compared outlier cases with thematic trends exposing my biases. Thus, the thematic data analysis plan included an inductive emic exploration of themes and a careful examination of etic researcher bias.

Confirmability

Confirmability in this study was primarily determined through the comparison of researcher memos and both interview transcripts. Qualitative studies are by design difficult to completely objectify; one way that researchers demonstrate their efforts to be objective is to repeatedly review data ("Qualitative Validity", n.d.). By evaluating data after each interview is entered and documenting the process with researcher memos, the qualitative methodology for this study demonstrated reflexivity with a memo audit trail (Olivia, n.d.). The iterative focus on participant data using memos helped guard against researcher bias.

Summary

Based on the lack of research on the instruction of abstraction in computer science, the complexity of the teaching experience, and the conceptual and procedural nature of abstraction, a qualitative case study design was indicated for this research. Triangulating teacher interviews and even researcher memos creates a reliable credible qualitative study. I employed an interpretive/constructivist perspective to inform this basic qualitative study designed to illuminate the understanding of effective K-12 curriculum, instruction, and assessment of abstraction. I used an emic qualitative coding

strategy to assist in discovering practical teaching pedagogy. Moreover, the study informed teaching practices for critical thinking and mathematics. In the next chapter, I share the results of the study including the demographics of participants, significant themes related to the interviews and analytic memos.

Chapter 4: Results

In the data collection phase of this study, the advanced and conceptually challenging nature of abstraction in computer science became readily apparent. The purpose of this descriptive qualitative inquiry was to examine K-12 teachers' experiences determining, curriculum, delivering instruction, and designing assessments regarding the topic of abstraction in computer science. The following specific research questions were a subset of the main question: How do teachers decide what effective instruction for teaching abstraction in computer coding is?

Research Question 1: What types of instruction do K-12 teachers find most effective for teaching abstraction in computer coding?

Research Question 2: How do teachers determine objectives and competencies for teaching abstraction in computer coding?

Research Question 3: How do teachers assess student abstraction skills in computer coding?

The results detailed in this chapter from data including interview transcripts, student artifacts, and researcher memos describes how teachers use a variety of instructional approaches to instruct and assess the multifaceted topic of abstraction in computer science.

Setting

I chose a purposive convenience sampling of teacher participants that also involved some snowball (word of mouth) sampling. The teacher participants in this study were complete strangers or teachers with whom I had limited professional contact. Many

of the teachers may have known of me or heard of me as the state department of education Computer Science Content Specialist. Part-way through data collection, the position with the state department position ended. I knew one teacher from our work together on several projects and from our joint membership in the Computer Science Teachers Association. Our relationship was only professional. No significant events in the lives of participants or myself, the researcher, were noted as interfering with interviews, data collection, or analysis. Interviews were conducted and recorded virtually using Zoom for ease of convenience and recording audio. Teachers were in their homes, away from school or in their classrooms outside of school hours. Interview rooms were quiet, and teachers generally were engaged and interested in answering the interview questions.

Demographics

The teachers in this study were primarily secondary AP Computer Science teachers. For confidentiality, the teachers were referred to in all communication and documentation by an alphabetical letter. The average number of years of experience teaching computer science was 15.5 years. As seen in Table 3, the teachers' primary teaching disciplines were either math or science. Only one teacher had a bachelor's degree in Computer Information Systems with an emphasis on programming, teacher C. Teacher C worked as a programmer and hardware technician before transferring into elementary and then secondary computer science teaching. Teacher A also teaches AP Physics. Teacher B has over 27 years of teaching experience in Business, AP Calculus and all levels of mathematics, as well as computer science.

Table 3
Teacher Participant Demographics

	Grades Taught	Courses Taught	Experience	Years Teaching	Teaching Discipline
A	9-12	AP CSA	4 years	20 years	Physics
B	9-12	AP CSA, AP CSP, Intro to Web Design	7 years	29 years	Math/Business
C	9-12	AP CSA, AP CSP, Intro to Web Design	11 years	13 years	Computer Science
D	K-6	Technology/ Digital Literacy Coach (Code.org trainer)	5 years	6 years	Math/Technology
E	6-8	STEM	1 year	14 years	Instructional Technology
F	9-12	Intro to Programming, Web Design, Nand2Tetris	3 years	5 years	Math
G	8-12	STEM, APCSP	5 years	16 years	Math/Physics
H	11-12	Intro to Programming, AP CSA	5 years	5 years	Math
I	6-8	Science, after-school STEM	5 years	16 years	Science
J	11-12	Cybersecurity, CTE Computer Science	13 years	13 years	CTE Information Technology
K	9-12	AP CSA	3 years	5 years	Math
L	6-8	Cybersecurity, Game Design, Intermediate CS (Python), Advanced CS (Java).	4 years	25 years	English

Teacher D is an elementary district and state trainer for elementary Code.org workshops. Three participants taught middle school courses. Five of the 12 participants were female, 7 were male. The teacher participants have a vast combined pool of experience teaching and teaching computer science.

Data Collection

I interviewed each of the 12 teacher participants twice. Each interview lasted between 30 to 60 minutes. Most interviews were conducted one to four weeks apart although both interviews for three teachers occurred during the same week due to scheduling constraints. After each interview, I recorded research memos. Interviews were conducted virtually on Zoom for ease in scheduling and recording. I introduced myself via video and then turned the video off after introductions, so interview questions were answered only recording the audio communication. Teacher participants were at home or at work outside of school teaching hours in a quiet room. I was also in my home office in a quiet environment.

I collected 5 deidentified student artifacts that teachers chose showing examples of abstraction in student coding for teachers A, C, and D. It took longer than I anticipated (4 months) to get district level letters of cooperation from four school districts out of over thirty that I requested. One school district turned the request down because I was not offering a teacher stipend. Other school districts had prohibitive deadlines for submitting research requests. Several school districts in major metropolitan areas in three states failed to respond to emailed research requests. Even trying to recruit 30 to 50 teachers in each of the four districts that did approve my research, yielded a very small number of

teacher volunteers. Teachers who did not want to participate responded that they were too busy, had multiple jobs and family commitments. Other teacher participants shared that the topic of abstraction was daunting and at the beginning of the school year they weren't sure if their students knew enough to produce abstraction in computer coding. After consulting with my committee and other university officials, I submitted a request to change my data collection requirements to two teacher interviews and researcher memos, no student artifacts. This change was approved and allowed me to contact any teacher which quickly resulted in obtaining the targeted number of 12 teacher participants. I was unable to obtain the desired number of 4 elementary, 4 middle school, and 4 high school teachers. In the end, the participants consisted of one elementary, 3 middle school, and 8 high school teachers.

The basis of questions from the first interview can be seen in Appendix A, and the second interview questions in Appendix C. In both interviews, I applied follow-up questions to the basic questions in order to ascertain as much detail from teachers' experiences as possible. The first interview questions were developed using the research questions. The second interview questions were developed thematically from the first interview transcriptions and memos. Second interview questions also included participant questions and concerns related to teaching abstraction in K-12 computer science.

I configured interviews to record on a cloud server using Zoom. After downloading the recordings to my computer and deleting them on Zoom, I uploaded the recordings to Trint transcription services. After transcribing the interviews using Trint, I downloaded them to my computer again and deleted the interviews from Trint. Then I

uploaded the transcribed interviews into NVivo software as a receptacle and organizational virtual location for thematic coding. Interview memos were also uploaded into NVivo, as were student computer coding artifacts. Teachers emailed me the artifacts. Once uploaded into NVivo, the emails with student artifacts were deleted on Zoom, Trint, and the download file on my computer. I made every attempt to ensure the privacy and confidentiality of collected data. I made two changes to my data collection plan in Chapter 3 to make the data collection easier and minimal. Zoom was easier to use than Skype because no log in information is required. Zoom also has the ability to record and save large files in the cloud minimizing memory demands on my computer. I only communicated by email and did not take phone numbers from participants, except with one teacher with whom I texted after she contacted me via phone. The other procedures including storing research data on an external hard drive were followed exactly as described in Chapter 3.

Data Analysis

Process of Inductive Analysis

I interacted and evaluated each of the 24 interviews between 4 to 5 times. During the interviews, I took notes on copies of the research questions used for the base questions in the two interview rounds. The logic model (Figure 9) shows the progression of data collection and analysis. I edited transcriptions and listened a second time to each interview making additional notes. Then I entered analytic memos for each interview, a process that yielded additional insights and themes.

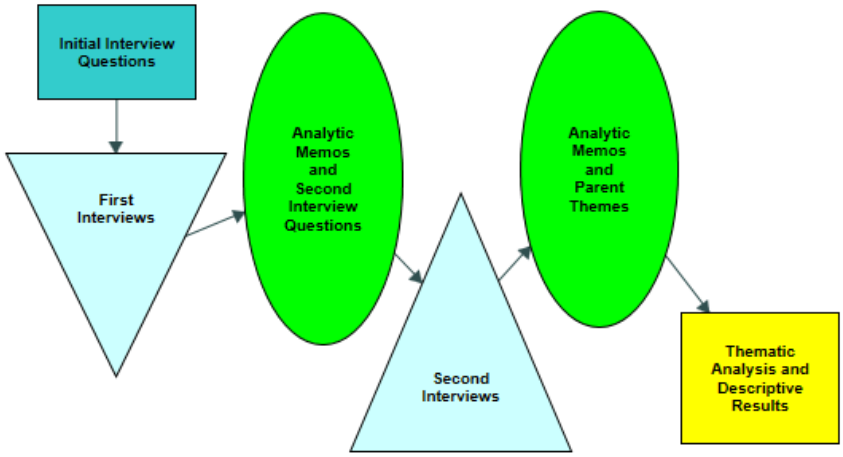

Figure 9. Logic model of research activities

Then, analyzing data for word frequencies and sentence level themes, I employed NVivo software coding parent and child theme. I coded all data iteratively including both interview transcripts and analytic memos often relying on visual data representations like the excerpt of a diagram seen in Figure 10. Parent themes and child themes are commonly used terms to describe categories and subcategories of qualitative themes (Merriam & Tisdell, 2016).

Initial parent themes arose from comparing research question categories (i.e. curriculum, instruction, assessment, and the definition of abstraction) with word frequencies in each interview. Specific parent themes corresponded strongly to the research questions and base interview questions and included: abstraction knowledge, instruction, assessment, curriculum, teacher experience, student experience. Each teacher participant is referred to by a randomly assigned alphabetical letter to respect

confidentiality. Responses regarding teachers' familiarity of abstraction ranged from Teacher J stating and indicating she was not familiar at all, "On a scale of 1-10, I'm a 1."

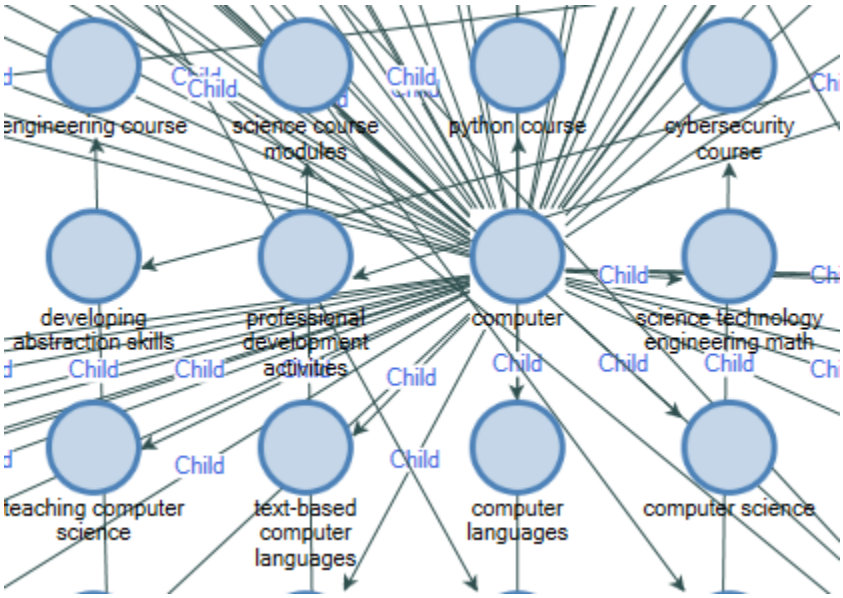

Figure 10. Exploring Parent and Child Themes using NVivo Software.

Whereas Teacher I explained, "I am very familiar with abstraction and teach it at the beginning of my intro class and all the way through my AP CSA course."

Teachers told many stories providing examples of their instruction of abstraction such as, from teacher C using games like rock, paper, scissors that students would work to program or teachers H and L using unplugged activities (instructional activities not using computers) to help students learn the concepts related to abstraction. Teacher J utilized student self-assessments but did not grade abstraction. However, Teacher L included "elegant coding", her term for abstraction, in rubrics she gave her students. Several teachers, namely teachers I, K, and L, indicated that they found it difficult to get students to independently demonstrate abstraction in computer coding projects.

Table 4: *Parent and Child Themes*

Parent Themes	Child Themes
Abstraction Knowledge	Ubiquitous, Transfer from other content areas, Metaphor, Learning skill first then concept, End-user
Instruction	Vocabulary, Unplugged activities, Thinking skills, Repetition-spiraling curriculum, Programming languages, Objectives, Logical problems, Learning by doing, Labs, Games, Frequency of abstraction activities, Design process, Debugging, Cooperative Learning, Contextualized learning, Challenges, Block-based coding
Curriculum	STEM, Standards, Simulator, Science, Robotics, Resources, Programming languages, Math, Game Design, Cybersecurity, Artificial Intelligence
Assessment	Summative, Formative, Self-Reflection
Teacher Experience	Years teaching, Teacher support, Courses, Abstraction professional development, CS Teacher pathway, Grade-level instruction, Self-efficacy, Support from district, Teacher support
Student Experience	Examples, Background knowledge, Ability – student dependent

Teacher H explained that his students had beginning exposure to basic programming and getting kids to demonstrate abstraction was sometimes,"….like trying to go fast with your training wheels on." The challenging nature of teaching students abstraction are more fully reported in the results section of this chapter. A complete accounting of parent and child themes are provided in Table 4.

Child themes arose from consistently thematically coding each interview and every researcher memo, looking for word frequencies, thematic frequencies, and thematic connections then revising parent and child themes accordingly. Connections between

parent and child themes can be seen in Figure 11. Child themes for abstraction knowledge include: end-user, metaphor, transfer from other content areas, and ubiquitous. Child themes for instruction include: pedagogy, block-based code, challenges, cooperative learning, contextualized learning (grandchild themes – demo, expo, competition, project-based learning, and real world service learning), debugging, design process, dialogue (grandchild themes – group discussion, Socratic dialogue, student led-inquiry), direct instruction (grandchild themes-online tutorials), frequency of abstraction instruction, games, labs (grandchild theme - maker spaces), learning by doing (grandchild themes – building background knowledge, student-led inquiry, too much code), logical problems, objectives, programming languages, repetition-spiral, thinking skills, unplugged activities (grandchild theme- engaging multiple senses). Child themes for curriculum include: artificial intelligence, cybersecurity, game design, resources, Math, Science, robotics, simulator, STEM, and unplugged activities. Child themes for assessment include formative, summative, and self-reflection.

Child themes for abstraction knowledge include end-user, metaphor, transfer from other content areas, and ubiquitous. Child themes for the student experience (as interpreted by teacher participants) include ability – student dependent, background knowledge, and examples of abstraction ability. Child themes for the teacher experience include abstraction professional development, courses taught, CS teacher pathway, grade level instruction, self-efficacy, teacher support (grandchild theme – support from district), and years teaching.

Discrepant Cases

As a descriptive study, all participants help inform the research questions in this study. However, it should be noted that only one elementary teacher and three middle school teachers were interviewed. Because there was only one elementary case, I don't have sufficient data on which to comment regarding abstraction in elementary. Additionally, due to the approved changes in methodology and the lack of student de-identified computer coding artifacts, I did not analyze the samples of student coding that were submitted.

Evidence of Trustworthiness

Credibility, Transferability, and Generalizability

All teachers in the study are currently employed and teaching CS, STEM, robotics, or some aspect of CS requiring computer programming. The participating teachers were curious and interested in the topic of abstraction. They genuinely wanted to learn more, and even long-time CS teachers were unsure of their performance and desired feedback. Due to the sincere nature of responses, the data is credible. However, the majority of teachers came from a western state, with the exceptions of one teacher from the Midwest and one teacher from the East Coast. The diverse grade levels teachers address provide a degree of transferability, as much as can be afforded in a qualitative study. Moreover, the educators from three states began to repeat answers indicating saturation. All 12 teachers asked for a definition of abstraction at the beginning of the first interview. I responded that because the study was designed to assess their experience, I would like to first find out their definition.

