

Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

5-2016

Exploring Trends in Middle School Students' Computational Thinking in the Online Scratch Community: a Pilot Study

Kevin N. Lawanto
Utah State University

Follow this and additional works at: <https://digitalcommons.usu.edu/etd>



Part of the [Educational Methods Commons](#)

Recommended Citation

Lawanto, Kevin N., "Exploring Trends in Middle School Students' Computational Thinking in the Online Scratch Community: a Pilot Study" (2016). *All Graduate Theses and Dissertations*. 5072.

<https://digitalcommons.usu.edu/etd/5072>

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.



EXPLORING TRENDS IN MIDDLE SCHOOL STUDENTS' COMPUTATIONAL
THINKING IN THE ONLINE SCRATCH COMMUNITY: A PILOT STUDY

by

Kevin N. Lawanto

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Instructional Technology and Learning Sciences

Approved:

Sheri Haderlie, Ph.D.
Major Professor

Breanne Litts, Ph.D.
Committee Member

Gilberto Urroz, Ph.D.
Committee Member

Mark R. McLellan, Ph.D.
Vice President for Research and
Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2016

Copyright © Kevin N. Lawanto 2016

All Rights Reserved

ABSTRACT

Exploring Trends in Middle School Students' Computational Thinking
in the Online Scratch Community: A Pilot Study

by

Kevin N. Lawanto, Master of Science
Utah State University, 2016

Major Professor: Sheri Haderlie, Ph.D.
Department: Instructional Technology and Learning Sciences

Teaching computational thinking has been a focus of recent efforts to broaden the reach of computer science (CS) education for today's students who live and work in a world that is heavily influenced by computing principles. Computational thinking (CT) essentially means thinking like a computer scientist by using principles and concepts learned in CS as part of our daily lives. Not only is CT essential for the development of computer applications, but it can also be used to support problem solving across all disciplines. Computational thinking involves solving problems by drawing from skills fundamental to CS such as decomposition, pattern recognition, abstraction, and algorithm design.

The present study examined how Dr. Scratch, a CT assessment tool, functions as an assessment for computational thinking. This study compared strengths and weaknesses of the CT skills of 360 seventh- and eighth-grade students who were engaged in a Scratch programming environment through the use of Dr. Scratch. The data were collected from a

publicly available dataset available on the Scratch website. The Mann-Whitney *U* analysis revealed that there were specific similarities and differences between the seventh- and eighth-grade CT skills. The findings also highlight affordances and constraints of Dr. Scratch as a CT tool and address the challenges of analyzing Scratch projects from young Scratch learners. Recommendations are offered to researchers and educators about how they might use Scratch data to help improve students' CT skills.

(79 pages)

PUBLIC ABSTRACT

Exploring Trends in Middle School Students' Computational Thinking in the Online Scratch Community: A Pilot Study

Kevin N. Lawanto

We live in a century in which technology has become part of our lives, and it is crucial that we become active creators and not merely passive users of technology. One characteristic that might distinguish someone who uses the ideas of others from one who innovates his own ideas is the ability to computer program. Computer programming is more than just learning how to code; it also exposes students to computational thinking (CT), which involves problem-solving using computer science (CS) concepts, such as decomposition, pattern recognition, abstraction, and algorithm design.

The rationale for introducing computing in K-12 in order to advance CT is compelling. While currently the need to introduce CT skills is prioritized at the high school level, there is a growing belief among researchers that CS experiences need to start at an earlier age. This study examines the elements of CT found in the projects of 7th- and 8th- grade students. Specifically, I used Dr. Scratch to examine whether there were patterns in the students' computational thinking skills. In order to explore the elements of CT found in the students' Scratch projects, datasets of 360 student projects from a publicly available repository of projects on the Scratch website were analyzed.

The results from the study suggested that there were specific similarities and differences between the seventh- and eighth-grade CT skills. The results also highlighted

affordances and constraints of Dr. Scratch as a CT tool and addressed the challenges of analyzing Scratch projects from young learners. Recommendations are offered to researchers and educators about how they might use Scratch data to help improve students' CT skills.

ACKNOWLEDGMENTS

I would like to thank God for the guidance and blessings He gave me while I was working on my thesis. I also would like to thank all those whose influence, friendship, and mentorship have helped me to perform this research and to be successful at Utah State University. It would be difficult to thank everyone individually; however, I would like to specifically thank a few. I express many thanks to Dr. Sheri Haderlie for her willingness to accept being my new advisor. Also, my appreciation goes to Dr. Breanne Litts for agreeing to be my new committee member and for her invaluable mentorship, guidance, and feedback during the thesis revision process. Another committee member, Dr. Gilberto Urroz, was very helpful and cooperative with me. I truly appreciate the expertise and insights that each of you offered. A special thanks is also dedicated to Dr. Sarah Brasiel who provided invaluable guidance and feedback during my graduate study, especially during the making of this thesis. Although she could no longer be my thesis advisor, she nevertheless showed an interest in the progress of my thesis revisions and cheered me on when I passed the defense.

I am also deeply indebted to my friends during my graduate study at Utah State University, especially Clarence Ames, Kevin Close, Soojeong Jeong, and Fred Poole, who supported me from day one of the program all the way through the end in addition to the valuable time they set aside to help review and provide feedback for this paper. Last, but not least, I thank my parents, who always supported and encouraged me during my graduate study. I would not have succeeded without their help.

Kevin N. Lawanto

CONTENTS

	Page
ABSTRACT.....	iii
PUBLIC ABSTRACT	v
ACKNOWLEDGMENTS	vii
LIST OF TABLES.....	x
LIST OF FIGURES	xi
CHAPTER	
I. INTRODUCTION	1
II. REVIEW OF LITERATURE	5
Computer Science in the Digital Age	5
Computing Education in K-12	7
Computational Thinking	12
Context of Current Study	16
Study Overview	21
III. METHODS	22
Research Design.....	22
Sample.....	23
Measures	24
Procedures.....	26
Data Analysis	27
IV. RESULTS	30
Strengths and Weaknesses in Students' CT.....	30
Affordances and Constraints of Dr. Scratch	36
Developing	36
Master	39
Summary	42

V. DISCUSSION AND CONCLUSIONS	45
Discussion	45
Conclusions.....	50
REFERENCES	51
APPENDICES	59
Appendix A: Summary of Computational Thinking Average Scores for each Component from 2013 to 2014 School Year.....	60
Appendix B: Summary of Computational Thinking Average Scores for each Component from 2014 to 2015 School Year.....	62
Appendix C: Histograms comparing seventh- and eighth-grade CT component scores from 2013 to 2014 (left) and 2014 to 2015 (right) school years	64
Appendix D: IRB Exemption Letter.....	67