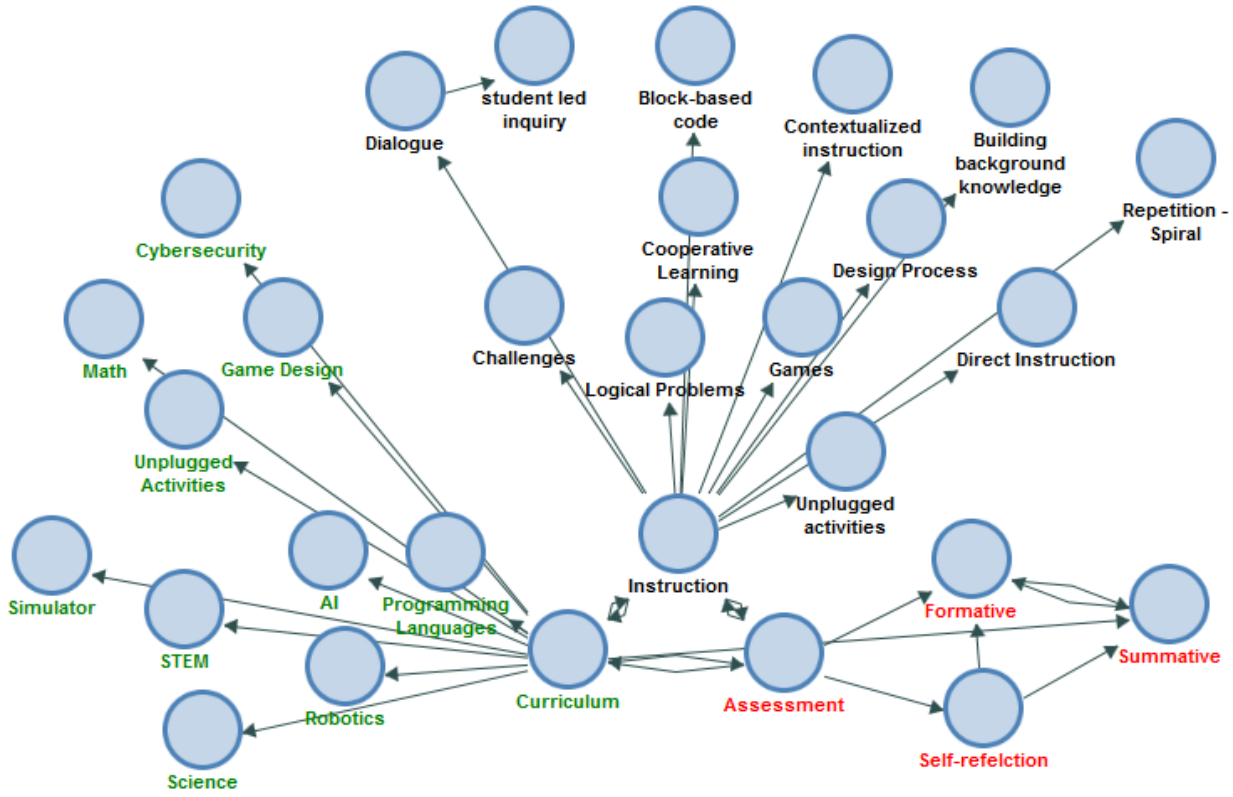

Figure 11. Project map depicting relationships between parent and child themes.

Sometimes, I shared the following basic definition of abstraction. "Some people define abstraction as managing complexity or hiding detail making computer code representative and more efficient." After sharing the basic definition of abstraction all teachers replied that they did teach students to make their code "elegant", "streamlined", and "efficient".

The high school teachers who taught AP CS and advanced CS courses shared a similar concern in that they had some students who easily demonstrated abstraction and understood it but they struggled to find ways to help the "bottom of the pack" understand

abstraction. Teachers also mentioned it was difficult to get students ready for the AP exam on time when students at the "bottom of the pack" seemed to need more time in order to learn abstraction. Such similarities and other themes which are explored in the results section of this chapter indicate transferability of data.

It is hard to gain generalizability with a qualitative sample, but some aspects of the results might apply to teachers in many states. One teacher from a state that has had an earlier push for CS than the primary western state from which most of this study's participants came from shared that he took the CS class that *he now teaches in high school*. He went to the same high school where he now teaches, and his mentor, as a CS teacher, is his old high school CS teacher. This teacher participant understood abstraction easily, discussed abstraction easily, and had a strong sense of how and when his students demonstrated abstraction in their computer code. He was also attending a Master's program in CS. In his fifth year of teaching the same curriculum and courses, he mentioned that he could incorporate more depth and abstraction into his courses now because he was more familiar with the progression and material. The amount of experience as both a CS student in high school and higher education logically seems like it would correlate with teaching knowledge of abstraction and self-efficacy.

In contrast, two middle school teachers who had much less formal training in CS and experience teaching CS courses were the least able to describe abstraction activities and student abstraction examples of any teachers. Three teachers were in the middle of teaching advanced year-long CS courses that they had not previously taught. These teachers all honestly shared that they were in the planning stages of learning material,

setting objectives, designing future lesson plans, designing assessments, and were not able to fully describe concretely examples of lesson plans and assessments that incorporated abstraction because they hadn't finished teaching the entire course. Again, generalizability with common variables such as experience with content and experience teaching the curriculum logically correlate with knowledge of abstraction and teacher self-efficacy. Additional common themes are elucidated in the results section of this chapter.

Dependability

As a teacher not currently practicing in K-12, I am more of an etic participant, although as a teacher immersed in CS education, I can easily relate to the experience of the teacher participants as an emic participant, a teacher. Also, as someone who is learning to program in multiple computer languages, I am approaching the subject from more of an emic educator lens with less content knowledge allowing me to be more objective in relation to the concept of abstraction and less objective about the art of teaching. As the study progressed I found that during the interviews I was making inferences about the degree to which teachers understood abstraction. Making such a judgement was clearly an etic bias preventing me from objectively describing the experience of the teacher participants. When I realized from studying my analytic research memos that I was making judgements about the degree to which teachers "understood" abstraction, I iteratively examined interview transcripts and researcher memos to see what new themes arose. Thus, throughout the study I was carefully monitoring any biases and iteratively examining the data.

Confirmability

Multiple researcher memos and interviews per participant provided reflexivity in the data analysis. As previously mentioned, I interacted with each interview data multiple times over the course of 4 months. I participated in the interviews, edited the transcriptions of each interview making notes as I listened, wrote analytic memos, and then thematically coded each interview multiple times. This exhaustive approach to analyzing data demonstrates my efforts to ensure confirmability of the results.

Results

Teacher participants' understanding of abstraction, designation of course objectives, instructional activities, and assessments varied with experience both with CS content and teaching CS courses. Experience was the overarching theme related with the other salient themes in this study. In the following section I describe teachers' knowledge of abstraction, curriculum, and demographic aspects related to research question two. I begin with results related to research question 2 because teaching begins with identifying terms, concepts, and objectives to instruct and then assess. Next, I share findings regarding teachers experience instructing abstraction and teachers' observations of student abstraction ability, related to research question one. Then I describe teachers' experience assessing abstraction related to research question three. Finally, I provide additional insights from teachers regarding professional development and suggestions from participants.

Research Question 1: What types of instruction do K-12 teachers find most effective for teaching abstraction in computer coding?

Teachers found many types of instruction effective for teaching abstraction in K-12. Teachers focused on sharing their knowledge of abstraction as a ubiquitous concept through metaphors, direct instruction, focusing on the end-user, and making transfer references from abstraction in other content areas such as Math or English. Teachers mentioned the following parent and child themes as effective modes of instruction: teaching vocabulary through context; unplugged activities; logical problems; learning by doing; design process; contextualized project-based learning; repetition of abstraction and spiraling curriculum; labs; debugging; cooperative learning; giving students challenges; and using a variety of programming languages including block-based programming. Teachers found that student ability made it sometimes unnecessary to teach abstraction to "savant" students, but students "at the bottom of the pack" who struggled to learn abstraction were difficult to teach. For some teachers, all of the strategies that work for many students don't work for some students who struggle with abstraction. This complex interplay between student, subject, and teacher illustrates the difficulty in conducting educational research. Are students who struggle with abstraction the discrepant cases under research question one or are the teacher's instructional strategies? The following stories and quotes from teachers interviewed will help illuminate results relating to research question one.

Abstraction Knowledge

Because teachers' definitions and understanding of abstraction influenced the way teachers chose curriculum, taught abstraction, and assessed abstraction, teacher knowledge of abstraction is relevant to all three research questions. Teachers' knowledge of abstraction ranged from concrete understanding based on traditional computing to a focus on the end-user's experience to a vague understanding of the concept. One teacher who had a B.S. in Computer Information Systems, explained, "…when I was in college I had a friend who we would take each other's code and we would look at it and we could see who could actually make the shortest most functional program to accomplish the task." And another teacher explained, "So actually, in programming for kids for anybody you know to make any efficient program there needs to be abstraction." Another teacher described abstraction as, "Then when you were programming you had you would do data hiding or data representations…" The idea of hiding data was repeated from another teacher,

> And I think you know the thing that I've tried to stress the most to my students and I believe I touched on this last week is just abstraction being something that hides the nonimportant details the extraneous kind of fluff but packages it all into some sort of black box.

A teacher who had some experience programming science simulations in college made the distinction between procedural and data abstractions. "So, when we go over like the level of abstraction we talk about you know in the program language that I'm working on I work primarily on procedural and data abstractions." Other teachers had a less concrete

ready definition of abstraction but more a sense of the concept. One teacher who had taught AP CSA and Java for several years explained,

> It's in my mind the way I think of abstraction is it's a sense that no variable actually can mean something else. You know you might pass in a parameter that's some variable that eventually will have some actual meaning. But when the kids are writing their code it's just this word. This letter this idea that's out there that's not actually implemented yet.

and she further elaborated,

> I know it is one of the most important principles as far as like object-oriented programming goes and I understand how it is related to encapsulation, inheritance, polymorphism and you know what I'm saying but yes abstraction, I'm like ok, not exactly, is that what you mean?

Another teacher explained and possibly was conflating abstraction as a programming skill and the related ability to think about nonconcrete concepts,

> I feel like it's a pretty natural part of what we do. You know this whole sense that they write something that will eventually be like get some sort of actual meaning. That's sort of abstract that sort of thing. I feel like it is just central to everything we do.

Another teacher resourcefully looked up the definition of abstraction on Google when I let her know I was first interested in her ideas of the topic before I shared a common definition and explained, "I mean because as I'm looking at right now I'm looking at you know the definition that it's used to reduce complexity and allow efficient design and implementation of complex systems." One teacher honestly was not sure of the definition and explained, "I think I don't know actually because I'm not really sure from a pure CS perspective what that actually means. So, I guess not really. You know I have a sense of what it is."

After initial questions and after I shared the basic definition of abstraction mentioned earlier, we discovered some teachers used words like elegant or architecture to describe abstraction. A former English teacher explained that although she had never heard the term abstraction or studied it, she focused on teaching her students to write elegant simple code, citing the rationale of Occam's Razor, the simplest answer being the best answer. One teacher who spent 20 years in IT before becoming a vocational CS instructor explained,

> I would be more inclined to use the word architect but the ideas are the same. I'm thinking about how these pieces parts go together to create what is that the user wants, so I get a lot of opportunities to do that in a PBL [sic: project-based learning] framework.

And further,

> So, it's this idea of trying to get kids to reverse engineer and to think about the pieces parts that go into a holistic system. But the outcome we want is that the kid understands that there are multiple parts that go into making a computer complete including software.

Linking the idea of the end-user's experience to abstraction connected the design process and the definition abstraction. Incorporating the lens of the end-user on abstraction in computer coding also introduced the idea of defining the concept from multiple perspectives. The previous teacher with 20 years of IT experience explained,

> So, I'll describe it in the way that I would to a kid. I call the end user Ma or Pa Kettle. And so, I'm always saying Ma Kettle comes to me. And she's in the marketing department or sales or engineering or whatever and they need a certain app. And so, they're able to describe the end goal but they don't have any idea about the technology or technologies. In the back office they are going to make that happen. So, in my mind abstraction is taking those requests you have to go through a process of discovering all of the requirements that are needed. Requirements gathering once I have the requirements. Abstraction means that I'm

> figuring out how things are going to be put together to create a useful app. If that makes sense.

Another teacher explained in the second interview new discoveries about the definition of abstraction, “I looked up the definition on the Internet and now it seems like it is more about the product and what the user experiences.”

Teachers also described their knowledge of abstraction in relation to skills, concepts, and thinking abilities. A STEM middle school teacher indicated the necessity of problem-solving thinking skills, “Like how can we leverage technology to be able to problem solve easier and faster more efficiently and that kind of thing.”

A high school teacher who had also taught elementary school shared the importance of teaching pattern recognition, “So, you start to teach people abstraction by helping them with pattern recognition.” When asked if abstraction was a skill or a concept, 6 teachers said it was both a skill and a concept. One teacher explained, “Both, more of a concept, kids could do the skill but understanding the concept is harder.” Another teacher explained how students learned some aspects of coding that allowed them to do the abstraction skill but then tried to use the same approach without success in other problems because they didn’t understand the concept of abstraction. “So, there's not understanding the situational need for that particular solution. And there again it's like going to the tool box and the only tool you have is a hammer so everything looks like a nail.” Four teachers immediately said abstraction was a concept that transforms into a skill. Interestingly, the two oldest teachers, both in their 60’s, who were also very focused on teaching multiple computer programs and courses said abstraction was a skill that

according to one teacher, “It begins as a concept but doesn’t do any good until it is applied. It is almost an art.”

Figure 12: Word frequencies with knowledge of abstraction

Treating abstraction as a skill or a concept or both influenced teachers’ instructional approaches. Figure 12 shows the frequency of words related to discussions about teachers’ knowledge of abstraction. As a bridge to the results section on instruction, several teachers shared metaphors they used with their students to explain the concept of abstraction. Previously, the metaphor including “Ma and Pa Kettle” as the end-users alluded to the product and process nature of abstraction. The product was whatever app the user needed. The process included requirements gathering from the user, decomposition of the problem, and then “abstracting out” or inductively proposing a solution from hardware, systems, and software, resulting in the design process of creating

a solution. Although, the entire process described in the previous sentence would be defined by some as CS, the teacher in this study defined the same process as abstraction. This same teacher used the metaphor of football to explain abstraction.

> I think a lot of kids understand football and football is a very very complex game with lots of different mathematics going on it plays and plans and how we get to the end zone. And so, kids really any kid that is into football doesn't have a hard time holding down all of the data that they need to figure out how to run that ball and get it into the end zone. If we ask a kid how an app got onto their phone, they have no clue apart from they went to the app store and searched for it and got it. So, these are two extremes in you know abstraction.

Another teacher described how he used the metaphor of liberal arts and technical higher education.

> I sort of I use that [sic: metacognition] as a way to sort of have the students realize that they already think about abstraction a lot in everyday life and that makes sense. One of the things I do is tie in to the higher education system and how you know some schools and colleges do a liberal arts model and some colleges do the sort of specialization model or the more technical model. We talk about it and I sort of take those models to an extreme and say how you know neither of you if you take the breadth first model to an extreme that it's not useful that you take the depth first models to an extreme that it's not useful either. And so, abstraction is sort of a way of meeting in the middle in some ways.

Another teacher shared how he uses an activity and a metaphor to teach abstraction. He combines the classic games of Pictionary and telephone by having students at one end of a circle write down the instructions for drawing a polygon. The next student draws the image they think the instructions describe and the task continues around the circle alternating with a picture and written directions. The teacher explained connecting the activity to abstraction in CS,

> It's kind of like telephone but with alternating instructions and diagram. We really talked about how when you were giving instruction, what was the instruction you needed and what was the instruction that that was lacking that caused the sort of

> loss of concept. Basically, how much is enough and how much isn't enough. We talked about Google Quickdraw and how now from an AI standpoint how it could quickly like if you said. sailboat how much do you need to draw for somebody to understand the idea of sailboat. Well, not much it turns out you know. So. We are so are we talking about abstraction a lot that way.

Another teacher simply stated that abstraction was, "It's going from messy to pretty."

Teachers all shared a common belief that abstraction was important for students to learn. According to one teacher, "I think it is really important. I don't know how you could really do CS without having a good grasp of it."

Another teacher explained, "It's critical to everything pretty much everything that you do in programming for sure. And in understanding other areas in CS, nonprogramming areas of CS, too."

Instructional strategies for teaching abstraction

As noted by the math teacher in the previous section, CS is a new content area for students. Students undoubtedly have used and seen computers and computational devices but learning how the computational devices work and then learning to solve problems with computational solutions, the essence of computational thinking, is a new avenue of study for students in almost any grade. Teachers noted that they needed to carefully build learner background knowledge of abstraction in CS through direct instruction, scaffolding, contextualized instruction, and activating background knowledge. Specifically, teachers mentioned utilizing collaborative learning, the design process, block-based coding, object-oriented programming, various forms of dialogue, and learning challenges to teach abstraction.

When asked about utilizing direct or contextualized instruction, a long-time business and math teacher who has taught AP CSA and Java replied,

> I'm going to model it and now we're going to do it together. That direct approach to instruction, honestly I've only ever really done it….with lots of practice lots of example problems and talking about what different things would mean.

Another teacher described her approach to direct instruction, "So, it's you know five to 10 minutes of direct instruction for an initial lesson to then apply that."

Demonstrating and modeling were mentioned as being an important aspect of teaching abstraction. One teacher explained, "And a lot of them picked up on it right away and some of them sort of understood it after I was showing them a bit."

Prescribed curriculum has scaffolded instruction built in. One teacher shared, "There's great, you know, curriculum step by step stuff that you can do." Another teacher noted that the online course he was teaching required students to complete basic foundational hardware simulation activities before moving on, "But I keep coming back to this but I think that one of the cool things about the Nand2Tetris course is that they have to get their chip to work."