LIST OF TABLES

Table	Page
1. Components of Computational Thinking Referenced in the Prior and Present Studies	15
2. Scoring System Measuring User Interactivity	20
3. Rubric for Scoring CT Components	27
4. Mann-Whitney U Scores across CT Components	35
5. CT Average Scores for Seventh- and Eighth-Grade Students in the Developing Category	37
6. Computational Thinking Average Scores for Seventh- and Eighth-Grade Students in the Master Category	40
A1. Summary of Computational Thinking Average Scores for each Component from 2013 to 2014 School Year	61
B1. Summary of Computational Thinking Average Scores for each Component from 2014 to 2015 School Year	63

LIST OF FIGURES

Figure	Page
1. An example of different types of Scratch genre	17
2. Scratch projects genres for seventh- and eighth-grade students across 2013 to 2015 academic years.....	24
3. An example of Dr. Scratch analysis results for a project with a basic CT score	25
4. An example of Dr. Scratch analysis results for a project with an advanced CT score	26
5. Histograms comparing seventh- and eighth-grade abstraction and user interactivity component scores from 2013 to 2014 (left) and 2014 to 2015 (right) school years	31
6. CT skill development for seventh and eighth graders from 2013 to 2014 (top) and 2014 to 2015 (bottom) school years.....	34
7. Example dashboard for one seventh-grade student in the developing category.....	38
8. Example of an incorrect attribute initialization (left) and a correct attribute initialization (right).....	39
9. Example dashboard for one eighth-grade student in the master category	41
10. Example of efficient script duplication.....	41
11. Examples of a “dead code” (left) and correct code initialization (right)	42
12. Example of one eighth-grade student who received a total score of 17 and had Scratch code glitches.....	44
C1. Histograms comparing seventh- and eighth-grade CT component scores from 2013 to 2014 (left) and 2014 to 2015 (right) school years.....	65

CHAPTER I

INTRODUCTION

Teaching computational thinking (CT) has been a focus of recent efforts to broaden the reach of computer science (CS) education. Among many researchers, Barr and Stephenson (2011) pointed out that modern students live and work in a world that is heavily influenced by computing principles. A report by the National Council for Research (2010) introduced a similar idea that CT is a cognitive skill that the average person needs to possess. The report highlighted that

(1) students can learn thinking strategies such as CT as they study a discipline, (2) teachers and curricula can model these strategies for students, and (3) appropriate guidance can enable students to learn to use these strategies independently. (p. 62)

Thus, the term CT has quickly become a prerequisite skill for many endeavors of the 21st century (Wing, 2008). CT is broadly defined as a mental activity for abstracting problems and formulating solutions that can be automated (Pulimood, Pearson, & Bates, 2016; Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014).

The use of CT has received significant attention most recently among educators of K-12 students. Computer Science for All is an initiative supported by President Obama to empower K-12 students to learn CS and to become equipped with the CT skills needed to emerge as active creators and not merely passive users of technology (Smith, 2016).

During President Obama's State of the Union 2016 Keynote Address, he mentioned that the economy is rapidly shifting, and both educators and business leaders are increasingly recognizing CS as a basic skill that is necessary for economic opportunity and social mobility. Learning how to computer program is one effective way to address the need

and teach CT skills, considered by many as one of the most important skills of the 21st century (Smith, 2016; Wing, 2006, 2011). Despite its importance, computer programming courses have not been widely implemented in the U.S. K-12 education system, and educators generally agree that the greatest lack of implementation exists in grades K-8 (Computer Science Teachers Association [CSTA], 2012; Lee, Martin, & Apone, 2014; Mannila et al., 2014). Recently, the CSTA created CS standards for K-12 education; however, only a handful of states have adopted these standards into their school systems (e.g., Massachusetts and Washington; Close, Janisiewicz, Brasiel, & Martin, 2015).

Historically, exposing high school students to CS principles has been prioritized by teachers; however, there is a growing belief that experiences with computer programming must start at an earlier age (Grover, 2014). Research has shown that when students are introduced to science, technology, engineering, and mathematics (STEM) curricula early, it can positively impact their perceptions, encouraging them to continue to develop important STEM skills (Bagiati, Yoon, Evangelou, & Ngambeki, 2010; Bybee & Fuchs, 2006; DeJarnette, 2012). Computer science classes in middle school are particularly important in that they can support the development of CT skills and ultimately influence career choices (Barendsen et al., 2015; Repenning, Webb, & Ioannidou, 2010; Settle et al., 2012).

Wing (2006) introduced the term *computational thinking* (CT) to describe the collection of diverse skills related to problem solving which were determined after studying the nature of computation. Since then, studies have shown that some teachers who use computer programming to teach CT struggle to identify and assess its

components (Grover & Pea, 2013). As a result, the focus in this topic area has shifted to tackling the more practical questions of how to promote and assess the development of CT (Grover & Pea, 2013; Werner, Denner, & Campe, 2015; Werner, Denner, Campe, & Kawamoto, 2012) and how to integrate it into the K-12 system (Lee et al., 2014). To address this problem, many researchers have tried to define CT skills, tools, and techniques that may be used to support students in CS education. More recently, Moreno-León and Robles (2015) listed seven important components of CT that teachers should know and implement when teaching computer programming: abstraction, parallelism, logical thinking, synchronization, flow control, data representation, and user interactivity. The above-referenced CT components have far-reaching implications for solving problems and understanding systems across the school curriculum. By implementing the CT concepts into the classroom, teachers will be able to teach step-by-step approaches to solve problems by first identifying key information in a problem.

Although there is a need to integrate CT concepts into classroom practices, there are barriers that teachers must overcome prior to doing so. One barrier is the lack of tools that support educators and researchers in the assessment of student projects. Recently, several tools have become available that teachers may use to analyze students' Scratch data, but currently there is inadequate evidence to support the effectiveness of the tools (Moreno-León & Robles, 2015). Thus, the present study seeks to contribute to this need by empirically investigating Dr. Scratch as an example of a user-friendly CT assessment tool to analyze students' Scratch projects. Specifically, Dr. Scratch allows researchers and instructors to visually assess students' CT by evaluating components used by students in their Scratch games and highlighting where improvement is needed. The goal

of the study was twofold: (1) use Dr. Scratch to build an understanding of middle school students' strengths and weaknesses in CT, and (2) identify affordances and constraints of how Dr. Scratch functions as a CT assessment tool. An important objective is to inform educators and researchers with methods to intervene and support student learning and design and implement CT assessment tools in CS education classrooms.

CHAPTER II

REVIEW OF LITERATURE

The present study was grounded in three key areas of research. The first area summarized important findings from literature which has studied the place of CS in the digital age, including an exploration of how CS and CT are related. The second area highlighted research on CT teaching in K-12 classrooms: what is known and what remains to be learned. The third area studied existing CT assessment tools for K-12 classrooms.