Teachers described that students might acquire skills related to abstraction at home or in school but not understand the concepts and be able to apply the skills in a variety of applications. A middle school STEM teacher explained, "They need that full practice time and I think they need a safe practice time to be able to figure it out and do that trial." Prescribed online curriculum was described as helpful, but not necessarily active learning. An AP CSA teacher explained, "I haven't used Code.org enough to think

this is a fair assessment but passive learning I feel is more out of online delivery systems that have students even if they're typing answers and trying things clicking around." He concluded by saying that online tutorials were good for drills and training. A middle school teacher who set up tutorials for her students using Agent Sheets to help her differentiate student learning because some students were "sitting there bored" when they easily finished work, shared that students didn't really understand what they were doing until they talked about their work.

As a segue to explaining significant themes regarding the contextual instruction of abstraction, a math teacher noted that teaching abstraction was very similar to how he taught math, "You sort of teach the process and try to ground that process in some conceptual understanding." The same teacher mentioned that it was important to let students fail and experience writing lengthy code to develop value for finding easier ways to achieve coding solutions. Additionally, this teacher shared how he showed the PBS Crash Course videos on CS as a contextual instructional activity,

> These videos talk about some idea in computer science and then they sort of cut away to this goofy graphic of an elevator and they do like this ten five or ten second montage of the elevator going up a new level of abstraction.

It is notable that the Code.org curriculum also utilizes videos as unplugged demonstrations. Experiencing programming abstraction was a way that teachers could then later explain the concept to students. An AP CSP teacher who used the AppInventor curriculum shared how he taught students to program a pseudo random number generator and then program coin flipping. "All we'll do is flip coins and flip a whole bunch of coins

and then we'll see is the app does the app have a good pseudo random number generator."

Incorporating traditional games into programming was a way that several teachers shared how they incorporated elements of direct instruction, building student background knowledge, and contextualized learning abstraction. One teacher described how she regularly had students play common games like rock-paper-scissors to learn 2D arrays or Yahtzee and then had student program the games. She had students program the dice in Java and then program the rules for the Yahtzee game demonstrating the object and procedures required to produce abstraction in object-oriented programming. Other teachers mentioned having students play Connect Four and then programming that game or hangman or evil hangman. To create evil hangman, the teacher explained he had the students program a random word generator making the hangman game more complicated.

Additional contextualized topics teachers shared included creating mazes for robots to navigate and creating online banking programs. Contextualized learning was a way that teachers could spiral curriculum and expand the concept of abstraction in new situations allowing students to make new connections to the concept. As one teacher explained, "Or if it happens inadvertently in context like as they're solving a problem kind of in a bigger context."

Teachers used group discussions and Socratic dialogue to help students understand abstraction. One teacher described her instruction of Scratch, "And I explained to them that was so much more work than using a broadcasting tool."

Another teacher shared how he explained to his students that using the modulator function in AppInventor produced abstraction. An instructor shared how he used Java libraries to explain abstraction. Student-led inquiry was another pathway to teaching abstraction. The AP CSP teacher described a student who recognized an easier way to program an app that the block-based AppInventor programming language did not accommodate. Another teacher explained,

> One just kind of fun discussion we had towards the end of the semester was if you have ten problems left on a multiple-choice test and you're not sure you know you can't eliminate any of these answers, is it better to choose a letter like C and mark it all the way down? And so that was something that you know we talked about the mean, the probability that we thought maybe the variance would shift and it was a little 5 to 10-minute discussion that came up.

The teacher shared how students involved in this discussion about ten remaining questions on a multiple-choice test went home unbidden and programmed in Java all the probabilities in this multiple-choice scenario as a way of studying the entire course material for the final.

The two middle school STEM teachers stressed that teaching abstraction was embedded in teaching building, creating, and the design process. One teacher explained she used the Lego EV3 robot kits which allow students to create a variety of robots,

> Now this whole programming idea of the EV3s, creating robots, that will help answer this question that involves science, technology, engineering, and math. So, they're kind of putting all their knowledge together which essentially was my goal in the end that it's not separate that all of this comes together and they can see how it comes together.

The other STEM teacher explained,

> …if they start just with coding on a screen then I'm basically a glorified programming teacher versus a teacher where I'm hitting it from how can we build

> something to solve a problem versus let's just learn how to code to solve a problem.

The experience and focus on teaching abstraction via the design process punctuates the complex nature of teaching abstraction with both hardware and software.

Programming languages, both text-based (also called line code) and block-based were described as vehicles for learning abstraction. An experienced CS teacher of 13 years stated that using block-based, drag and drop, coding was easier for students to grasp the concept of abstraction. She explained,

> I felt like at least when I taught CSP a couple of years ago the fact that you know when you're using something within the abstraction, when kids built a block and then they used blocks that they had already built in a new block that they were building they could kind of see that more than just in the line code.

Another teacher concurred explaining the difficulties of line code,

> Everything was right but the syntax and it just drives you crazy because you don't have a colon in the right spot or a semi-colon or you know you use parentheses when you're supposed to use brackets. And I really think that introduces a level of frustration that doesn't necessarily need to be there especially when you're trying to develop some sort of basic ideas. So, I'm really coming around to the drag and drop world.

All the high school introductory CS teachers used some type of drag and drop programming language such as Snap, Alice, or AppInventor. However, two AP CSA teachers noted that their Java students didn't start to develop and truly understand abstraction until they started writing longer more complete programs in the second semester of their year-long courses. One teacher explained,

> I think when we talk about abstractions and specifically kind of what they are and programming is when the students start to see a little bit of the bigger picture and feel as if they're actually writing a program that can do something as opposed to just working on the nuts and bolts of syntax and the language and everything.

Providing repetition of both the concept and skills related to abstraction were suggestions teachers made for new CS instructors learning about abstraction. The introduction of the concept of abstraction is a requirement in AP CS Principles, a course designed as an introductory high school survey course for CS (College Board, n.d., 2019). Providing instruction in block-based coding to introduce the concept of abstraction and then repeat learning abstraction with line-based coding was mentioned by high school teachers in three states. One teacher explained, "Yeah, I think it's definitely not a bad thing to introduce the word early and then keep coming back to it and spiral around again and again." Another teacher who had a dual bachelor's degree in CS and math education shared that he couldn't remember hearing about the word abstraction in college although he was definitely taught to hide data and make his code efficient. He shared again stressing repetition that the vocabulary word abstraction didn't necessarily have to be taught immediately but could be explained later on in the CS learning progression.

Collaborative learning was a classroom management tool all teachers described allowing them to engage students, manage student learning differentiation, and facilitate learning. One teacher explained how she used the "cup system". Instead of students raising their hand for help, they had a set of four cups on their computer. If they put a red cup on their computer, the teacher knew they needed help. If students put a yellow cup on their computer, they were busy working independently. If students put a green cup on their computer, they understood and were finished with the task. A purple cup on the

computer meant the student understood the task, finished, and was available to tutor other students.

A middle school teacher described how she explained to students that coding was difficult and that some students were going to get it easily but others had to work hard to get the material, which didn't mean they couldn't learn but that they had to work harder. The teacher gave students a finite set of time on projects. On the last day of the project, she would have students list on the board who had finished and who needed help. The teacher would ask students who finished to help the students who had not finished and shared that the students really liked this part of the project progression. If both students working together could not solve the project, the students would put a check mark on the board indicating that they needed the teacher's assistance.

Pair programming was also mentioned as a collaborative instructional technique employed to teach abstraction. One high school teacher shared that he used the pair programming designation for one student as the navigator (not actually typing but suggesting) and the other student as the driver (the student actually typing). Another teacher described how using pair programming allowed him to team students who understood abstraction or could use it somewhat with students who needed more assistance.

Alluding to the advanced nature of learning and demonstrating abstraction, all of the high school teachers and one middle school teacher, who focused on programming, mentioned being unsure how to help students who did not understand abstraction attain proficiency. One teacher who was working on his Master's in CS explained,

> But it's that next higher level of conceptual thinking that I'm struggling to teach them, which is why if I didn't give them direct prompts would they be able to see exactly where abstraction fits into the program and how it can help them and what they should do as opposed to me feeding them step by step instructions.

Teacher Perceptions of Student Ability

Teachers' experience of student ability influences the instruction of abstraction. Consistently, teachers mentioned being challenged by students who easily understood programming and students who struggled. Teachers with some experience at the elementary level noted aspects of abstraction are taught in the elementary grades. However, the majority of teachers felt that abstraction could be learned in middle school. One teacher, who instructed juniors and seniors, said he recognized some students had a proclivity towards programming and abstraction whereas others did not. All other teachers shared they felt any student could learn abstraction. Teachers also shared specific examples of students demonstrating abstraction.

Descriptions of how students understood abstraction varied. One teacher explained, "Some kids think about it naturally; a word will represent something later. Kids who look at something more concretely have a harder time."

Most teachers ascribed to the idea that the ability to learn and demonstrate abstraction was student-dependent not based on grade level. One teacher explained, "So, I think it is a matter of more where they are intellectually than a specific grade."

Another teacher explained, "I don't want to say it's an innate ability but I get these students who are much better at reasoning and students who really struggle with that."

Another teacher explained about students' ability to learn abstraction, "And I think students who are really strong with their logical step by step reasoning end up being much better able to."

Thinking skills related to abstraction are taught in math in elementary school. A teacher who taught elementary school noted, "In first grade they have to be able to recognize different patterns and things. Even in Kinder [sic: kindergarten] they start looking at patterns and doing pattern recognition." She described learning sequence through learning addition in first grade and learning abstraction via a process for simplifying addition by learning multiplication in third grade. Another teacher agreed students in elementary grades might be able to learn aspects of abstraction and explained, "But I think there are parts and skills taught in lower level grades."

A middle school teacher thought the concrete nature of elementary student thinking might facilitate student knowledge of computer coding skills and remarked, "They just want to make the duck walk…or in the case of the dance party they just wanted to see their little you know three cats with cute pants dance instead of two cats or whatever." Other teachers noted that learning algebra, as previously mentioned in the results, facilitated learning abstraction. Regarding the mastery of abstraction and grade level, one teacher concluded, "I think them truly understanding what it is doesn't come until higher level grades."

Teachers provided examples of students failing to demonstrate abstraction as well as applying abstraction. In an introductory course, talking about a student's inability to create effective representations via naming a teacher explained in his discussion with one

student who protested naming a function correctly, “And I said, oh well forgive me. I didn’t see “list picker 2” as the leading location button. Whereas, everyone else had named it delete button or something like that.” A high school Java instructor shared an example of students failing to apply abstraction,

> And a lot of students set the values in the fields explicitly with each constructor rather than calling other constructors from or rather them calling the constructor from the square constructor and then calling the square constructor from the new args [sic: arguments].

Other teachers mentioned that it was hard sometimes to figure out what questions to ask students who didn’t understand, and even if the teacher did ask a question, sometimes students still wouldn’t know how to answer.

Describing how her students employed AI features in constructing chat bots as a group, a teacher shared an example of successful abstraction, “…if it is interacting with somebody it has some answers and if it sees the word mother, or brother, or sister or whatever it is, it will then ask a question, ‘will you tell me about your family.’” Another teacher shared how one student successfully applied abstraction,

> They were just trying to organize their work better but what I think they effectively did and in any large program you’re gonna have lots of files but what they effectively did without me prompting them to was to sort of take this thing and get it to work and then just push the files away into this file import that works but not have to worry about what’s in the file.

Teachers hypothesized that abstraction is difficult for students because they lack the experience, background knowledge, the inability to see patterns, and the inability to organize information. Math teachers noted that unlike math where students had years of practice, CS was almost always a completely new subject for students. Regarding

including real world experiences and activating background knowledge one teacher explained,

> I feel like the more hands on and the more sort of real you can make it with manipulatives the better any teaching is. I feel like it's just sort of good teaching to give them as many physical models of these ideas as well as actual models.

Another teacher mentioned how the robotics curriculum she used included games students knew, such as hot potato. However, teachers mentioned that it was difficult to get students to solve problems with minimal direct instruction in their courses. Another teacher shared, "Yeah, you have to understand the ideas in order to understand the hierarchy of ideas." A veteran 13-year CS teacher shared another possible reason that students struggled with abstraction, "When you start with those basic patterns, one of the biggest things that I have found is that kids struggle with pattern recognition, kind of like they struggle with number sense in the quantity and place value."

The teacher with the most experience in this study, over 25 years in education, shared that educators used to focus on teaching the acquisition of knowledge. She said she learned in school by copying outlines from teachers as they wrote on the board. She replicated writing outlines to learn in college when she studied textbooks. She further explained that students today probably learn by outlining and organizing information less than in the past because education has changed,

> You know the interesting thing is, I think the reason I got that is that when I was taught way back when before Noah came over on the Ark… back then we didn't have the Internet, so it was all about learning information. Today it's more about finding information and analyzing it.

A CTE teacher who was focused on helping his students get ready for employment provided certification trainings for CompTIA encouraging students who had

more of an interest in hardware to focus on learning about information systems rather than programming. He also shared that business analysts and systems analysts have to know a great deal about all aspects of CS and especially abstraction to connect client goals with their team's design process. He shared that although not all students might be interested or talented with computer programming, knowing some degree about computer programming and abstraction would serve them as a future employee.

Research Question 2: How do teachers determine objectives and competencies for teaching abstraction in computer coding?

Three teachers interviewed did consciously plan and determine objectives for teaching abstraction in yearly curriculum, daily projects, and rubrics used in assessment. Most of the teachers inadvertently or unconsciously addressed abstraction relying often on their curriculum to address the topic. The three teachers who consciously planned to include abstraction, the discrepant cases, were either required to teach abstraction to prepare their students for the AP CSP test, or they had already learned about abstraction in their college coursework and professional development. These three discrepant cases underscore the variable of experience learning about abstraction both as a college student and in teacher professional development. The following stories and quotes will illustrate the variety of teacher experiences directly or indirectly determining course objectives and competencies for abstraction in CS.

Curriculum

Teachers use curriculum and objectives to determine instructional activities and assessments. Because the majority of discussions in the interviews that addressed

curriculum and objectives for abstraction were contextualized around instruction, the following examples from teachers will also illustrate results for the instruction of abstraction. Teacher participants in this study utilized online tutorial programs, such as Code.org, AppInventor, Project Lead the Way, Nand2Tetris, and teacher created tutorials to provide direct instruction and differentiate instruction. Middle school teachers relied on Agent Sheets and Scratch focusing on game development, as well as robotics. High school teachers used Snap and even Logo and TI basic calculator programming as drag and drop or block-based coding curricular resources and HTML, Python, and Java as text-based coding languages. Intersections between Math, Science, and CS also provided opportunities to teach abstraction. Teachers discussed ways to make curriculum engaging, accessible, and interesting as much as possible.

The idea of artificial intelligence, AI, was used to both describe teaching abstraction and engage students. An AP CSA teacher used the idea of chat bots in a lesson and explained how she engaged her students regarding features of cell phones that are attuned to their voices, “So how many of you have Alexa at home and isn’t it kind of creepy to know that something is listening to you all the time?”

And further,

> Well I think it's fascinating because I'm wanting them to think beyond just expecting, you know oh gosh somebody really smart did this. And so therefore all this must be right. And my approach to that. Is more. Well let's think about where this came from. Look at the people that you know created Watson. And then there's this funny video that I just show a clip of. And it's two chat bots interacting with each other. They said wow it's really quite humorous the way they respond back and forth to each other and then they start talking about God and so one of the chat bots says Do you believe in God. And the other chat bot says yes and the other chat bot says Oh well then you're a Christian. And that chat bot responds

> with. No, I'm not a Christian. I specifically chose that example because I just wanted them thinking about morality and ethics.

Presumably, the teacher meant instructing about morality and ethics of chat bots and AI.

Two teachers mentioned standards which guide the creating of course curriculum and objectives. One teacher mentioned that he had more flexibility in his course curriculum because he was not teaching in a state that had adopted Common Core standards. Another middle school teacher shared implementing multiple standards including the International Society for Technology in Education (ISTE), "So, our district has priority standards and innovation standards as well as ISTE standards."
Both the ISTE standards and the Computer Science Teachers Association (CSTA) mention abstraction in their definition of computational thinking (CSTA, 2019; ISTE, 2019).

The AP teachers in this study used Project Lead the Way, Stacey Armstrong's A+ CSA, AppInventor, and self-developed curriculum for AP Computer Science Principles (AP CSP) and AP Computer Science A (AP CSA). The AP CSP test requires an abstraction task, so all of the AP CSP instructors described introductory abstraction lessons in which the topic was introduced, practiced and then later revisited throughout the course. AP CSA instructors agreed that abstraction was the nature of Java and object-oriented programming. Even though the AP CSA instructors did not all use abstraction specifically as a vocabulary term, they taught the skill of abstraction, required abstraction in their coding assignments, and directly or indirectly assessed abstraction.

Several teachers described game design using Agent Sheets, Scratch, Snap, or AppInventor as a way to engage kids and teach them abstraction. One teacher mentioned she included challenges encouraging abstraction skills,

> I have them speed it up when it reaches a certain score or throw up congratulations you won something like that and then from there it's up to them to sort of puzzle it out how to do it either independently or through pair programming.

She also described how she taught students to make a procedural abstraction in the game Frogger called "anticheat",

> So, the question is how do you what's the most elegant way to prevent the frog from cheating. And the first solution the kids come up with is to say to write a rule for every instance where the frog can cheat and there are like six of them or seven of them. Right. And the idea is can we, can we get that down to one rule? And eventually we'll talk it through. And a kid will figure it out. Here's the way to do it. You put you put an agent underneath all of those and if you say if the frog is somewhere above them the game resets with one rule.