Computer Science in the Digital Age

The most commonly cited rationale for including CS in K-12 instruction is the growing demand for CS skills in the workplace (CSTA, 2003, 2010, 2012; Israel, Pearson, Tapia, Wherfel, & Reese, 2015; Wilson & Moffat, 2010). In 2014, research revealed that CS received less than 1% of the educational funding allocated to schools for STEM (Partovi, 2014). Furthermore, as recently as 2015, CS curriculum was unavailable in the vast majority of schools in the U.S. (Barendsen et al., 2015).

The majority of American computer and technology companies have responded to the lack of CS-trained American workers by lobbying the federal government to allow more technology-credentialed workers from other countries to work in the U.S. (Preston, 2015). The efforts are significant because the CS industry is responsible for much of the economic growth and opportunity in STEM (Partovi, 2014). Furthermore, technology companies in the United States have pushed hard for a fast-track, green card application

process for foreign graduates who excel in CS education and other STEM-related areas, and for doubling the number of visas awarded through the H1-B program (Preston, 2015).

Due to the urgency of the situation, President Obama introduced numerous campaigns and initiatives such as: The Educate to Innovate Campaign in 2009, the TechHire Initiative in 2015, and more recently the Computer Science for All Initiative in 2016. The initiatives represent a collaborative effort between the government, private sector, nonprofit, and research communities to provide multimillion-dollar funding to encouraging youth, especially those from underrepresented groups, to become involved in STEM fields and to pursue technology-related careers (“Fact Sheet,” 2015, 2016; Holdren, Lander, & Varmus, 2010; Smith, 2016). The President expressed his belief that great teaching is an important part of any child’s success, and that the Initiatives will help to prepare more teachers in the STEM areas and to introduce them to different technologies and tools that can be implemented in the classroom (Repenning et al., 2015).

The President also mentioned that developing the technology skills of our workforce is important for our economic future and is a critical need for employers today. Over half a million of today’s job openings are in technology fields such as software development and cyber security, many of which did not exist a decade ago (“Fact Sheet,” 2016). According to the White House, the average salary for a job that requires information technology (IT) and CS skills is 50% higher than an average job in the private sector. The campaigns and initiatives created by the government call for various governmental and educational institutions as well as IT companies to empower Americans with the skills they need through traditional (e.g., school) and nontraditional

(e.g., web-based tutoring and classes) approaches.

A primary example of a nontraditional approach to introduce CS was the Hour of Code Initiative (<http://code.org>). Every year during the Computer Science Education Week, the Initiative challenges millions of users (mainly targeted at the K-12 levels) worldwide to spend at least one hour in a coding exercise (Israel et al., 2015). One of the benefits of using the nontraditional approach to introduce CS is that users could learn CS on their own, in their leisure time, and at their pace. The traditional and nontraditional methods to learn CS could improve individual skills in daily life and be advantageous for job-searching purposes.

Computing Education in K-12

One of the exciting things about learning CS is that users learn new and fundamental ways of thinking and problem solving, called *computational thinking* (CT). Computational thinking is one of the big advantages of studying computer science, and there is a growing interest in incorporating it in reading, writing, and math as a core ability that every student should learn (Wing, 2006, 2011). One way to introduce CT is to put a new curriculum into the K-12 systems in which teachers can introduce CT to children as early as the kindergarten level, so that as they grow they can become more proficient in problem solving and logical thinking, algorithmically and recursively. In the following sections, research studies that have been conducted in the K-12 environment are highlighted in which teachers were introduced to CT training.

Teaching CT

As CT becomes a fundamental skill for the 21st century, K-12 teachers should be exposed to computing principles. Yadav, Zhou, Mayfield, Hambrusch, and Korb (2011) conducted a study that looked at preservice teachers' attitudes toward CT in a required educational psychology course at a large university in the Midwest. Researchers surveyed 100 preservice teachers: 55% were preparing to teach at the elementary level and 45% at the secondary level. The results suggested that 95% of preservice teachers' attitudes toward CS became more favorable after the researchers implemented a 1-week module on CT in the course. Specifically, the educators were more likely to integrate computing principles in their future teaching.

In another study, Bell, Frey, and Vasserman (2014) investigated methods of how to introduce programming to preservice teachers by teaching a workshop held for sixth-through ninth-grade students. Five art and music preservice teachers and one inservice teacher participated in the study. During each of four week-long sessions, the teachers-in-training gradually took over more teaching responsibilities by modifying and presenting lessons that incorporated their own music and art expertise into the programming activities. Student surveys showed that self-efficacy towards programming, enjoyment of programming, and interest in continuing to program increased over the course of the sessions. Meanwhile, after the initially skeptical teachers were trained in programs such as Scratch, they expressed an interest in continuing to use the tools in their teaching (Bell et al., 2014).

Despite the growing need to integrate CS into K-12 education, many inaccurate perceptions of CS exist that influence attitudes toward CS learning and careers (Armoni,

2011; Israel et al., 2015). Research suggests some of the reasons for the declining enrollment in CS are related to teachers' attitudes that the only computing experiences available to students occur through learning programming languages such as Java or C++ (Burke & Kafai, 2012; Goode, 2007; Siegfried, Chays, & Herbert, 2008; Wilson & Moffat, 2010). Because complex programming languages are introduced to provide computing experiences, students often think that CS is boring, confusing, and too difficult to master (Israel et al., 2015; Wilson & Moffat, 2010).

Research is also needed to understand how to best support teachers to improve their attitude towards CS and develop their capacity to teach CS skills to their students in engaging ways. Currently, the longest running CS teaching program in the U.S. is the Exploring Computer Science (ECS) Initiative (Goode, Margolis, & Chapman, 2014; Margolis, Goode, & Ryoo, 2014). The program, started in 2012 and targeted initially at students in Los Angeles, has grown from a local to a national one. Margolis et al. noted that ECS is not just a program that teaches CS skills to students, but it also includes the ECS professional development program for teachers. The course was developed around a framework of both CS content and computational practice. The preliminary findings of the study indicated that ECS participation produced a robust and significant increase in students' self-assessment of their CS-related knowledge and skills. The findings also indicated a significant increase in students' interest in pursuing CS-related coursework and motivation when dealing with computer problem-solving activities.

When looking at CS education reform in the U.S., it is helpful to learn from similar efforts in other countries. For example, the United Kingdom (UK) has implemented a computing curriculum in which CT concepts and skills were taught, even

though many of its teachers had no computing background and were unprepared to implement the new curriculum changes in their classrooms (Curzon, McOwan, Plant, & Meagher, 2014). Scholars in the UK (e.g., Curzon et al., 2014) have examined the effectiveness of using *unplugged* computing methods to introduce teachers to CT topics. Unplugged computing is an approach to teaching computer concepts using constructivist activities away from computers. The activities introduce teachers to basic CT ideas through concepts such as debugging, binary numbers, algorithms and data compression, through the use of board games and puzzles. The activities provide teachers with a “programming-free” way to think about algorithms and problem solving without having to worry about the details that actual codes and programming languages impose (Lamagna, 2015; Taub, Armoni, & Ben-Ari, 2012).

Tools for Learning CT

While many efforts have focused on helping young programmers to become more interested in coding, the use of programming to teach problem-solving skills in K-12 declined significantly after the creation of Logo, a computer programming language created by Seymour Papert (Lye & Koh, 2014). However, in recent years, there has been a renewed interest in introducing programming to K-12 students (Grover & Pea, 2013; Kafai & Burke, 2013). The interest has been fueled by the availability of visual programming languages, such as: Scratch, Google Blockly, Tynker, Greenfoot, and Storytelling Alice.

With the recent developments in the visual programming languages for K-12, there is a renewed interest to consider how computer programming can benefit K-12

students (Barr & Stephenson, 2011; Grover & Pea, 2013; Resnick et al., 2009). Furthermore, Resnick et al. provided suggestions about features of effective tools for teaching CT in K-12 education. The term “low floor, high ceiling” that he suggested essentially means that though it should be easy for a beginning student to create a working program (low floor), the tool should be complex enough to fulfill the needs of advanced programmers (high ceiling). To make these types of CT-rich environments a reality, teachers must present to students the tools that have low floors and high ceilings (Grover & Pea, 2013). The tools for learning CT must adequately scaffold student learning, enable knowledge transfer, support equity, and be systemic and sustainable (Grover & Pea, 2013; Reppenning et al., 2010).

Interest and Engagement in CT

When selecting tools for teaching CT, one should note that effective tools have the potential to increase student interest and excitement. Lye and Koh (2014) reviewed 27 articles that mentioned the term *computational thinking*. Of those, only nine studies were conducted in the K-12 environment, and most of them reported positive outcomes including increased positive attitudes about computing and CS (Lambert & Guiffre, 2009; Lin, Yen, Yang, & Chen, 2005) as well as increased skills in computer programming (Baytak & Land, 2011; Kwon, Kim, Shim, & Lee, 2012).

Efforts to prepare precollege students for a career in computer science typically use two strategies: increase student interest and excitement about computing, and introduce them to computational concepts and skills (Denner, Werner, & Ortiz, 2012). In 2012, the CSTA revised its CS standards that targeted mainly high school students,

providing researchers and instructors with information about how to improve implementation of CS skills at the high school level. Roughly two thirds of the 50 states have no CS standards for secondary school education (Barendsen et al., 2015; Wilson, Sudol, Stephenson, & Stehlik, 2010). Despite its importance as an academic field, few states count CS as a core academic subject for graduation (CSTA, 2012).

At the K-8 level, more work needs to be done to encourage instruction in the area of CS. Even when they exist, computer science standards at the K-8 level often confuse CS with the use of software applications (CSTA, 2012). Furthermore, educators remain wary of introducing new core subjects into curricula already challenged by high-stakes test preparation and accountability measures (Burke & Kafai, 2010). More resources are needed to support successful integration of CS into core subjects, especially at the K-8 level. By successfully introducing and integrating CS in early education, students could become better critical thinkers and problem solvers, a set of skills that many scientists described as computational thinking.

Computational Thinking

A topic that has been discussed recently in the context of K-12 education involves the use of computer programming as an approach to introduce CT to children. Although the idea of CT is not new, establishing a definition that everyone agrees with has proven difficult for the CS education community (Mannila et al., 2014). Specifically, there has been little agreement regarding what CT encompasses (Allan, Barr, Brylow, & Hambrusch, 2010; Barr & Stephenson, 2011; Berland & Lee, 2011; National Research Council, 2010), and even less agreement regarding what strategies to use for assessing

the development of CT in youth (Brennan & Resnick, 2012). According to Wing (2006, 2008), the computer scientist who coined the term CT, it means thinking like a computer scientist and using principles and concepts learned in CS as part of one's daily life. A couple of examples of CT key concepts include *problem decomposition*, which is breaking down a problem into smaller, manageable parts, and *algorithmic design*, which is developing step-by-step instructions for solving problems.

While other researchers have attempted to define CT (e.g., Ater-Kranov, Bryant, Orr, Wallace, & Zhang, 2010; Denning, 2009; Guzdial, 2011), Steve Furber (2012) from the Royal Society offered a more concise definition that it is using the methods of CS to understand a wide variety of topics. It has also been suggested that CT is the process of recognizing aspects of computation and applying tools and techniques from CS to understand natural and artificial systems and processes (Nickerson, Brand, & Reppenning, 2015). However, even with the available definitions of CT, there are few specifics as to what skills comprise CT and how to achieve those skills. Researchers and CS educators for the most part now work broadly with the aforementioned definitions of CT.

Components of CT

Although the definitions of CT vary widely, researchers (e.g., Brennan & Resnick, 2012; Grover & Pea, 2013; & Moreno-León & Robles, 2015) identified core components to provide a working definition of CT. For example, Brennan and Resnick introduced seven key concepts through the CT programming language, Scratch, including sequences, loops, parallelism, events, conditionals, operators, and data. Additionally, Grover and Pea examined essential concepts of CT suitable for use in K-12 education, including:

abstractions and pattern generalizations; systematic processing of information; symbol systems and representations; algorithmic notions of flow of control; structured problem decompositions; iterative, recursive, and parallel thinking; conditional logic; efficiency and performance constraints; and debugging and systematic error detection. The concepts they proposed have been endorsed by the CSTA.

Over the years, researchers have attempted to clarify and improve the understanding of CT components. Even though CT components continue to shift as the definition progresses, the new components are nonetheless consistent with the nine key components suggested by Grover and Pea (2013). For example, Moreno-León and Robles (2015) created their own list by highlighting seven components of CT that included: abstraction and problem decomposition, parallelism, logical thinking, synchronization, flow control, user interactivity, and data representation. Table 1 shows how the different perspectives on CT vary and establishes some common components. In the present study, I used the CT tool that Moreno-León and Robles developed to assess the CT skills of middle school students through the use of Dr. Scratch.