App development (for mobile phones or tablets) was an additional curriculum option that both high school and one middle school teacher used. AppInventor and Google Android Studio were used at the high school level and Swift was used at the middle school level. One teacher used AppInventor as the primary curriculum for the AP CSP class he taught.

Middle school teachers offered instruction with a variety of robotics including Lego, Sphero, Ozobots, and Edison robots. Both teachers were designated STEM instructors and combined engineering, science, and math along with programming instruction. Neither of these teachers had specific examples of teaching abstraction in computer coding. One teacher explained that teaching robotics brings technology,

engineering, and math together, “So, in my mind that’s the most useful way to teach abstraction is actually building something that they can say oh I imagined that.” STEM teachers also used 3D printers and Tinker CAD to teach computer programming, the design process and inadvertently abstraction. An advanced high school CS teacher shared how he taught students to build circuits as a way to help them understand the levels of abstraction in hardware and software,

> For example, I have them build an adder circuit and have them build a half adder and a full adder and then they use them both the half adder and the all full adder to create a larger four bit and then an eight-bit adder.

See Figure 13 for a description of a full adder circuit used in the arithmetic logic unit (ALU) within the central process unit of a computer.

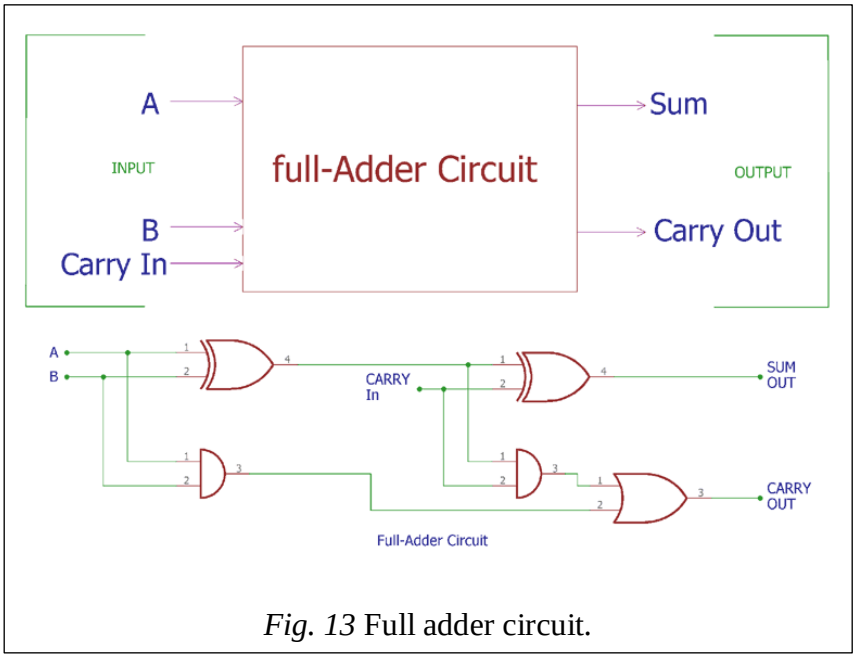

Fig. 13 Full adder circuit.

Teachers also employed cross-curricular connections between Science and Math to engage students and help them learn abstraction. A middle school Science teacher and after-school STEM advisor, offered programming as a choice in each of her middle

school science modules. This teacher just learned to use Rasberry Pi's and was excited to offer a unit next year where her students will use the Rasberry Pi's to build sensors and measure biological and weather information. She explained several examples of how students used computer coding, not specifically coding with abstraction, to demonstrate their science knowledge,

> One kid was working on a Scratch animation that shows an ocean scene where he shows physical and chemical changes. It's really cool. He drew an ocean scene and then this creature comes out of the ocean and eats the plastic bottle that's on the beach and the plastic bottle shrinks and then it zooms into the stomach.

Because this teacher grades on student reflections and student understanding, she is more concerned about students' Science knowledge than working computer code. Referring to models in computer programming, she explained,

> Stuff that works is always important but at the same time like if I'm having for instance in my astronomy unit I have them build models. It could be a working model or it could not be a working model.

Science field trips and connections with community members who understand and demonstrate computer coding are another way she has made connections with Science and abstraction. She explained, "We have an astronomy club here and they support STEM. We went on a trip and learned about the technology and coding behind these amazing telescopes which they remotely run." A physics teacher who also taught AP CSA, explained that he used test tubes and test tube racks in an unplugged activity (instructional activity not using computers) to demonstrate arrays, a possible way of hiding data or demonstrating abstraction. Another Science and AP CSPrinciples teacher connected Math and Science having students input body mass index variables for weight and height in JavaScript notation.

Seven out of the 12 teacher participants had taught Math or were currently teaching Math. Four of the teachers in this study also had taught or were currently teaching AP Calculus. Math was one curricular aspect of teaching abstraction that was mentioned in most of the interviews.

One teacher described a success teaching abstraction with a student creating an independent project in Snap to demonstrate an International Baccalaureate (IB) math concept,

> And I said well she's using block code. The syntax isn't a problem if she can figure it out mathematically and logically she can do it. Like, it's all about the problem solving. With block coding, it's not about the syntax. And she ended up writing a program that graphed different types of functions from math and the abstraction that she used in it was absolutely amazing. In fact, she had one of the highest scores that had ever been given at the high school with an AI in math both from that IB teacher and on the final score from IB.

One teacher used the idea of a square root on a calculator as a metaphor for abstraction, "When you do math, that square root is going to give us the square root. We don't know how it does it. We just know that it is going to give us the square root." Another teacher asked her students to handwrite code line by line "kind of like you do when you teach long division." Several teachers used the logic and math problems on the Project Euler website. One teacher explained she would have students write computer code to demonstrate their solution to the Project Euler problems, "I make them do it handwritten. Then I'll let them code it and actually check to see if the answer in their program output is the correct one in Project Euler."

Another math metaphor for abstraction came from a teacher who described how rote knowledge of quadratic equations were an abstraction allowing students to complete

complex calculus problems,

> So, when a student's doing a calculus problem, they can actually think about the larger context of that calculus problem and not worry about the smaller algebraic steps in the mix even though those algebraic steps…they don't have to put a lot of mental energy toward them.

The math and CS teachers compared the similarities and challenges of teaching algebra and functions in both math and CS. According to one teacher, many students struggle with understanding the basic principle of representation for the value X in algebra,

> If I had a dollar for every time a student asked me what X was, I would be a millionaire. X is a holder. X is something that holds all numbers. More, X is something that you operate on and place an operating number.

Because this teacher also knows that the terms function in math and function in CS mean slightly different things, he uses teaching functions in math to introduce the idea of naming functions as abstractions in CS. He shared how he explains this to his students and extends the concept of functions from math to CS,

> We're going to write lots and lots of functions so we're going to be super lazy and call them just all of the function. Then in another context you will know instead of using C of T, I might use cost and time as the inputs and so show them how the functions are not. Not necessarily show them but sort of route to the way that functions in mathematics are related to the things that they'll learn, the structure they'll learn in programming, later on. So, I think that's one of the ways that with an eighth-grade class I really build the idea of abstraction and functions into math as a foreshadow for what I'm going to do computer science.

Four teachers mentioned they found that students who already knew algebra could learn abstraction in CS fairly easily. Another teacher noted that in some ways math was easier to teach but harder to see progress in than CS and abstraction because, "Math does take a long time to acquire and lots and lots of necessary skills that they don't necessarily see the immediate results." Another teacher remarked that students were more engaged in

CS classes because the curriculum was new and students chose his CS courses as an elective. He found that he could challenge the salutatorian of his school who had over ten years of Math, Science, English, and Social Studies, as opposed to over one year of CS which she found challenging.

Teachers mentioned helping students learn abstraction employing geometry. One teacher had students build squares in Java, then triangles, then rectangles, and then put all the shapes together in a program to build a house. Other teachers used squares and polygons to demonstrate recursion and procedural abstraction. Additional elementary math and CS cross-curricular connections were noted, "And when I taught third grade mathematics and I was teaching multiplication we actually use the term array with the kids and it's a one by five."

One teacher who did not teach an AP CS class at his school because it was a smaller school with many IB courses, was excited about a free online course designed for introductory college CS called Nand2Tetris (free and online) that simulated computer hardware and software design essentially teaching all levels of abstraction over the course of a year. He explained,

> You build up the hardware of a computer, and so you start with NAND gates and you build all the elementary logic gates so and or XOR and then you use those to build ALU and memory and then you build a CPU and then you basically build from all of those pieces a general-purpose computer. It's all simulated online, well in a hardware simulator. You download the hardware simulator on your computer and then you can write little short lines of code that basically connect these smaller chips together.

The second semester of the Nand2Tetris course takes students through learning to write assembly code, binary code, and on to programming language.

Teachers mentioned additional commercial, course, and community curricular resources. Two AP teachers mentioned regularly contacting mentor AP teachers. Facebook groups and local CSTA chapters were also mentioned as resources. Teachers accessed materials and suggestions on Piazza, Beauty and Joy of Computing, and the College Board AP listserv. Online resources such as W3 schools, CyberPatriots, and the NASA Hunch Program were recommended as teaching sources for abstraction. Stacey Armstrong's A+ AP CSA curriculum was recommended along with certification courses, such as CompTIA.

Research Question 3 – How do teachers assess student abstraction skills in computer coding?

Teachers approaches were mixed regarding assessing abstraction using formative and summative means. Many teachers placed emphasis on their classroom conversations with students to determine student knowledge of abstraction (as well as to offer instruction through dialogue). Teachers shared employing metacognitive tasks to assess abstraction knowledge. Teachers interviewed in this study used several means to determine student abstraction knowledge and skill including formative, summative, and metacognitive assessments. There were no distinct discrepant cases.

Assessment

Assessment is the method teachers use to identify student ability and the success of their instructional efforts. The previously mentioned teacher observations of student ability arose from formative assessments, or observations, discussions, and informal student assessment. As evidenced by the previous results, teachers relied on formative

assessment to understand student ability and the effect their instruction had on student learning. Only three teachers mentioned providing tests or multiple-choice quizzes in the classroom, formal summative assessments, aside from the formal assessment of abstraction on the AP CSP test. Teachers did mention that abstraction was included, although not always called abstraction, on their project rubrics.

Regarding the assessment of abstraction, one teacher explained, "Most of my tests and quizzes are AP type questions from the College Board. I think it is a natural part of any sort of programming assignment." Another teacher who graded 20% on participation and 80% on projects shared how abstraction was included in her grading, "We'll definitely talk about it, and so it's a part of their grade on tests or projects." Teachers included the topics of "managing complexity" and "elegant simple code" on their project rubrics. One teacher said she could give students feedback on abstraction in their coding but felt less confident creating assessments and relied on AP practice questions. Another teacher shared, "AP CSP directly assesses abstraction. Students have to know what it is and how to demonstrate it. Science assesses abstraction with modeling through chemistry labs that show formulas for say gasses that are applied in a variety of combinations."

Assessment was the most difficult research question about which to get follow-up information or examples from teachers. Another teacher shared that the online tutorial course he was using required students to complete one module before moving on, which was a form of summative assessment. He utilized questions first and later discussions to aid students who were unable to complete modules. Regarding the challenge of teaching abstraction, one teacher who started teaching through an alternative route and did not

have formal university courses in education shared, “I think it is an extremely difficult thing to assess because in my current view of abstraction, it’s much more of a thought process.”

Research Question Context – Teacher Experience

The overarching theme of experience was shared as teachers described their pathways to becoming CS teachers, degrees of self-efficacy teaching abstraction, their lack of specific courses or professional development regarding abstraction, and their requests for future professional development. Only two of the teachers in this study had taught CS for more than 5 years. The majority of teachers with one to five years of experience were teaching a combination of new courses and courses that they had been teaching. Regarding the demanding nature of teaching technology and simultaneously learning new course material, one middle school instructional technology teacher

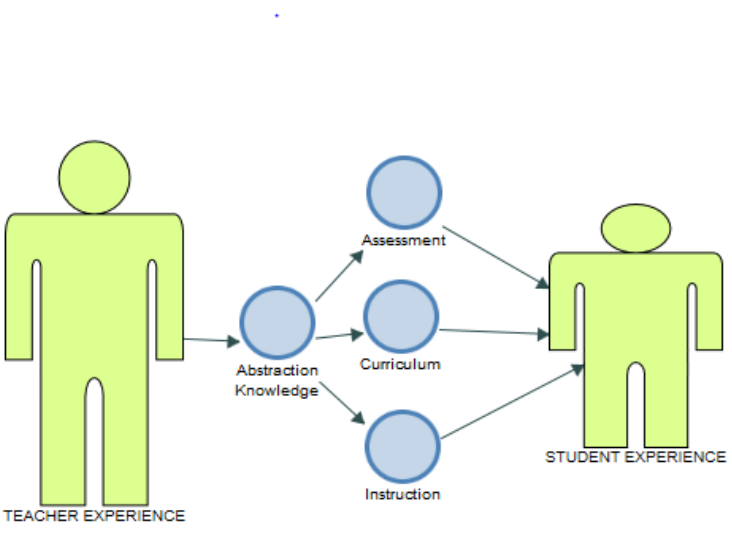

Figure 14. Teacher experience and parent themes

remarked she was confident that teaching abstraction would get easier,

> As I'm as I'm learning and figuring all this out it will be more comfortable to be able to do that. I also believe though in a job like this and with technology it's constant. You're learning, you're changing, you're trying to figure it out, so trying to make that or using that is just something that will be ongoing.

The relationship between the teacher experience, parent themes, and the student experience is illustrated in Figure 14.

Learning with students as opposed to be the expert was another common experience teachers shared. One teacher explained,

> I have had to come to terms with no longer being the expert in the room. And that was a hard shift after. You know 20 plus years of being an English teacher and being the know it all. And then all of a sudden kids ask me question I'll point to somebody across the room I said you know that kid over there he's really good at those. Let's get him over here for you. So, we're all learning together.

Regarding self-efficacy, a teacher explained feeling challenged but enjoying the experience of teaching a new content area that she did not know as well as she did math, her main teaching area of expertise, "It's been a challenge but it's those moments I have so many moments where I stop and just observe and think, this is the most amazing thing that I'm doing." Another teacher shared that teaching abstraction was difficult initially, "First dealing with it was kind of uncomfortable before I really felt confident." Another teacher shared that he felt confident teaching most students but not as much with students who struggled with abstraction. He explained, "I feel fairly comfortable with it. I guess I have sort of a one-dimensional way of teaching. I don't feel I have a good way to teach it to my kids who struggle." A physics teacher shared feeling confident about intuitively teaching abstraction in the moment because he understood the concept of abstraction better than directly teaching the computer coding skills of abstraction.

Teachers' pathways and educational background may be associated to their self-efficacy teaching abstraction. Teachers H and K decided to teach CS in college and obtained math and physics teaching licenses due to the lack of CS teaching licenses in their states. Teachers C and J worked in IT before becoming teachers. Teachers H, K, C, and J spoke easily about abstraction and described teaching abstraction more confidently than the other participants. Four other teachers (A, F, G, and I), science and math teachers, took one programming course in college or had a year or less of experience in the software industry. Teachers A, F, G, and I struggled to explain abstraction succinctly and described struggling with teaching students who didn't understand abstraction. Teacher E had a master's degree in Instructional Technology, and teacher I is working on a master's degree in CS. Teachers E and I having had master's level courses in CS or related topics easily discussed abstraction and teaching abstraction, even when the term was somewhat unfamiliar.

Teachers' described abstraction through the lens of their initial content area. A former English teacher shared that teaching writing was similar to teaching abstraction in CS. She was able to use a lot of her strategies as a writing teacher in terms of classroom management, curriculum development, assessment, and engaging students to transfer into CS education. The former English teacher explained,

> We have one lesson where I just have the kids just gather around and say here's a problem. We have to solve it together. And we keep talking it through and I'd say OK you've got it down to three rules. Can we get it down to 1 - 1 line of code? And in fact, it's interesting because I did the same thing as a writing teacher. And it was one of my favorite things to do was to teach kids how to cut the fat out of their writing.

A math teacher shared how teaching students concepts in math was similar to teaching the concept of abstraction in CS. He postulated that the concept assisted in learning future skills. The concept of abstraction even became an abstraction in the learning process making learning easier and more efficient. Regarding teaching algebra and using the concept as a learning abstraction to facilitate learning math skills, he explained, "Inevitably the students then forget about or don't have to pay attention to that conceptual understanding every time that they say factor a quadratic."

All teachers shared that abstraction was not addressed, or addressed very little if at all, in the professional development trainings they attended related to CS. The teachers who attended AP CSP professional development said that abstraction was covered, but they still didn't have a solid grasp on what abstraction was. One teacher shared that he understood the entire curriculum scope and sequence of math from K-20. He explained, "Sometimes my students asked me what's after AP Calculus and I said well more calculus." However, he couldn't say what the abstraction curriculum looked like before his AP CSP class nor afterwards in college.