Assessing CT

Despite the efforts aimed at the assessment of CT (Basawapatna, Koh, Repenning, Webb, & Marshall, 2011; Fields, Searle, Kafai, & Min, 2012; Grover, Pea, & Cooper, 2014; Meerbaum-Salant, Armoni, & Ben-Ari, 2013; Werner et al., 2015), evaluating the learning of CT concepts and constructs in a programming environment such as Scratch remains a challenge. The use of surveys has been one of the main methods used to analyze CT (Bell et al., 2014; Clark, Rogers, Spardling, & Pais, 2013; Mishra, Balan,

Table 1

Components of Computational Thinking Referenced in the Prior and Present Studies

Components of CT	Moreno-León & Robles (2015)	Grover & Pea (2013)	Brennan & Resnick (2012)	Present study
Abstraction	✓	✓	✓	✓
Parallelism	✓	✓	✓	✓
Logical thinking (e.g., conditional logic, operators, events)	✓	✓	✓	✓
Synchronization	✓		✓	✓
Algorithmic notions of flow of control	✓	✓		✓
User interactivity	✓		✓	✓
Data representation	✓		✓	✓
Iterative and recursive thinking (e.g., loops)		✓	✓	
Efficiency and performance constraints		✓		
Debugging and systematic error detection		✓		
Pattern generalization		✓		
Systematic processing of information		✓		

Iyer, & Murthy, 2014). Even though the results from surveys provide answers to some important questions, surveys alone are inadequate to detail how CT has been assessed (Grover et al., 2014). Thus, more research is needed to better assess CT.

Researchers have developed new methods for measuring student growth in CT using available CT evaluation tools, such as Scrape (Riversound Media, n.d.; Wolz, Hallberg, & Taylor, 2011), Hairball (Boe et al., 2013), and Dr. Scratch (Moreno-León & Robles, 2015). Nonetheless, there is a lack of tools that support educators in the

assessment of the development of CT and the evaluation of projects programmed by students. Several researchers (e.g., Boe et al., 2013; Close et al., 2015) proposed different approaches for evaluating the development of CT by analyzing students' projects, but most tools require intermediate knowledge of programming skills, which make them less suitable for educators who are not confident with such environments.

In one study, Fields, Giang, and Kafai (2014) examined ways to provide teachers with information about what CT scores revealed from four groups of Scratch youth programmers. The profiles of the four groups were identified as: beginner, intermediate, advanced, and experienced. Individuals considered as beginners were those who created a simple Scratch project using relatively few loops and almost no other advanced concepts. Intermediate students used more complex programming concepts, except Boolean. And advanced and experienced were those who used all programming concepts, including Boolean. The length and complexity of the students' Scratch projects were the main factors that the researchers used to determine whether students were in the advanced or experienced groups. The programming profiles were also introduced in the Dr. Scratch community to help Scratch users to determine their CT progress.

Context of Current Study

Scratch

Scratch programming has risen in prominence as a useful programming language for K-12 CS curricula (Boe et al., 2013). Visually based programming languages such as Scratch facilitate K-12 students' CT, because traditional programming syntax is reduced (Lye & Koh, 2014). Using Scratch, students may create their own interactive stories,

games, and simulations. An example of the different types of Scratch genre that users can create and choose from is shown in Figure 1.

Using an intuitive *drag-and-drop code* mechanism helped reduce the cognitive load of Scratch users, making the testing and debugging process less demanding (Resnick et al., 2009). The mechanism enabled students to develop computational problem-solving practices more easily and focus on the logic and structures involved in programming rather than worrying about the mechanics of writing programs (Kelleher & Pausch, 2005). Through the use of Scratch, students learn mathematical and computational concepts, as well as think creatively, improve reasoning skills, and work collaboratively.

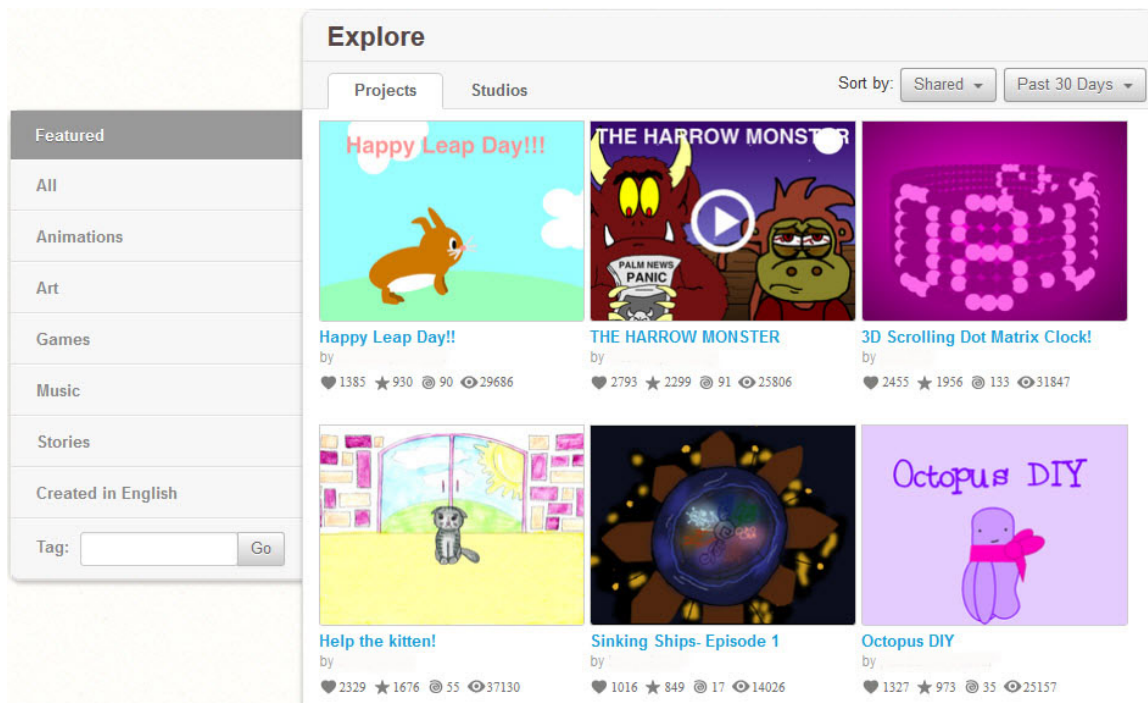


Figure 1. An example of different types of Scratch genre.

Dr. Scratch

The Scratch community includes users, developers, and scholars. A group of developers created Dr. Scratch, a digital instrument that is easy to use without the need for background or programming knowledge. Dr. Scratch is a user-friendly, free/open-source web application tool that allows researchers and instructors to visually analyze students' CT, and it can automatically measure the degree of CT evidenced in a certain Scratch project (Moreno-León & Robles, 2015). Dr. Scratch focuses on the elements of CT most easily interpretable, such as: abstractions and problem decomposition, parallelism, logical thinking, synchronization, flow control, user interactivity, and data representation (see <http://drscratch.programamos.es/> for more information). Below are short definitions of the CT elements measured by Dr. Scratch.