Several teachers mentioned support from their district, their principals, their communities, and students' parents was helpful in learning effective CS teaching skills. One teacher explained, "A dad of a student who came in was a programmer and he would just sit in the class and help me like just help the kids troubleshoot and problem solve." Another teacher shared how financial support allowed her to expand her curriculum, "So now a couple of years later just from some private donors we have a class that uses Lego robotics." A teacher explained how supportive principals influenced her effectiveness,

"So now I'm enjoying kind of having free rein to grow the program at my school and I have a lot of support from my administration."

Another teacher offered that support from both administrators and teachers in other content areas was helpful, "So, I'm very lucky that both my admin team and my math department chair supported me in this and we've kind of been adding one class per year each of the last three years." Two teachers shared how district level support from school boards was crucial for their courses. One teacher shared that she regularly attended school board meetings and was consulted on districtwide IT and CS curriculum implementation. Another teacher explained, "It did take my school board a little bit to get on board." However, then she was able to take a lead role in training other teachers to lead STEM after school programs in her district.

Teachers' suggestions for professional development ranged from very broad general introductions on the topic to more collegial sharing teacher to teacher. One teacher explained that any type of course on abstraction would be helpful, "I think just understanding what it is to because I think a lot of teachers struggle with what it is." Another teacher observed, "I think there is a lot of room for professional learning." Another teacher suggested, "Just offering it in general with any sort of programming. I think you know giving them the opportunity to learn you know concepts that aren't surface level and aligning a little bit of resources behind that."

Another teacher requested, "Some good awesome lesson plans for that because it's something I don't feel super confident in." More specifically, teachers requested coding and abstraction professional development relevant to their courses. One teacher

suggested, “I think it would be important to have some professional development around abstraction/coding for STEM teachers.”

Several teachers shared that a training where teachers were taught to experience the syntax related to abstraction in several programming languages, from drag and drop languages like Scratch to AP CSA languages like Java, would be helpful, especially focusing on data structures, arrays, encapsulation, and object-oriented programming. Another teacher suggested providing a wide array of learning activities because he liked trying learning experiences that were completely different. A STEM teacher suggested offering professional development for abstraction using a three-dimensional lab approach focusing on engineering design and rubrics. All of the teachers were interested in some type of professional development related to abstraction.

Summary

In conclusion, the 12 teachers interviewed in this study shared their experience of teaching abstraction from primarily high school and some middle school CS courses. Information from the one elementary teacher, the outlier case, was generally excluded from the results due to the lack of information from other elementary teachers. I employed rigorous repetition and careful analysis of all themes and data to ensure dependability, confirmability and transferability.

In relation to RQ1 (What types of instruction do K-12 teachers find most effective for teaching abstraction in computer coding?), teachers shared that a variety of dialogue techniques, collaborative learning techniques, direct instruction, and contextualized instruction including project-based learning were helpful. Teachers also mentioned that

utilizing preexisting curriculum such as AppInventor, Nand2Tetris, and Project Lead the Way, or even self-made tutorials, provided teachers with a foundation from which they could offer advanced instruction and guidance related to applying abstraction. Teachers relied on AP test criteria and preexisting understanding of teaching students to use elegant, simple, or efficient code in relation to RQ2 (How do teachers determine objectives and competencies for teaching abstraction in computer coding?). Teachers utilized primarily formative assessment through dialogue to assess abstraction (RQ3 – How do teachers assess student abstraction skills in computer coding?). Although a few teachers did employ summative assessments in the form of project rubrics, quizzes, and tests. Some teachers chose to put more emphasis on assessing abstraction via student self-reflections versus abstraction in computer coding.

In relation to the general research question guiding this study (How do teachers decide what effective instruction for teaching abstraction for computer coding is?), the overarching theme was that the more experience teachers had with their course material, with programming languages, with teaching CS, with CS courses, the more teachers found ways to explain abstraction, instruct abstraction, and assess abstraction.

Chapter 5: Discussion, Conclusions, and Recommendations

The purpose of this descriptive qualitative inquiry is to illuminate the teaching experience regarding abstraction in K-12 CS and examine effective ways to teach abstraction. This study also provides variables, such as professional development, experience with course content, and previous teaching content areas for future quantitative research. Insights comparing how the results confirm, disconfirm, or extend the theoretical framework and literature review are offered in this section to help educators better understand the effective instruction of abstraction. Finally, avenues of future inquiry indicated from this study are offered. In general, the results of this study show that CS teachers do not have a common definition of abstraction. Abstraction in CS is a multifaceted concept, attributed to both hardware and software, and used as a noun, a verb, and an adjective. Teachers generally understood and taught the concept of abstraction but were not as confident teaching all students abstraction and assessing abstraction. Abstraction is a topic that is a ubiquitous concept requiring knowledge of many aspects of CS. As teachers become more versed in abstraction, they will become better CS instructors.

Interpretation of Findings

Defining Abstraction

As reported in Chapter 4, the majority of teachers interviewed in this study did not have a succinct definition of abstraction. Four out of five AP CSA (the most advanced level of AP CS taught in high school) teachers interviewed shared that they had a sense of abstraction but did not actively teach it or assess abstraction. Two of these AP

CSA teachers had taught the course for four or more years. It is understandable that an advanced topic such as abstraction may take a while to master for teachers new to a content area, such as teaching Java (a complex programming language), the programming language taught in AP CSA, due to the demanding nature of both CS and Java. Other teachers used abstraction as a noun, verb, and adjective which indicates teachers had an understanding of the multi-faceted nature of abstraction. The majority of teachers requested specific professional development on the topic of abstraction with direct applications and demonstrations in a variety of programming languages. Perhaps, the conceptual framework of abstraction is too large and should be broken down into smaller more meaningful concepts and skills for successful integration into K-12 CS education.

Comparison with Theoretical Framework

Overall, the results of this study confirmed the theories and frameworks incorporated into the broad theoretical framework detailed in Chapter 2. The only theories or frameworks that teachers mentioned by name were computational thinking and Piaget by three out of 12 teachers interviewed. One teacher had specific professional development related to the instruction of computational thinking, of which abstraction is designated as a foundational principle (Wing, 2008, p. 3718). Therefore, according to the results, the teachers interviewed in this study did not share consciously incorporating the theories and frameworks described in Chapter 2. The results from teachers do indicate some theories and frameworks might help teachers understand the instruction of abstraction.

Philosophy, Abstraction, and the Teacher Experience

Ontological and epistemological interpretations of the relationship between humans, computers, and abstraction seem interestingly similar to the experience teachers had instructing abstraction as a skill and a concept. The majority of teachers related that students who learned skills first were later able to demonstrate some foundational algorithmic, syntactic, and procedural programming skills demonstrating an understanding of abstraction as a concept. The implication for the instruction of abstraction from Fichte (as cited in Whistler, 2016) was that teachers should employ metacognition in order to develop deduction and induction thinking skills. It appears that helping students build background knowledge and basic skills needed to produce abstraction facilitates students activating background knowledge through metacognition resulting in learning abstraction. If students lack essential background knowledge, they have no ontological markers to use for analysis, evaluation, application, and creative problem-solving. As teachers in this study noted, when they helped students build background knowledge, students were then able to epistemologically apply their background knowledge to demonstrate abstraction.

Student metacognition provided teachers with formative and summative assessment information regarding ontological and epistemological background knowledge. Teachers shared several ways they encouraged student metacognition through dialogue and written self-reflection used as assessments, but teachers didn't focus on developing student awareness of expressing thoughts by programming computers. It appears that Gobbo & Benini's (2012) and Ben-Ari's (2001) input on

extending human ontological identity through computing was not directly acknowledged by teachers at all. Teachers did talk about enjoying watching students share joy in programming successfully, implying that student self-efficacy more than the student intrapersonal awareness of their relationship with a computer as an inforg may be more important to teachers in teaching abstraction. Student motivation and self-efficacy may be more important for learning abstraction than philosophical frameworks inviting ontological and epistemological reflection.

Ultimately, applying abstraction elegantly in computer coding requires learning abstraction as a concept and a skill. The concept of abstraction could be equated with ontologically understanding the computational solution, the exact nature of the solution. The skill of abstraction could be equated with epistemologically understanding the computational solution, how the solution could be executed. Declarative knowledge is also aligned with ontology (Marzano & Kendall, 2007). Whereas, procedural knowledge is more aligned with epistemology. One teacher mentioned teaching data abstractions and procedural abstractions which are also respectively similar to ontological/declarative knowledge and epistemological/procedural knowledge.

As computers help humans to solve problems and technology becomes more complex with layers of abstraction, teachers and students may benefit from thinking about teaching and assessing abstraction focusing on both the skills and the concept of abstraction, building both declarative and procedural knowledge. Furthermore, if teachers want to focus on teaching the concept of abstraction, they might focus on contextual instruction because contextual instruction can help students build and activate

background knowledge making connections that facilitate the understanding of the concept of abstraction. Teachers might focus more on direct instruction if they want to help students understand the skill of abstraction. With either the concept or the skill of abstraction, both direct and contextual experience were reported to be helpful from teachers participating in this study. Possibly, alternating between concept and skill as several teachers reported, returning to the concept of abstraction periodically as programming skills are developed may be the most effective way to help students learn abstraction.

Inviting teachers to understand, discuss, and consider creating lessons around potentiation, the inforg, epistemology, and ontology may actually be more helpful for teachers than students allowing them to gain an understanding of abstraction from multiple vantage points. All of the teachers in this study shared that they have little to no experience discussing abstraction in professional development or even in college computer courses. One teacher noted it is very different to be a CS student taking college courses than a teacher of CS.

Psychology, abstraction, and the teacher experience

The majority of teachers concurred with Piaget (1950) in his assertion that the development of abstraction thinking and imagining a problem and a solution occurs around age 11. A few teachers suggested that aspects of abstraction could be taught to elementary students. The lack of elementary teachers in this study precludes additional implications related to the ability of elementary abstraction skills. All teachers agreed on the point that abstraction ability was student-dependent, not related to grade-level.

Teachers speculated that math exposure and personal interest might help some students exhibit better abstraction skills than others. In any case, it seems that teachers would benefit from recognizing a range of abstraction skills that help teachers differentiate instruction.

Vygotsky's (1978) zone of proximal development theory provides a basis for teaching students abstraction skills in computer programming in elementary school. As with semantic language acquisition, exposing students to a multitude of algorithms and elegant, simple, functional code, may provide students with essential background knowledge required to construct efficient effective programs later on in middle and high school (Chomsky, 2006; Vygotsky, 1986). Teaching students metacognitive skills, induction, deduction, and logical thinking in the elementary grades might also help teachers foster thinking skills necessary for developing proficient abstraction skills in computer coding in middle and high school. The teachers in this study were not sure exactly which thinking skills might be engaged in elementary, middle, and high school – more reason to include a variety of psychological learning theories in professional development for teaching abstraction in CS.

The majority of teachers stated that they utilized subjective formative assessments to determine the extent of student abstraction abilities. According to the zone of proximal development, students would understand abstraction better than they might be able to express it verbally or apply abstraction in computer coding. It may be most effective to assess abstraction utilizing primarily formative assessments and secondarily offer summative assessments in quizzes, on tests, and in projects. Providing teachers with

experiences in professional development related to the zone of proximal learning and speech facilitating thought applied to the instruction of abstraction may help teachers develop more consciously focused instructional strategies.

All of the teachers that were interviewed employed collaborative learning which aligns with Vygotsky's (1986) theory that speech facilitates the development of thought. POGIL, or process-oriented guided learning, was not mentioned as a collaborative learning strategy but pair programming and group projects were cited by teachers in this study. Collaborative learning provides students with opportunities to ask questions, verbalize answers, and develop critical thinking and problem-solving skills. Teachers in this study did not mention intentionally applying collaborative learning as an instructional technique for teaching abstraction. However, collaborative learning that focusses on activities and questions and assessments designed to help students learn abstraction, may provide an excellent environment for teaching and differentiating instruction for abstraction.

Constructionism, computational thinking, and teaching abstraction

Teachers interviewed shared that collaborative learning environments with aspects of constructionism appear to support learning computational thinking and abstraction. Collaborative learning is a necessary environment in constructionism proposed by Papert (1980) as an optimal learning framework for CS education. Another element of constructionism that teachers in this study utilized is student-led inquiry. One teacher noted that the real learning occurs when students ask questions about their work in class. Several other teachers shared how they used student conversations to teach

abstraction, either when students made suggestions demonstrating their understanding of abstraction or asked questions requiring teachers to offer direct instruction on abstraction. Pure constructionist learning necessitates an open-lab for exploration. No teacher offered that an open-lab was a helpful or useful learning environment for teaching abstraction. However, many teachers shared how students in their courses had a great deal of independent time to explore, learn, develop, and complete projects. Aspects of constructionist learning, such as collaboration and student-led inquiry, appear to be useful in teaching abstraction, but an open-lab learning environment was not employed by any of the teachers in this study.

Teachers did not equate computational thinking and abstraction. Only one of the teachers who had taken a course in computational thinking shared a how he incorporated CT as an educational objective in his CS courses. Several other teachers explained that creating a computational solution was the goal of their STEM or CS courses but did not mention ways they aligned this educational objective with instruction and assessments. The conceptual framework of computational thinking from Wing (2006) and Brennan and Resnick (2012) may be useful for helping teachers identify broad objectives for courses but do not appear to be useful in helping teachers identify learning outcomes related to abstraction for lesson plans and corresponding assessments. Possibly, teachers are overwhelmed with teaching the highly complex new content area of CS and incorporating a broad framework such as computational thinking might be too much. New CS teachers related they relied on prescribed curriculum and were learning the content along with their students. Teachers with little or no content knowledge who rely

on specific objectives and outcomes may not have enough content knowledge to effectively develop the curriculum needed to apply a conceptual framework such as computational thinking. Abstraction, a subskill of computational thinking, is also very complex. The results of this study indicate that teachers need more clarification understanding computational thinking, the relationship between computational thinking and abstraction, as well as related guidance creating objectives, curriculum, and assessments. Possibly, teachers similarly need detailed objectives and outcomes by grade level to effectively teach abstraction.

Levels of abstraction, programming languages, and the teaching experience

Two teachers mentioned teaching levels of abstraction, and one other teacher used the word architecture to define levels of abstraction. However, the majority of teachers were unaware of levels of abstraction such as the PKG hierarchy (Armoni, 2013). The PKG hierarchy is a conceptual framework for understanding some of the multi-faceted aspects of abstraction. Parallels can be found comparing the PKG hierarchy with the programming languages teachers described utilizing and the metaphors for abstraction that teachers shared (Figure 15). Teachers shared how they utilized unplugged activities, dialogue, and discussions about what the end-user needs which relate to the problem level of the PKG hierarchy. The focus at the problem level of the PKG hierarchy is on the human experience of the computational solution. I equated unplugged activities at this level because of the human to human element of problem-solving. At the object level of the PKG hierarchy, the computational artifact, both hardware and software, is a grouped and experienced as a thing provides a function. At the object level, the graphic user

interface (GUI), or what is seen on the computer screen is the level of abstraction that equates with block-based coding and robotics. The metaphors teachers shared of driving a car or solely liberal arts versus solely technical college educations, or even understanding how football is played but not understanding app development, relate to the object level where people experience the efficiency of the computational artifact.

The program level of the PKG hierarchy relates to software and the variety of line-based languages, such as Java or Python that teachers reported using in the classroom. The metaphors teachers discussed using for instructing abstraction relating to the program level were two dimensional, the abstract class in Java, and data and procedural abstractions.

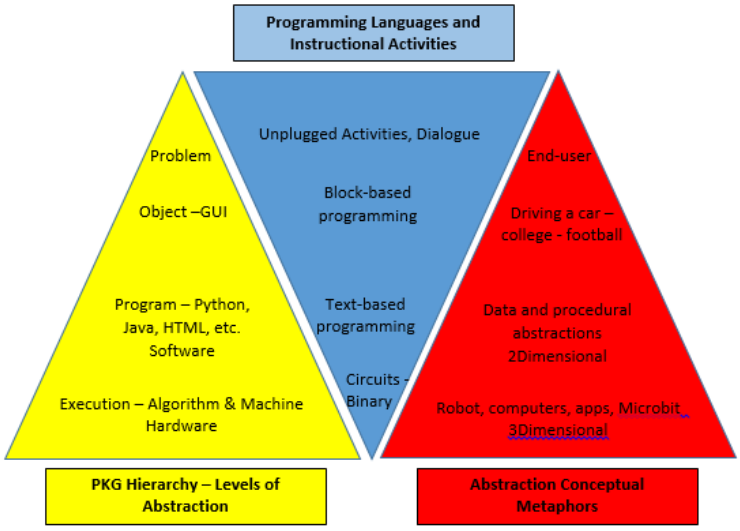

Figure 15. Relationship between PKG hierarchy of abstraction with instructional programming languages and conceptual metaphors

Different types of abstraction in computer coding like data and procedural abstractions, could also be seen as a skill, but because these were mentioned conceptually by teachers

as subsets of abstraction I have related them as metaphors at the PKG hierarchy program level. The execution level of the PKG hierarchy relates to the instruction of circuits and binary code, the underpinning of modern computational devices. Three-dimensional instruction using drones, apps, and Microbits or Rasberry Pi's (small hand-held functional computing devices) are metaphorical applications of the execution level of the PKG hierarchy. It might aid teachers to understand connections between levels of abstraction, hardware, software, human needs, computer languages, and instructional explanations and applications. If teachers learned about conceptual frameworks related to abstraction, such as the PKG hierarchy, they might be able to help students better navigate and develop abstraction skills.

Most teachers with less experience programming and teaching abstraction were confused if algorithmic representations, such as variables, recursion, and classes were abstractions (illustrated in Figure 16). The program level of the PKG hierarchy undoubtedly could include many types of algorithmic abstractions in the universe of programming languages. Teachers of all grade levels would benefit with expert guidance from CS scholars about the exact relationship of representation and abstraction.

Critical thinking, abstraction, and the teacher experience

No teachers interviewed discussed addressing specific thinking skills such as deduction or induction. Several teachers shared the importance of teaching generalization, pattern recognition, and logical thinking in teaching abstraction.

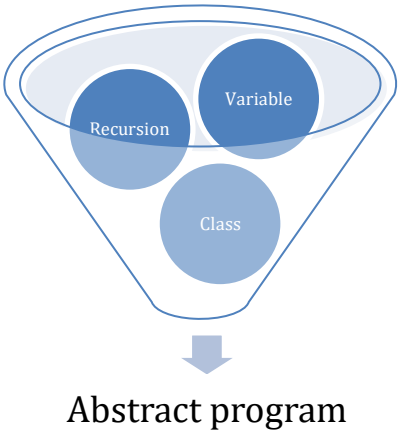

Figure 16. Algorithmic representations resulting in an abstract program

Thus, results from this study confirm the importance of teaching some aspects of critical thinking to teach abstraction, namely pattern recognition (analysis), and generalization (synthesis), and logical thinking (also possibly a combination of decomposition, deduction, and induction). Marzano & Kendall (2007) described abstraction as the process of retroduction requiring both induction and deduction. Perhaps it would be useful for teachers to experience, discuss, then apply the critical thinking skills of deduction and induction in relation to the other thinking skills like pattern recognition, generalization, and logical thinking, in order to understand the array of thinking skills needed for abstraction in computer coding.

Results from teacher interviews do not confirm the multiple pathways to learning CS described in the taxonomy proposed by Fuller et al. (2007) The taxonomy for learning CS shows how through a variety of thinking pathways involving combinations of producing and interpreting some students learn CS more conceptually and theoretically; whereas, other students may learn by experimenting and figuring out code on their own.

None of the teachers shared any awareness of students learning abstraction via multiple pathways. Several teachers did mention students who seemed as if they were "savants" and picked up abstraction "on their own" with little teacher guidance indicating these students were experimenting and figuring out code and abstraction on their own. It may assist teachers in teaching abstraction to understand that students, as Fuller et al. (2007) contend, have multiple pathways for learning CS. Except for one teacher, all interviewed expressed the belief that every student could learn abstraction. Many teachers also described feeling frustrated and unsure how to help students who were struggling to learn abstraction. Possibly, if teachers began to monitor a variety of student preferences and pathways for learning abstraction, it might be easier for teachers guide students who struggle.

Comparison with Literature

The results of this study both confirm and disconfirm a variety of topics including instruction via tangible software, universal design for learning, game-based instruction, utilizing microworlds, STEM instruction, scaffolding, collaborative learning, using rubrics and portfolios, and the ability of elementary students to demonstrate computational thinking. It is important to note that due to the lack of specific research regarding abstraction, the majority of research evaluated in the literature review analyzed studies that investigated computational thinking because abstraction is deemed a subskill of computational thinking (Wing, 2008). Aspects of previous educational research related to abstraction could help teachers gain insight into teaching struggling students and abstraction in general.

Teachers unequivocally recommended using manipulatives (if they had used them before in the classroom), such as Microbits and Rasberry Pi's to teach abstraction. Tangible software has been attributed to elementary students learning computational thinking (Bers, 2010; Kazakoff & Bers, 2012; Wang,Wang & Liu,2014; Zhong et al., 2016). Most teachers interviewed mentioned that engaging multiple intelligences in the learning process helped students. Perhaps if more teachers understand how helping students understand the relationship between hardware and software, teachers will be able teach students about levels of abstraction. Teaching how hardware works may also help students to be able to create computational solutions that operate efficiently and effectively. Teaching students about hardware may help students understand and apply abstraction in computer coding.

Universal design for learning and scaffolding, especially utilizing pseudo code as an instructional technique, have been recommended as instructional techniques for increasing computational thinking (Israel, et al., 2015; Shane & Sherman, 2014). The majority of teachers participating in this study explained that they used scaffolding and aspects of universal design for learning including videos, tutorials, and pseudo code. Additional training specifically focusing on examples applying utilizing universal design for learning and scaffolding with abstraction in several grade levels might assist teachers in providing more effective instruction.

Several teachers interviewed in this study shared how they included games and game-based programming into beginning and even advanced CS classes. Game-based curriculum has shown promise in stimulating computational thinking (Carbonaro et al.,

2010; Lee et al., 2014). Teachers described using games and gaming as a way to contextualize skills needed to express abstraction in computer coding. Students activate background knowledge when programming games they know, such as Connect Four or hangman. Games also have objects and rules which make them helpful for teaching object-oriented programming, data abstractions, and procedural abstractions.

None of the teachers in this study mentioned utilizing microworlds, such as Unity or Second Life, to teach computer coding or abstraction. Immersion into microworlds has been cited as a possible way to help students generate computational thinking (Jenkins, 2015; Reuker et al., 2013). Teachers might be interested in seeing and exploring lesson plans focused on abstraction situated in microworlds. The students at "the bottom of the pack", as one teacher described, who struggle to understand abstraction might learn the concept and skills in an imaginary microworld.

STEM curricula was used by two of the middle school teachers as a way to include CS in the design process. STEM and robotics instruction have been used to engage middle school girls in engineering and improve creativity and computational thinking (Cooper & Haverlo, 2015). The interdisciplinary nature of STEM instruction naturally accommodate project-based learning, contextualized instruction which helps students activate and build background knowledge. Teaching the design process in STEM courses helps students practice logical thinking, problem decomposition, deduction, and induction – all useful thinking skills for learning abstraction. STEM curricula or modules could be helpful in teaching and learning abstraction, especially in the elementary and middle school grades.

Teachers interviewed in this study concurred with theories implicated by Vygotsky (1986), Papert (1980) that collaborative learning is helpful in teaching abstraction. Elementary, middle school, high school, and college students showed improved computational thinking skills when instructors used collaborative learning (Harlow & Leak,2014; Huang et al., 2016; Hu et al., 2016; Porter et al., 2013). Teachers mentioned using collaborative learning techniques such as paired programming with one student designated as a navigator and the other student designated as a driver. Teachers also shared using small groups and agile project management techniques to help students learn the array of tasks needed to create computational solutions. More examples of collaborative learning activities addressing abstraction for a variety of grade levels could assist teachers in providing more thoughtful instruction for abstraction in CS.

Although the majority of interview data from this study focused on the instruction of abstraction, teachers had less information to share about how they assessed abstraction. Rubrics and portfolios have been used to assess computational thinking (Sanford & Naidu, 2016; Zhong et al., 2016). Although teachers interviewed in this study did not have specific rubrics for abstraction in computer coding or in projects, several shared they did require efficient or elegant code in their rubrics. None of the teachers used portfolios to grade students. Several teachers used sample quizzes and AP test problems that addressed abstraction. All the teachers mentioned interest in viewing or learning about ways to assess abstraction.

A variety of research regarding elementary students' ability to learn conceptual and procedural knowledge via nonformal contextual interactions suggests that elementary

students can learn abstraction in CS (Braithwaite et al., 2016; Rittle-Johnson & Schneider, 2014; Szucs et al., 2014). A few teachers interviewed in this study speculated that elementary students could learn some aspects of abstraction. Unfortunately, the one elementary teacher in this study was not very familiar with abstraction and could not offer much input about elementary students' abstraction skills. Teaching abstraction in elementary CS is an entire topic that could use more research.

Limitations of the Study

This study was limited by time, the number of participants, the predominance of secondary teachers, and the lack of student artifacts. Because abstraction is an advanced skill that several teachers mentioned saving to teach until the second half of the school year, I may have been able to recruit more participants who were actively teaching abstraction if I had recruited in the spring rather than the fall. Teachers were interviewed twice in one month. If teachers were interviewed four times, quarterly, or even monthly over the course of an entire school year, the data might have been more representative and more thorough. Twelve participants were recruited, the majority being high school teachers. More middle school and elementary teacher participants might yield more complex results. Finally, as stated in Chapter 4, it was taking too much time to get the district-level approval needed to acquire deidentified student artifacts. Examination of teacher assessment data and student computer coding artifacts would inform the assessment of abstraction. However, the lack of student deidentified data (only from four teachers) limited a deeper examination of the assessment of abstraction.

Recommendations for Future Research

This study solely focused on the teaching aspect, the input, of the educational process. Additional studies about student the student experience learning abstraction in CS, grade level abstraction abilities, and student curricular interests are needed to more fully understand the output, or the learning aspect of the educational process. Additional research investigating effective instructional approaches to teaching abstraction in the elementary grades would inform an aligned and accurate curricular progression of abstraction skills. More investigation into grade-level appropriate abstraction skills and concepts would aide teachers in creating objectives and outcomes. Research that tests refined abstraction rubrics and assessments would help teachers with needed resources. It would also be interesting to offer a survey to a larger teacher population and inquire about the variety of thinking skills (pattern recognition, decomposition, generalization, induction, deduction, and logical thinking) and abstraction. Potential variables for future quantitative study of the instruction of abstraction include programming languages, the relationship between hardware and software, concepts, skills, direct instruction, contextual instruction, teaching experience, student math experience, and STEM curriculum.

A more thorough investigation of a succinct definition of abstraction that K-12 teachers can understand and apply in the classroom would be helpful. The Nand2Tetris course seemed to provide a low-cost simulation for teaching levels of abstraction, which might warrant future investigation. It would be interesting to see the effect of experience applying abstraction for teachers with self-efficacy teaching abstraction. Curriculum to

teach abstraction including graphic organizers and scaffolded lesson plans would help teachers of all grade levels. Teaching cybersecurity and levels of abstraction might provide a context for learning abstraction that would be beneficial. Finally, developing and researching project-based lesson plans or modules supporting AP curricula that helped students to learn abstraction would also be helpful.

Implications for Computer Science Instruction

Positive Social Change

In the four years that I have been working on this dissertation, K-12 CS education has garnered a great deal of national attention and funding. Thirty-seven states have either adopted or are in the process of adopting K-12 CS standards that include computational thinking (Code.org, 2018). Computer science professionals are in high demand - the majority of STEM jobs in marketplace (National Academies of Sciences, Engineering, and Medicine, 2018). Many states and countries are developing CS legislation, policy, curriculum, graduation requirements, and teacher professional development (Rees et al., 2016). In September 2017, President Trump signed a memorandum on increasing access to high-quality STEM and CS education. In 2018, the US Department of Education offered $195 million in grant funds for STEM/CS education, the Support for Effective Educator Development (SEED), and the Education Innovation and Research (EIR) grants. The Perkins Career and Technical Education for the 21st Century Act was reauthorized in July 2018 providing dual coding for both academic and CTE CS courses as well as increasing CS teacher pathways. Perhaps a

more compelling reason than economics for including computing in K-12 education is the argument that computer scientists are the architects of our virtual world.

The more computer scientists understand all the levels and aspects of abstraction, the more efficient and effective our virtual world will work. Computer code that utilizes optimal abstraction uses less energy and is called green code (Hasan et al., 2016). Teachers who study and teach abstraction will understand more of the complexity of CS and become better CS teachers. As Colburn (2000) stated, computers are essentially abstractions of human thought, expanding our content and capability. Teachers who understand and teach that computers are our creations and expressions, will be able help students make ethical decisions and create computational solutions to aide humanity.

Curricular Implications

One teacher interviewed pointed out that the newness of CS for both teachers and students was a challenge and an asset. Obviously, a new content area can be confusing and include a large amount of information to learn. New content areas can also be exciting, especially for high school students who have had many years of Math, Science, Social Studies, and English. Multiple studies showed that connecting CS with content areas, such as Writing, Science, and English as a Foreign Language facilitates computational thinking (Alsamani & Daif-Allah, 2015; Chang, 2014; Kafai & Burke, 2013; Merricks & Henderson, 2013). Perhaps, more of an effort needs to be made to cross-walk CS standards with all content areas in all grade levels, truly adding CS as a fourth foundational literacy. Providing all teachers with cross-curricular connections may assist students in learning difficult topics like abstraction in CS. Promoting the inclusion

of CS in all content areas may also help girls and underrepresented minority student populations in participating in computing.

Conclusion

Abstraction is a multi-faceted concept that the majority of the 12 teachers interviewed in this study admittedly did not fully understand and did not feel comfortable teaching. Current K-12 CS professional development appears to lack essential training for teachers regarding computational thinking of which abstraction is a subskill. Overall, teachers in this study reported addressing directly or indirectly the concept of abstraction. The most experienced teachers shared that introducing the concept of abstraction, building programming skills, and referring back to abstraction as students applied their programming skills contextually facilitated knowledge and abstraction skills. Teachers also reported that dialogue was an essential aspect in teaching abstraction. Overall, the teachers interviewed shared that they would benefit from summative tests, quizzes, and rubrics designed to assess abstraction. Better training and a better definition of abstraction would make their instruction easier and more effective. The analysis of teacher interviews in this study revealed several variables for future quantitative study including programming languages, the relationship between hardware and software, concepts, skills, direct instruction, contextual instruction, teaching experience, student math experience, and STEM curricula. As research informs CS education, the study of abstraction will help teachers and students manage the complexity of computing.

References

Abelson, H., Ledeen, K., & Lewis, H. R. (2008). *Blown to bits: Your life, liberty, and happiness after the digital explosion*. Upper Saddle River, NJ: Addison-Wesley.

Adair, D., & Jaeger, M. (2016). Incorporating Critical Thinking into an Engineering Undergraduate Learning Environment. *International Journal of Higher Education*, *5*(2), 23.

Alsamani, A. A. S., & Daif-Allah, A. S. (2015). Introducing Project-based Instruction in the Saudi ESP Classroom: A Study in Qassim University. *English Language Teaching*, *9*(1), 51.

Andreou, C., Papastavrou, E., & Merkouris, A. (2014). Learning styles and critical thinking relationship in baccalaureate nursing education: a systematic review. *Nurse education today*, *34*(3), 362-371.

Anton, G., & Barany, A. (2013). Power of play: Exploring computational thinking through game design. *Velvet Light Trap*, *72*(1), 74-75.

Apple (nd). (2019). Teacher resources. Retrieved from https://www.apple.com/education/apple-teacher/

Armoni, M. (2013). On teaching abstraction in Computer Science to novices. *Journal of Computers in Mathematics and Science Teaching. (32)* 265-284.

Arnoux, P., & Finkel, A. (2010). Using mental imagery processes for teaching and research in mathematics and computer science. *International Journal of*

Mathematical Education in Science & Technology, *41*(2), 229–242. https://doi.org/10.1080/00207390903372429

Aksu, G. & Koruklu, N. (2015). Determination the effects of vocational high school students' logical and critical thinking skills on mathematic success. *Eurasian Journal of Educational Research*, 59, 181-206 http://dx.doi.org/10.14689/ejer.2015.59.11

Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C., & Norman, M. K. (2010). *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons.

Atabaki, A. M. S., Keshtiaray, N., & Yarmohammadian, M. H. (2015). Scrutiny of Critical Thinking Concept. *International Education Studies*, *8*(3), 93.

Baloukas, T. (2012). JAVENGA: JAva-based Visualization Environment for Network and Graph Algorithms. *Computer Applications in Engineering Education*, *20*(2), 255–268. https://doi.org/10.1002/cae.20392

Baxter, P., & Jack, S. (2008). Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers. *The Qualitative Report*, 13(4), 544-559. Retrieved from http://nsuworks.nova.edu/tqr/vol13/iss4/2

Bell, T., Andreae, P., & Robins, A. (2014). A case study of the introduction of computer science in NZ schools. *ACM Transactions on Computing Education (TOCE)*, *14*(2), 10.

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, *20*(1), 45-74.

Bers, M. U. (2010). The TangibleK Robotics Program: Applied Computational Thinking for Young Children. *Early Childhood Research & Practice*, *12*(2).

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157. doi:10.1016/j.compedu.2013.10.020

Biggs, J. B., & Collis, K. F. (2014). *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press.

Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). *Taxonomy of educational objectives, handbook I: The cognitive domain* (Vol. 19, p. 56). New York: David McKay Co Inc.

Braithwaite, D. W., Goldstone, R. L., van der Maas, H. L., & Landy, D. H. (2016). Nonformal mechanisms in mathematical cognitive development: The case of arithmetic. *Cognition*, *149*, 40-55.

Brenan, K., Resnick, M. (AERA, 2012). *New frameworks for evaluating and discussing the development of computational thinking.* White paper. MIT Medial Lab.

Brookshear, J. (2012). *Computer science and overview.* Boston, MA: Addison-Wesley.

Brown, E., & Jacobsen, M. (2017). Developing Technological Fluency in and through Teacher Education: An Applied Research Project in Teachers' College. In *Teacher Education for Ethical Professional Practice in the 21st Century* (pp. 1-24). IGI Global.

Bucher, T. (2016). 'Machines don't have instincts': Articulating the computational in journalism. *new media & society*, 1461444815624182.

Buckley, J., Archibald, T., Hargraves, M., & Trochim, W. M. (2015). Defining and teaching evaluative thinking: Insights from research on critical thinking. *American Journal of Evaluation*, *36*(3), 375-388.