- Abstractions and problem decomposition are defined as the ability to filter out information that is unnecessary to solve a problem and at the same time generalize information that is necessary. It is also the ability to break down a task into minute details so that the process may be clearly explained to another person or to a computer, or even to write notes for ourselves (Google for Education, n.d.).
- Parallelism is defined as the ability to engage in a thinking process where the focus is split in specific directions and involves many repetitions (Dr. Scratch, n.d.).
- Logical thinking is the ability to use an “if-then-else” construct. It requires a student to think globally about the local consequences of the truth-value of a given statement (Berland & Lee, 2011).
- Synchronization is the ability to thoroughly understand available information through careful attention, deep thinking, and intensive reasoning. Thereafter, connecting one piece of information to another is the next step to make sense of a particular problem (Chaiken & Ledgerwood, 2012).
- Flow control is the ability to create a data “recipe” or set of instructions. In its simple form, it is the planning of actions for events that are taking place. In its complex form, it is planning for unknown events (Berland & Lee, 2011).




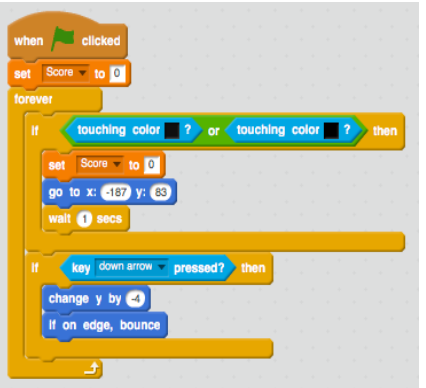
- User interactivity is the ability to input information that would change the program, such as integrating audio or video into the programming codes in order to trigger certain actions (Dr. Scratch, n.d.).
- Data representation is the ability to display the correct order of programming codes so that the program can run properly (Dr. Scratch, n.d.).

For each of these measures, users can earn 0 to 3 points. For example, a user might earn 0 points for synchronization if they only used “wait commands” to sync up two or more scripts. A user can earn 1 point for broadcasting messages to other scripts, 2 points if the broadcasted messages have complex wait commands that ensure scripts run in a certain order, or 3 points if the user fulfills all of the criteria described by Dr. Scratch (for another example, see Table 2).

The information was then organized into user-friendly dashboards that showed student progress and allowed teachers to personalize instruction and to cater to individual student needs. Dr. Scratch provided scores related to each individual CT component. A score of 3, for example, indicated student proficiency in that area, while zero meant that the skill was not evident. The scores were then totaled, creating a programming profile (beginner, developing, and master) to show the user’s competence in CT. Depending on the overall CT score (which could range from 0 to 21), distinct data were displayed on the dashboard page and suggestions on how to improve their programming habits were offered. The tool also provides links to information that could be used to improve skills related to CT components. Teachers need resources like these to support successful integration of CT into core subjects, especially at the K-8 level.

Table 2

Scoring System Measuring User Interactivity

Points	Evidence	Example code
0	Uses only the most basic interactive block “When green flag is pressed” block.	
1	Uses other types of interactive blocks utilizing mouse clicking, mouse positioning, question asking blocks, and sprite clicking.	
2	Uses complex interactive blocks utilizing webcam and microphone input.	
3	If all of the requirements for user interactivity according to Dr. Scratch are met. Scratch blocks are in chronological working order.	

Study Overview

This study was built on past research and addressed the growing need to teach and implement CT in classrooms, especially for middle school students. Even though educators are now able to implement CT skills in classrooms, little is known about the struggles that students experience during the development of CT. This study addressed this gap in the literature by answering the following research questions.

1. How does Dr. Scratch function as an assessment for CT?
 - a. Among the seven CT components used in student projects, which are common areas of strength and weaknesses for the seventh- and eighth-grade students?
 - b. What were some of affordances and constraints of Dr. Scratch as a CT tool?

CHAPTER III

METHODS

Research Design

In this study I used a quantitative research approach to explore students' strengths and weaknesses related to CT skills while learning with Scratch. Scratch was used because it has a large youth user community with publicly available data. Furthermore, the Scratch interface allows an easier interpretation of core CT constructs. I conducted a secondary data analysis of an extant dataset collected from seventh- and eighth-grade students during the 2013 to 2015 school year, which is publicly available from the Scratch website (<https://scratch.mit.edu/>).

I designed the study to address two research questions to understand the effectiveness of Dr. Scratch as a tool to assist programming learners: (a) among the seven CT components used in student's projects, what were common areas of strength and weakness for the seventh- and eighth-grade students? and (b) what were some affordances and constraints of Dr. Scratch as a CT tool? The research objective related to the first research question is to examine where seventh- and eighth-grade students excel and are challenged when considering the CT components described by Moreno-Leon and Robles (2015). Whereas the objective for the second research question is to find strengths and weaknesses of Dr. Scratch as a tool to evaluate students Scratch projects, I will discuss the procedures I used to collect data for 360 students and to evaluate students' strengths and weaknesses.

Sample

I collected the extant data from publicly available information on the Scratch website. This study was approved as an exempt study by the University Institutional Review Board (IRB), given the analysis of publicly available data (see Appendix D). I found the data by searching the MIT Scratch website using keywords such as: middle school projects, seventh-grade Scratch projects, and eighth-grade Scratch projects. I selected data from one Scratch project curator (a teacher) for the study because of his frequent updates to his students' Scratch projects since 2013.

The curator of the Scratch projects was a teacher from the East Coast of the United States who taught CS to middle school students. No contact with the teacher was made before or after the study, and the information regarding school information was provided in his Scratch profile website. The dataset was comprised of 360 seventh- and eighth-grade student projects published from the 2013 to 2015 academic years. From the 2013 to 2014 school year, there were 102 seventh- and 75 eighth-grade student projects, while from the 2014 to 2015 school year, there were 93 seventh- and 90 eighth-grade student projects. Demographic information such as age, gender, and race of the students was not publicly available on Scratch so I was not able to include that information as part of the study. Although the specific classroom practices and demographics of the participants were unknown, all seventh- and eighth-grade students were taught by the same teacher. Thus, it was reasonable to assume that students received similar instructions for their respective grades.

Within this dataset, all seventh-grade student projects from 2013 to 2015 school

years were maze games. The eighth-grade student projects from both school years had more project genre variation, which might have been due to teacher expectation or assignment constraints, though the majority of the eighth graders (75% and 91%, for 2013 to 2014 and 2014 to 2015, respectively) also created a game-based project (see Figure 2).