Cajkler, W., Wood, P., Norton, J., Pedder, D., & Xu, H. (2015). Teacher perspectives about lesson study in secondary school departments: a collaborative vehicle for professional learning and practice development. *Research Papers in Education*, *30*(2), 192-213.

Cappetta, R. W., & Zollman, A. (2013). Agents of Change in Promoting Reflective Abstraction: A Quasi-Experimental, Study on Limits in College Calculus. *Journal of Research in Mathematics Education*, *2*(3), 343-357.

Carbonaro, M., Szafron, D., Cutumisu, M., & Schaeffer, J. (2010). Computer-Game Construction: A Gender-Neutral Attractor to Computing Science. *Computers & Education*, *55*(3), 1098–1111.

Cargas, S. (2016). Honoring controversy: Using real-world problems to teach critical thinking in honors courses. *Honors in Practice, 12(*123-137).

Chang, C.K. (2014). Effects of Using Alice and Scratch in an Introductory Programming Course for Corrective Instruction. *Journal of Educational Computing Research*, *51*(2), 185–204. https://doi.org/10.2190/EC.51.2.c

Charmaz, K. (2014). *Constructing grounded theory*. Los Angeles, CA: Sage.

Chesimet, M. C., Githua, B. N., & Ng'eno, J. K. (2016). Effects of Experiential Learning Approach on Students' Mathematical Creativity among Secondary School Students of Kericho East Sub-County, Kenya. *Journal of Education and Practice*, *7*(23), 51-57.

Cho, J. Y., & Lee, E. (2014). Reducing Confusion about Grounded Theory and Qualitative Content Analysis: Similarities and Differences. e Qualitative Report, 19(32), 1-20. Retrieved from http://nsuworks.nova.edu/tqr/vol19/iss32/2

Chomsky, N. (2006). *Language and mind*. Cambridge, UK: Cambridge University Press.

Cioffi-Revilla, C. (2014). Computation and Social Science. In *Introduction to Computational Social Science* (pp. 23-66). London, UK: Springer.

Cleary, M., Horsfall, J., & Hayter, M. (2014). Data collection and sampling in qualitative research: does size matter?. *Journal of advanced nursing*, *70*(3), 473-475.

Code Fellows (nd). (2019). Code Fellows course information. Retrieved from https://www.codefellows.org/

Code.org (nd). (2017). State computer science statistics. Retrieved from https://code.org/statistics

Colburn, T. (2000). *Philosophy and computer science*. Armonk, NY: M.E. Sharpe.

Colburn, T. (2015). *Philosophy and computer science*. Abingdon, UK: Routledge.

Colburn, T., & Shute, G. (2007). Abstraction in computer science. *Minds and Machines*, *17*(2), 169-184.

Cole, D., & Zhou, J. (2014). Diversity and Collegiate Experiences Affecting Self-Perceived Gains in Critical Thinking: Which Works, and Who Benefits? *The Journal of General Education*, *63*(1), 15-34.

College Board (nd). (2016). Computer science. Retrieved from https://apcentral.collegeboard.org/courses/ap-computer-science-principles

Computer Science Teachers Association (nd). (2015). Science education research. Retrieved from http://csta.acm.org/Research/sub/KeyResearch.html

Computer Science Teachers Association (2019, February). Standards. Retrieved from https://www.csteachers.org/page/standards

Connelly, F. M., & Clandinin, D. J. (1988). *Teachers as Curriculum Planners. Narratives of Experience.* New York, NY: Teachers College Press.

Cooper, S., Dann, W., & Pausch, R. (2003). Teaching objects-first in Introductory Computer Science. *SIGSCE Bulletin (35)* 191-195.

Cooper, R., & Heaverlo, C. (2013). Problem Solving And Creativity And Design: What Influence Do They Have On Girls' Interest In STEM Subject Areas? *American Journal of Engineering Education*, *4*(1), 27.

Cooper, S., Pérez, L. C., & Rainey, D. (2010). Education K-12 Computational Learning. *Communications of the ACM*, *53*(11), 27–29. https://doi.org/10.1145/1839676.1839686

Corbin, J, Strauss, A. (2015). *The basics of qualitative research*. Thousand Oaks, CA: Sage Publications.

Costley, J. (2016). The Effects of Instructor Control on Critical Thinking and Social Presence: Variations within Three Online Asynchronous Learning Environments. *Journal of Educators Online*, *13*(1), 109-171.

Creswell, J. W. (2003). *Research design: Qualitative, quantitative, and mixed methods approaches* (2nd ed.). Thousand Oaks, CA: Sage Publications.

Creswell, J. (2007). *Qualitative inquiry and research design*. Thousand Oaks, CA: Sage Publications.

CS10K (nd). (2015). CS10K initiative to train 10,000 computer science educators. Retrieved from https://cs10kcommunity.org/

Csernoch, M., Biró, P., Máth, J., & Abari, K. (2015). Testing algorithmic skills in traditional and nontraditional programming environments. *Informatics in Education*, *14*(2), 175.

Cuny, J., & Aspray, W. (2002). Recruitment and retention of women graduate students in computer science and engineering: results of a workshop organized by the computing research association. *ACM SIGCSE Bulletin*, *34*(2), 168-174.

Cuny, J. (2017). Computer science for everyone: A groundswell of support [Infosys blog]. Retrieved from http://www.infosys.org/infosys-foundation-usa/media/blog/Pages/groundswell-support.aspx

Czerkawski, B. C., & Lyman III, E. W. (2015). Exploring issues about computational thinking in higher education. *TechTrends*, *59*(2), 57-65.

Dale, N., & Walker, H. M. (1996). Abstract data types—specifications, implementations,

and applications. Lexington: D.C. Heath and Company.

Daily, S. B., & Eugene, W. (2013). Preparing the Future STEM Workforce for Diverse Environments. *Urban Education*, *48*(5), 682–704. https://doi.org/10.1177/0042085913490554

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.

Denning, P. J., Tedre, M., & Yongpradit, P. (2017). Misconceptions about computer science. *Communications of the ACM*, *60*(3), 31-33.

Deruy, E. (2017). In Finland kids learn computer science without computers. *The Atlantic Website.* Retrieved from: https://www.theatlantic.com/education/archive/2017/02/teaching-computer-science-without-computers/517548/

Dubinsky, E. (2002). Reflective abstraction in advanced mathematical thinking. In *Advanced mathematical thinking* (pp. 95-126). Springer Netherlands.]p0w2

Dwyer, C. P., Hogan, M. J., Harney, O. M., & O'Reilly, J. (2014). Using interactive management to facilitate a student-centred conceptualisation of critical thinking: a case study. *Educational Technology Research and Development*, *62*(6), 687-709.

Emir, S. (2013). Contributions of Teachers' Thinking Styles to Critical Thinking Dispositions (Istanbul-Fatih Sample). *Educational Sciences: Theory and Practice*, *13*(1), 337-347.

Ernst, J. V., & Clark, A. C. (2012). Fundamental computer science conceptual understandings for high school students using original computer game design. *Journal of STEM Education: Innovations & Research*, *13*(5), 40–45.

Facione, P. A. (1990). Critical Thinking: A Statement of Expert Consensus for Purposes of Educational Assessment and Instruction. Research Findings and Recommendations.

Facione, P., & Gittens, C. A. (2015). *Think critically*. Upper Saddle River, NJ: Pearson.

Farrell, T. S. (2015). *Reflective language teaching: From research to practice*. New York City, NY: Bloomsbury Publishing.

Fayer, S., Lacey, A., & Watson, A. (2017). BLS Spotlight on Statistics: STEM Occupations-Past, Present, and Future. Retrieved from: https://digitalcommons.ilr.cornell.edu/key_workplace/1923/

Fereday, J., & Muir-Cochrane, E. (2006). Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International journal of qualitative methods*, *5*(1), 80-92.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87-97.

Festo, K. (2016). Question Classification Taxonomies as Guides to Formulating Questions for Use in Chemistry Classrooms. *European Journal of Science and Mathematics Education*, *4*(3), 353-364.

Flavell, J. H., Green, F. L., Flavell, E. R., & Grossman, J. B. (1997). The development of children's knowledge about inner speech. *Child Development*, *68*(1), 39-47.

Flick, U. (2014). *An introduction to qualitative research*. New Dehli, India: Sage.

Floridi, L. (2008). Artificial intelligence's new frontier: Artificial companions and the fourth revolution. *Metaphilosophy*, *39*(4-5), 651-655.

Floridi, L. (2011). *The philosophy of information*. Oxford, United Kingdom: Oxford University Press.

Fouh, E., Akbar, M., & Shaffer, C. A. (2012). The role of visualization in computer science education. *Computers in the Schools*, *29*(1-2), 95–117.

Fuller, U., Johnson, C., Ahoniemi, T. et al (2007). Developing a computer science specific learning taxonomy. ITiCSE working group report on innovation and technology in computer science education. doi: 10.1145/1345443.1345438

Fusch, P. I., & Ness, L. R. (2015). Are we there yet? Data saturation in qualitative research. *The Qualitative Report*, *20*(9), 1408.

Fyfe, E. R., McNeil, N. M., & Rittle-Johnson, B. (2015). Easy as ABCABC: Abstract language facilitates performance on a concrete patterning task. *Child development*, *86*(3), 927-935.

General Assembly (n.d.) (2019). Courses. Retrieved from https://generalassemb.ly

Giannakos, M. N., Koilias, C., Vlamos, P., & Doukakis, S. (2013). Measuring Students' Acceptance and Confidence in Algorithms and Programming: The Impact of

Engagement with CS on Greek Secondary Education. *Informatics in Education-An International Journal*, (Vol12_2), 207-219.

Gibbs, G. R., (2010). *Coding part 2: Thematic coding*. [Web Video]. Retrieved from http://www.youtube.com/watch?v=B_YXR9kp1_o

Gobbo, F., & Benini, M. (2014). The minimal levels of abstraction in the history of modern computing. *Philosophy & Technology*, *27*(3), 327-343.

Goode, J., Margolis, J., & Chapman, G. (2014, March). Curriculum is not enough: The educational theory and research foundation of the exploring computer science professional development model. In *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 493-498). ACM.

Google for Education (n.d.). (2019). Teacher resources. Retrieved from https://edu.google.com/computer-science/?modal_active=none

Grout, V., & Houlden, N. (2014). Taking Computer Science and Programming into Schools: The Glyndŵr/BCS Turing Project. *Procedia - Social and Behavioral Sciences*, *141*, 680–685. https://doi.org/10.1016/j.sbspro.2014.05.119

Groome, M., & Rodríguez, L. M. (2014). How to Build a Robot: Collaborating to Strengthen STEM Programming in a Citywide System. *Afterschool Matters*, *19*, 1-9.

Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*, *8*(6), 1-165.

Hakverdi-Can, M., & Thomas, M. D. (2012). Exemplary science teachers' use of technology. *TOJET: The Turkish Online Journal of Educational Technology*, *11*(1).

Harlow, D. B., & Leak, A. E. (2014). Mapping students' ideas to understand learning in a collaborative programming environment. *Computer Science Education*, *24*(2/3), 229–247. doi:10.1080/08993408.2014.963360

Hasan, S., King, Z., Hafiz, M., Sayagh, M., Adams, B., & Hindle, A. (2016). Energy profiles of java collections classes. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 225-236). ACM.

Hazzan, D. (1999). Reducing abstraction level when learning abstract algebra concepts. *Educational Studies in Mathematics (40)* 71-90. Netherlands: Kluwer Academic Publishers.

Hazzan, O., Lapidot, T., & Ragonis, N. (2015). *Guide to teaching computer science: An activity-based approach*. London, UK: Springer.

Haynes, A., Lisic, E., Goltz, M., Stein, B., & Harris, K. (2016). Moving Beyond Assessment to Improving Students' Critical Thinking Skills: A Model for Implementing Change. *Journal of the Scholarship of Teaching and Learning*, *16*(4), 44-61.

Hsu, Y. C., & Ching, Y. H. (2013). Mobile app design for teaching and learning: Educators' experiences in an online graduate course. *The International Review of Research in Open and Distributed Learning*, *14*(4).

Hu, H. H., Kussmaul, C., Knaeble, B., Mayfield, C., & Yadav, A. (2016, July). Results from a survey of faculty adoption of Process Oriented Guided Inquiry Learning (POGIL) in Computer Science. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*(pp. 186-191). ACM.

Huang, H. F., Ricci, F. A., & Mnatsakanian, M. (2016). Mathematical teaching strategies: Pathways to critical thinking and metacognition. *International Journal of Research in Education and Science*, *2*(1), 190-200.

Ismail, M. N., Ngah, N. A., & Umar, I. N. (2010). Instructional strategy in the teaching of computer programming: A need assessment analyses. *Turkish Online Journal of Educational Technology*, *9*(2), 125–131.

Israel, M., Wherfel, Q. M., Pearson, J., Shehab, S., & Tapia, T. (2015). Empowering K–12 Students with Disabilities to Learn Computational Thinking and Computer Programming. *TEACHING Exceptional Children*, *48*(1), 45-53.

ISTE. (2019, February). International Society for Technology in Education: computer science standards. Retrieved from: https://www.iste.org/standards/for-computer-science-educators

Jacobsen, T. E., & Mackey, T. P. (2013). Proposing a metaliteracy model to redefine information literacy. *Communications in information literacy*, *7*(2), 84-91.

Jenkins, C. (2015). Poem Generator: A Comparative Quantitative Evaluation of a Microworlds-Based Learning Approach for Teaching English. *International Journal of Education and Development Using Information and Communication Technology*, *11*(2), 153–167. Retrieved

from http://ezp.waldenulibrary.org/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=eric&AN=EJ1074163&site=ehost-live&scope=site

Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan*, *95*(1), 61-65.

Kafai, Y. B., & Burke, Q. (2015). Computer programming goes back to school. *Education Week*, 61–65. Retrieved from http://www.edweek.org/ew/articles/2013/09/01/kappan_kafai.html?qs=digital+divide

Kalelioglu, F., & Gülbahar, Y. (2014). The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. *Informatics in Education*, *13*(1), 33–50. Retrieved from http://ezp.waldenulibrary.org/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=eric&AN=EJ1064285&site=ehost-live&scope=site

Kaufman, R. (2017). Practical strategic leadership: Aligning human performance development with organizational contribution. *Performance Improvement*, *56*(2), 16-21.

Kazak, S., Wegerif, R., & Fujita, T. (2015). The importance of dialogic processes to conceptual development in mathematics. *Educational Studies in Mathematics*, *90*(2), 105-120.

Kazakoff, E., & Bers, M. (2012). Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. *Journal of Educational Multimedia and Hypermedia*, *21*(4), 371-391.

Kivunja, C. (2014). Do you want your students to be job-ready with 21st century skills? Change pedagogies: a pedagogical paradigm shift from Vygotskyian social constructivism to critical thinking, problem solving and Siemens' digital connectivism. *International Journal of Higher Education*, *3*(3), p81.

Kong, L. N., Qin, B., Zhou, Y. Q., Mou, S. Y., & Gao, H. M. (2014). The effectiveness of problem-based learning on development of nursing students' critical thinking: A systematic review and meta-analysis. *International Journal of Nursing Studies*, *51*(3), 458-469.

Kramer, J. (2007). Is abstraction the key to computer coding? *Communications of the Association for Computing Machinery (50)*.

Kwan, Y. W., & Wong, A. F. (2015). Effects of the constructivist learning environment on students' critical thinking ability: Cognitive and motivational variables as mediators. *International Journal of Educational Research*, *70*, 68-79.

Lan, Y.F., & Lin, P.C. (2011). Evaluation and improvement of student's question-posing ability in a web-based learning environment. *Australasian Journal of Educational Technology*, *27*(4), 581–599.

Lau, W. (2018). *Teaching computing in secondary schools*. New York, NY: Routledge.

Lee, Y. J. (2010). Developing computer programming concepts and skills via technology-enriched language-art projects: A case study. *Journal of Educational Multimedia and Hypermedia*, *19*(3), 307-326.

Lee, T. Y., Mauriello, M. L., Ahn, J., & Bederson, B. B. (2014). CTArcade: Computational thinking with games in school age children. *International Journal of Child-Computer Interaction, 2*, 26–33. doi:10.1016/j.ijcci.2014.06.003

Lewins, A., & Silver, C. (2009). Choosing a CAQDAS package. Retrieved from: http://eprints.ncrm.ac.uk/791/1/2009ChoosingaCAQDASPackage.pdf

Lihui, W. H., Qun, Z., Feng, L., & Qin Yuqing, W. (2015). Teacher Questioning in College English Class: A Guide to Critical Thinking. *Global Journal of Human-Social Science Research*, *15*(11).

Liu, O. L., Frankel, L., & Roohr, K. C. (2014). Assessing Critical Thinking in Higher Education: Current State and Directions for Next-Generation Assessment. *ETS Research Report Series*, *2014*(1), 1-23.

Lim, B., Hosack, B., & Vogt, P. (2012). A framework for measuring student learning gains and engagement in an introductory computing course: A preliminary report of findings. *Electronic Journal of e-Learning*, *10*(4), 428–440.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61. doi:10.1016/j.chb.2014.09.012

Madsen, C. K., & Geringer, J. M. (2014). The Relationship between Teacher Preparation and Long-Term Teaching Effectiveness. *ISME Commission on Research*, 229.

Mahn, H. (2012). Vygotsky's analysis of children's meaning making processes. *International Journal of Educational Psychology*, *1*(2), 100-126.

Malatji, K. S. (2016). Moving away from Rote Learning in the University Classroom: The Use of Cooperative Learning to Maximise Students' Critical Thinking in a Rural University of South Africa. *Journal of Communication*, *7*, 34-42.

Martinez, M. C., Gomez, M. J., Moresi, M., & Benotti, L. (2016, July). Lessons learned on computer science teachers professional development. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 77-82). ACM.

Marzano, R. J., & Kendall, J. S. (Eds.). (2006). *The new taxonomy of educational objectives*. Thousand Oaks, CA: Corwin Press.

Maxwell, J. A. (2013). Applied social research methods series: Vol. 41. *Qualitative research design: An interactive approach*, *3*.

Mayes, R., & Koballa Jr, T. R. (2012). Exploring the science framework. *The Science Teacher*, *79*(9), 27.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239-264.

Merriam, S., & Tisdell, E. (2016). *Qualitative research: A guide to design and implementation.* San Francisco, CA: Jossey-Bass.

Merricks, J., & Henderson, J. (2014). From Vibration to Vocalization. *Science and Children*, *51*(6), 44–49. Retrieved from http://ezp.waldenulibrary.org/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=eric&AN=EJ1035542&site=ehost-live&scope=site

Microsoft (n.d.). (2019). Teacher resources. Retrieved from https://www.microsoft.com/en-us/education/educators/2018/stem-computer-science/default.aspx

Miles, M. B., & Huberman, A. M., Saldana, J. (2014). *Qualitative data analysis*. Newbury Park, CA: Sage

Morris, M. W., Leung, K., Ames, D., & Lickel, B. (1999). Views from inside and outside: Integrating emic and etic insights about culture and justice judgment. *Academy of Management Review, 24*(4), 781-796.

Morse, J. M., Barrett, M., Mayan, M., Olson, K., & Spiers, J. (2002). Verification strategies for establishing reliability and validity in qualitative research. *International journal of qualitative methods*, *1*(2), 13-22.

Mudrikah, A. (2016). Problem-based learning associated by action-process-object-schema (APOS) theory to enhance students' high order mathematical thinking ability. *International Journal of Research in Education and Science, 2*(1), 125-135.

National Academies of Sciences, Engineering, and Medicine. (2018). Assessing and responding to the growth of computer science undergraduate enrollments.

National Center for Education Statistics (ED). (2012). *The nation's report card: Science in action--hands-on and interactive computer tasks from the 2009 Science Assessment. NCES 2012-468*. National Center for Education Statistics.

Norman, L. [LarryNorman]. (2006, September 12). *CSPI / What is "Computer Science?"* [Video file]. Retrieved from https://www.youtube.com/watch?v=zQLUPjefuWA

Novack, M. A., Congdon, E. L., Hemani-Lopez, N., & Goldin-Meadow, S. (2014). From action to abstraction: Using the hands to learn math. *Psychological Science*, *25*(4), 903-910.

Olivia (n.d.). What is confirmability in qualitative research and how do we establish it? [Web log comment]. Retrieved from http://www.statisticssolutions.com/what-is-confirmability-in-qualitative-research-and-how-do-we-establish-it/

Özyurt, Ö. (2015). Examining the Critical Thinking Dispositions and the Problem Solving Skills of Computer Engineering Students. *EURASIA Journal of Mathematics, Science & Technology Education*, *11*(2), 353–361.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Harper Collins.

Patton, M. (2002). *Qualitative research and evaluation methods*. Thousand Oaks, CA: Sage Publications.

Pellas, N., & Peroutseas, E. (2016). Gaming in Second Life via Scratch4SL Engaging High School Students in Programming Courses. *Journal of Educational Computing Research*, *54*(1), 108–143. https://doi.org/10.1177/0735633115612785

Perrenet, J. (2010). Levels of thinking in computer science. Development in bachelor students' conceptualization of algorithm. Education & Information Technologies (15) 87-107. doi: 10.1007/s10639-009-9098-8

Perrenet, J.C., J.F. Groote & E. Kaasenbrood (2005). Exploring Students' Understanding of the Concept of Algorithm: Levels of Abstraction; In: Proceedings of the 10th

annual SIGCSE-conference on Innovation and technology in computer science education, 64–68; Caparica, Portugal. © ACM 1-59593-024-8/05/0006. Retrieved from http://acm.org/10.1145/1070000/1067467

Perrenet, J.C. & E. Kaasenbrood (2006). Levels of Abstraction in Students' Understanding of the Concept of Algorithm: the Qualitative Perspective; In: Proceedings of the 11th annual SIGCSE-conference on Innovation and technology in computer science education, 270–275; Bologna, Italy. © ACM 1-59593-055-8/06/0006. Retrieved from http://acm.org/10.1145/1150000/1140196

Piaget, J. (1950). *The psychology of intelligence*. London, UK: Routledge.

Piaget, J. (2014). *Studies in reflecting abstraction*. New York, NY: Psychology Press.

Porter, L., Bailey Lee, C., & Simon, B. (2013, March). Halving fail rates using peer instruction: a study of four computer science courses. In *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 177-182). ACM.

Promraksa, S., Sangaroon, K., & Inprasitha, M. (2014). Characteristics of Computational Thinking about the Estimation of the Students in Mathematics Classroom Applying Lesson Study and Open Approach. *Journal of Education and Learning*, *3*(3), 56.

Przybylla, M., & Romeike, R. (2014). Physical Computing and Its Scope--Towards a Constructionist Computer Science Curriculum with Physical Computing. *Informatics in Education*, *13*(2), 241–254.

Qualitative Validity, (n.d.). In *Web Center for Social Research Methods.* Retrieved from https://socialresearchmethods.net/kb/qualval.php

Ralston, P. A., & Bays, C. L. (2015). Critical thinking development in undergraduate engineering students from freshman through senior year: a 3-cohort longitudinal study. *American Journal of Engineering Education*, *6*(2), 85.

Rees, A., García-Peñalvo, F. J., Jormanainen, I., Tuul, M., & Reimann, D. (2016). An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers.

Repenning, A., Webb, D. C., Koh, K. H., Nickerson, H., Miller, S. B., Brand, C., … others. (2015). Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation. *ACM Transactions on Computing Education (TOCE)*, *15*(2), 11. Retrieved from http://dl.acm.org/citation.cfm?id=2700517

Rittle-Johnson, B., & Schneider, M. (2014). Developing conceptual and procedural knowledge of mathematics. *Oxford handbook of numerical cognition*, 1102-1118.

Rowles, J., Morgan, C. M., Burns, S., & Merchant, C. (2013). Faculty perceptions of critical thinking at a health sciences university. *Journal of the Scholarship of Teaching and Learning*, *13*(4), 21-35.

Rubin, H., Rubin, I. (2005). *Qualitative interviewing: The art of hearing data.* Thousand Oaks, CA: Sage Publications.

Ruecker, S., Grotkowski, A., Gabriele, S., Roberts-Smith, J., Sinclair, S., Dobson, T., … Rodriguez, O. (2013). Abstraction and realism in the design of avatars for the simulated environment for theatre. *Visual Communication*, *12*(4), 459–472.

Ryoo, J.J., Margolis, J., Goode, J., Lee, C., Moreno Sandoval, C.D. (2014). *ECS teacher practices research findings—In brief.* Los Angeles, CA: Exploring Computer Science Project, University of California, Los Angeles Center X with University of Oregon, Eugene. Retrieved from http://www.exploringcs.org/ecs-teacher-practices-research.

Saldaña, J. (2015). *The coding manual for qualitative researchers*. Sage.

Sánchez, P., Zorrilla, M., Duque, R., & Nieto-Reyes, A. (2011). Are models easier to understand than code? An empirical study on comprehension of entity-relationship (ER) models vs. structured query language (SQL) code. *Computer Science Education*, *21*(4), 343–362. https://doi.org/10.1080/08993408.2011.630128

Sanford, J. F., & Naidu, J. T. (2016). Computational Thinking Concepts for Grade School. *Contemporary Issues in Education Research*, *9*(1), 23–32.

Saeli, M., Perrenet, J., Jochems, W. M. G., & Zwaneveld, B. (2012). Programming: Teachers and pedagogical content knowledge in the Netherlands. *Programavimas: Mokytojai Ir Pedagoginio Turinio Žinios Nyderlanduose.*, *11*(1), 81–114.

Sanz, C. (2005). *Mind and context in adult second language acquisition: Methods, theory, and practice*. Georgetown University Press.

Shannon, Claude E. ; Weaver, Warren & Burks, Arthur W. (1951). The Mathematical Theory of Communication (review). *Philosophical Review* 60 (3):398-400.

Shehane, R., & Sherman, S. (2014). Visual Teaching Model for Introducing Programming Languages. *Journal of Instructional Pedagogies*, *14*.

Shirazi, A. S., von Mammen, S., & Jacob, C. (2013). Abstraction of agent interaction processes: Towards large-scale multi-agent models. *Simulation*, *89*(4), 524-538.

Simon, M. A., Kara, M., Placa, N., & Sandir, H. (2016). Categorizing and promoting reversibility of mathematical concepts. *Educational Studies in Mathematics*, *93*(2), 137-153.

Shell, D. F., & Soh, L.-K. (2013). Profiles of Motivated Self-Regulation in College Computer Science Courses: Differences in Major versus Required NonMajor Courses. *Journal of Science Education and Technology*, *22*(6), 899–913. Retrieved from http://ezp.waldenulibrary.org/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=eric&AN=EJ1038476&site=ehost-live&scope=site

Snyder, J. J., & Wiles, J. R. (2015). Peer led team learning in introductory biology: Effects on peer leader critical thinking skills. *PloS one*, *10*(1), e0115084.

Sho-Huan Tung, Tsung-Te Lin, & Yen-Hung Lin. (2013). An Exercise Management System for Teaching Programming. *Journal of Software (1796217X)*, *8*(7), 1718–1725. https://doi.org/10.4304/jsw.8.7.1718-1725

Stake, B. (2006). *Multiple case study analysis*. New York, NY: Guilford Press.

Sullivan, A., Kazakoff, E. R., & Bers, M. U. (2013). The wheels on the bot go round and round: Robotics curriculum in pre-kindergarten. *Journal of Information Technology Education*, *12*, 203-219.

Szűcs, D., Devine, A., Soltesz, F., Nobes, A., & Gabriel, F. (2014). Cognitive components of a mathematical processing network in 9-year-old children. *Developmental Science*, *17*(4), 506-524.

Teague, D. (2015). *Neo-Piagetian Theory and the novice programmer* (Doctoral dissertation, Queensland University of Technology).

Tóth, L., Adorjani, A. K., & Katai Z. (2014). Multi-Sensory Informatics Education. *Informatics in Education-An International Journal*, (Vol13_2), 225-240.

Turner, R. (2013). The philosophy of computer science.

Tung, S. H., Lin, T. T., & Lin, Y. H. (2013). An exercise management system for teaching programming. *Journal of Software*, *8*(7), 1718-1725.

Uysal, M. P. (2016). Evaluation of learning environments for object-oriented programming: measuring cognitive load with a novel measurement technique. *Interactive Learning Environments*, *24*(7), 1590-1609.

Vaismoradi, M., Turunen, H., & Bondas, T. (2013). Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study. *Nursing & health sciences*, *15*(3), 398-405.

Vakil, S. (2014). A critical pedagogy approach for engaging urban youth in mobile app development in an after-school program. *Equity & Excellence in Education*, *47*(1), 31-45.

Vanicheva, T., Kah, M., & Ponidelko, L. (2015). Critical thinking within the current framework of ESP curriculum in technical universities of Russia. *Procedia-Social and Behavioral Sciences*, *199*, 657-665.

Vygotsky, L. (1978). *Mind in society: The development of higher psychological processes.* Cambridge, MA: Harvard University Press.

Vygotsky, L. (1986). *Thought and language*. Cambridge, MA: MIT Press.

Wang, D., Wang, T., & Liu, Z. (2014). A tangible programming tool for children to cultivate computational thinking. *Thescientificworldjournal*, *2014*, 428080–428080. doi:10.1155/2014/428080

Waite, J., Curzon, P., Marsh, W. & Sentence, S. (2016, October). Abstraction and common classroom activities. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 112-113). ACM.

Weintrop, D., & Wilensky, U. (2014). Situating programming abstractions in a constructionist video game. *Informatics in Education*, *13*(2), 307.

Whistler, D. (2016). Abstraction and utopia in early German idealism. *LOGOS*, (2), 5-27.

White, P., Mitchelmore, M. (2010). Teaching for abstraction: A model. *Mathematical Thinking and Learning (12)* 205-226.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, *366*(1881), 3717-3725.

Youssef, B. B., & Berry, B. (2012). Learning to think spatially in an undergraduate interdisciplinary computational design context: a case study. *International Journal of Technology and Design Education*, *22*(4), 541-564.

Yildiz, M. & Scharaldi, K. (2015). Introduction to Engineering and Computer Science in Teacher Education: Hour of Code Project. In D. Rutledge & D. Slykhuis (Eds.), *Proceedings of Society for Information Technology & Teacher Education International Conference 2015* (pp. 857-865). Chesapeake, VA: Association for the Advancement of Computing in Education (AACE). Retrieved March 18, 2017 from https://www.learntechlib.org/p/150102.

Yin, R. K. (2015). *Qualitative research from start to finish*. Guilford Publications.

Zendler, A., & Klaudt, D. (2012). Central Computer Science concepts to research-based teacher training in Computer Science: An experimental study. *Journal of Educational Computing Research*, *46*(2), 153–172.

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research*, *53*(4), 562-590.

Appendix A: First Interview Base Questions

The following questions will be asked in all interviews along with follow-up questions specific to each interview.

1. What grade(s) do you teach?
2. How long have you been teaching?
3. What types of computer science classes do you teach?
4. How did you become a computer science teacher?
5. How familiar are you with abstraction in computer science?
6. To what degree do you include abstraction in your course objectives?
7. How capable are your students of using abstraction in their computer coding?
8. How do you know when your students are using abstraction?
9. How comfortable are you teaching abstraction?
10. How often do your instructional activities teach students about abstraction?
11. To what degree do you include abstraction in your course objectives?
12. What kind of professional development, if any, has informed your instruction of abstraction?
13. How confident do you feel about creating and using assessments that measure abstraction?
14. Would you describe abstraction as a skill or a concept, and why?
15. How important do you think abstraction is as a skill in computer science?

Appendix B: Alignment of Research and Interview Questions

Table 1
Alignment of research questions with interview questions

Interview Questions	RQ1	RQ2	RQ3
What grade(s) do you teach?	X	X	
How long have you been teaching?		X	
What types of computer science classes do you teach?	X	X	
How did you become a computer science teacher?		X	
How familiar are you with abstraction in computer science?	X	X	X
To what degree do you include abstraction in your course objectives?			X
How capable are your students of using abstraction in their computer coding?	X		
How do you know when your students are using abstraction?			X
How comfortable are you teaching abstraction?		X	
How often do your instructional activities teach students about abstraction?			X
To what degree do you include abstraction in your course objectives?			X
What kind of professional development, if any, has informed your instruction of abstraction?		X	
How confident do you feel about creating and using assessments that measure abstraction?			X
Would you describe abstraction as a skill or a concept, and why?		X	X
How important do you think abstraction is as a skill in computer science?	X		X

Appendix C: Second Interview Base Questions

What additional thoughts did you have regarding abstraction in K-5 computer science education?

What are your favorite lesson plans for teaching abstraction?

Do you have any additional thoughts on how you would define abstraction?

How have your students talked about abstraction?

What is easy about teaching abstraction?

What is difficult or challenging about teaching abstraction?

What are your successes with teaching abstraction?

What kind of code tells you that your students are using abstraction skills?

What programming languages do you think are best for teaching abstraction?

What grade do you think abstraction is best introduced?

Do you think any student could demonstrate abstraction?

How would you define abstraction?

What do you think beginning CS teachers should know about teaching abstraction?

Do you think abstraction should be assessed in CS elementary courses? Why? Or why not?

Is it better to teach abstraction with online tutorial curriculum, such as Code.org, or with manipulatives like Microbits and Raspberry pi's?

What type of professional development would you find helpful regarding teaching abstraction?

Appendix D: Email to Participants

Dear ,

I would like to invite you to participate in my dissertation research study. I am seeking computer science teachers with two or more years of experience teaching K-12 computer science, or prior experience as a computer science professional and K-12 computer science teacher. The purpose of this descriptive qualitative inquiry is to examine teachers' experiences determining curriculum, delivering instruction, and designing assessments regarding the topic of abstraction in computer science.

Your participation will require:

1) Two one-hour interviews in person or virtually.
2) Five student artifacts that you determine show evidence of abstraction or show evidence of developing abstraction. You will need to de-identify each of the artifacts before you submit them as a pdf document. If your principal requires parental consent, I will ask you to email the student and their parents to obtain consent for their participation in the research study. Once I have obtained all necessary consent forms, I will email you to schedule interviews and ask you to submit digital copies of the artifacts to me.

It is estimated that about 4 hours of your time is required for this research. The total time of the interviews and data collection will be one month. Your participation is voluntary and you can opt out of the study if you so desire.

Please see the attached research participation checklist. If you would like to participate in this research study, please email me your adult consent form with your principal's signature of assent before or in one week.

Warm regards,

Christine Liebe