Measures

I used Dr. Scratch to analyze students' Scratch projects. The amount of information Dr. Scratch provided about a given project was dependent on the resulting CT score. If the CT level was low, Dr. Scratch assumes the user was a novice programmer, and Dr. Scratch shows only basic information of the most important improvements to perform in the Scratch project (see Figure 3). As the scores increase and users became more advanced at programming, more detailed information is provided by

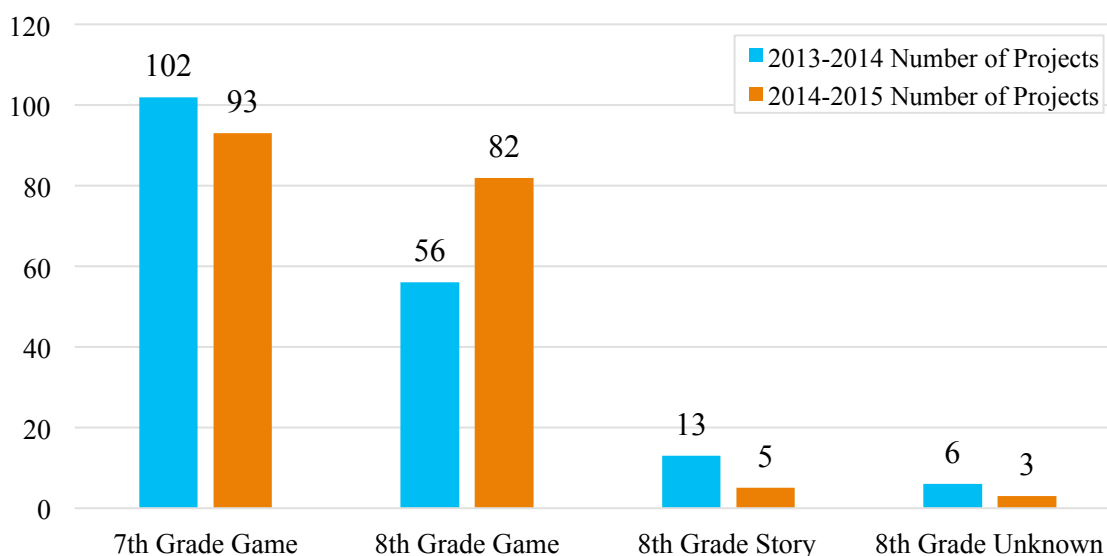


Figure 2. Scratch projects genres for seventh- and eighth-grade students across 2013 to 2015 academic years.

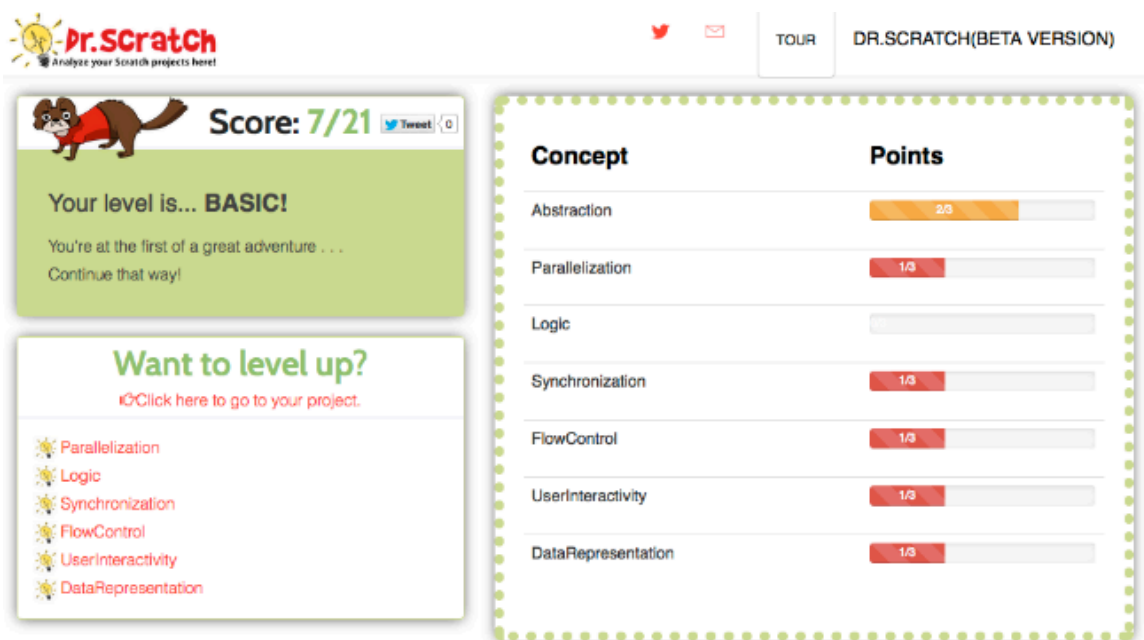


Figure 3. An example of Dr. Scratch analysis results for a project with a basic CT score.

Dr. Scratch for a project, including bad programming habits, such as: duplicated scripts, default sprite naming, and dead/useless Scratch blocks (see Figure 4). Although Moreno-León and Robles (2015) did not address specifically how they came up with the score of 0 to 21 nor how they related the scores to the programming profiles, the scores were an important part of this study and provided me with information regarding a student's area of difficulty with individual CT components.

Using Dr. Scratch, I analyzed projects for evidence of seven CT components (abstraction, parallelism, logic, synchronization, flow control, user interactivity, and data representation) on a scale from 0 to 3 points. For example, in the abstraction and problem decomposition category, students who used more than one script or sprite earned 1 point; students who defined their own blocks earned 2 points; students who used clones earned 3 points. Students with none of these coding sequences or blocks earned 0 points.

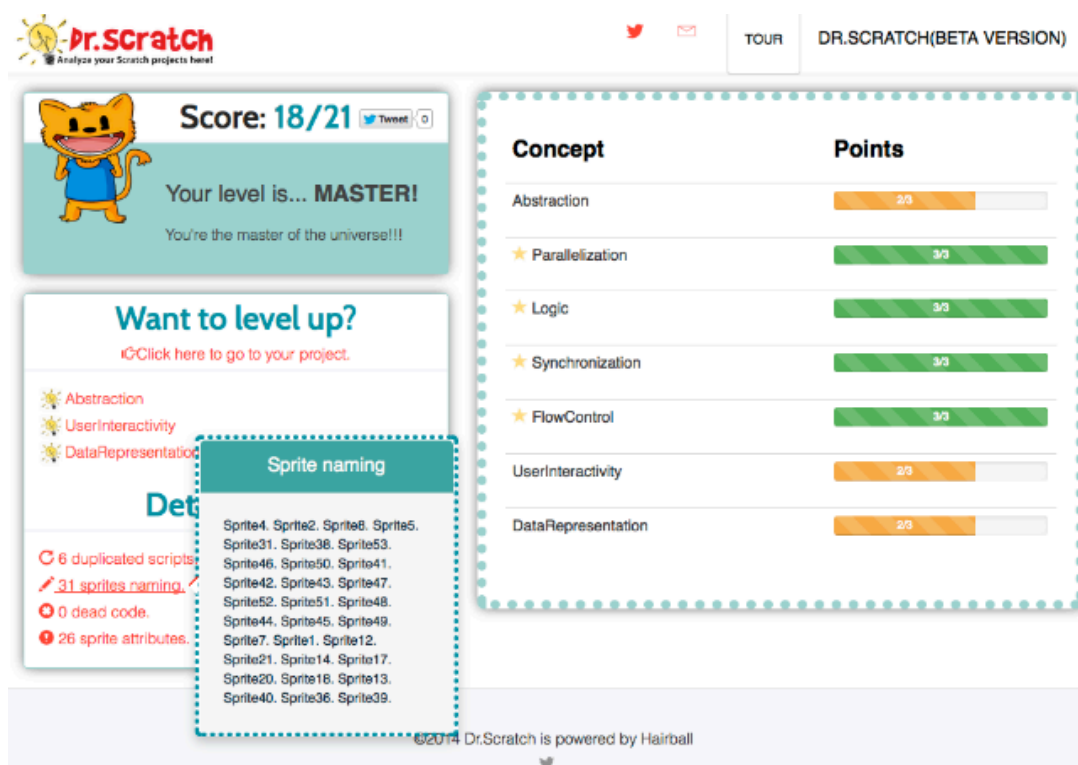


Figure 4. An example of Dr. Scratch analysis results for a project with an advanced CT score.

Likewise, for each category, Dr. Scratch searched for particular pieces of code or certain blocks (see Table 3).

The results from individual Scratch projects that were analyzed by Dr. Scratch allowed me to address the specific CT components in which students were lacking. Also, the results offered suggestions regarding how students could improve their programming habits by providing best practice information that students could follow.

Procedures

Each individual Scratch project URL was copied from the Scratch website for the sample of seventh- and eighth-grade students for projects created during the 2013 to

Table 3

Rubric for Scoring CT Components

CT Component	Score		
	1 Point	2 Points	3 Points
Flow control	Used sequence of blocks	Used <i>repeat</i> and <i>forever</i> blocks	Used <i>repeat until</i> block
Data representation	Used modifiers for sprite properties	Used operations on variables	Used operations on lists
Abstraction	Used more than one script and more than one sprite	Defined own block	Used clones
User interactivity	Used <i>green Flag</i> block	Used <i>key pressed</i> , <i>sprite clicked</i> , <i>ask and wait</i> , <i>mouse</i> blocks	Used video and audio features
Synchronization	Used <i>wait</i> block	Used <i>broadcast</i> , <i>when I receive message</i> , <i>stop all</i> , <i>stop program</i> , <i>stop programs</i> sprite	Used <i>wait until</i> , <i>when backdrop change to</i> , <i>broadcast and wait</i> blocks
Parallelism	Used two scripts on green flag	Used two scripts on key pressed, two scripts on sprite clicked on the same sprite	Used two scripts on <i>when I receive message</i> block, create clone, two scripts on <i>backdrop change to</i> block
Logic	Used <i>if</i> block	Used <i>if/else</i> block	Used logic operations

2015 academic years. Next, the individual projects were analyzed using the Dr. Scratch tool, and the results were transferred to an Excel spreadsheet. Further statistical analysis was conducted using SPSS statistical software.

Data Analysis

A primary focus of this research dealt with understanding how Dr. Scratch could function as an assessment for CT. Thus, two research questions were developed in order to assess the effectiveness of Dr. Scratch as a tool to assist programming learners: (a)

among the seven CT components used in student's projects, what were common areas of strength and weakness for the seventh- and eighth-grade students? and (b) what were some affordances and constraints of Dr. Scratch as a CT tool? To answer these questions, first, I created an Excel spreadsheet that contained the following information: student grade, project name, student Scratch username, Scratch project genre (i.e., games, stories, music, animations, and art), and the seven CT components that Dr. Scratch measured. Second, I analyzed individual Scratch projects using Dr. Scratch, and the score (up to 3 points) for each CT component for an individual student's project was transferred to an Excel spreadsheet. To address the first research question, to understand strengths and weaknesses in the seven CT components, I used descriptive statistics (mean and standard deviation of each CT component) with formulas in Excel. After I analyzed the projects of all 360 students, I did a visual analysis of common trends related to which CT components most middle school students missed or achieved a low score on as well as which they achieved the highest scores on.

I used the Mann-Whitney U test to determine significant similarities or differences between the two grade levels. The Mann-Whitney U test is a nonparametric method designed to detect whether two or more samples come from the same distribution or to test whether medians between comparison groups are different, under the assumption that the shapes of the underlying distributions are the same. However, if the two distributions have a different shape, the test may be used to compare mean ranks. I used the Mann-Whitney U test to compare the mean rank of each of the seven CT components between the seventh- and eighth-grade students' projects.

To answer the second research question that dealt with affordances and

constraints of Dr. Scratch, I looked at seventh- and eighth-grade students' programming profiles provided by Dr. Scratch. The programming profiles were displayed next to the students' CT scores. There were three different programming profiles that Dr. Scratch provided (i.e., beginner, developing, and master), depending on the students' overall CT scores. Lastly, I highlighted several bad programming habits that Dr. Scratch identified for the students. Specifically, Moreno-León and Robles (2015) summarized bad programming habits into four categories (i.e., sprite naming, sprite attributes, duplicated scripts, and dead codes) in order to assist evaluators to detect the malpractices and to propose ideas to try to avoid such situations. Dr. Scratch identified the profile and any areas of bad programming habits as part of its analysis output.

CHAPTER IV

RESULTS

Publicly available Scratch data were compiled, cleaned, and analyzed in Microsoft Excel 2016 and IBM SPSS version 21. Descriptive statistics were obtained to determine which CT components most middle school students missed, as well as which ones they excelled on. Then, the Mann-Whitney U test was used to determine significant similarities or differences between seventh- and eighth-grade levels. The following section discusses the findings based on the statistical tests conducted.

Strengths and Weaknesses in Students' CT

The first part of the research question was: Among the seven CT components used in student's project, what are common areas of strengths and weaknesses for the seventh- and eighth-grade students? To answer the question, the means of all student ($n = 360$) scores for each CT measurement were calculated. The results showed that students were, on average, best at synchronization, parallelism, and flow control. These strengths were followed by user interactivity and logic, with relative weaknesses in data representation and abstraction (see Appendices A and B for means and standard deviations for each CT measurement). The data indicated that common strengths and weaknesses existed among middle school students and, in particular, that middle school students excelled at synchronization, parallelism, and flow control, but struggled with abstraction and data representation.

A histogram was also created to examine specific difference in seventh- and

eighth-grade students' Scratch projects. Histograms of student projects from both 2013 to 2014 and 2014 to 2015 school years showed the distribution of CT component scores.

The comparisons are provided in Figure 5. Based on the histograms, there were similarities for abstraction and user interactivity, showed by a similar bar trend for both school years. The two histograms showed that both seventh- and eighth-grade students achieved a low score on abstraction and a moderate score on user-interactivity. Other CT

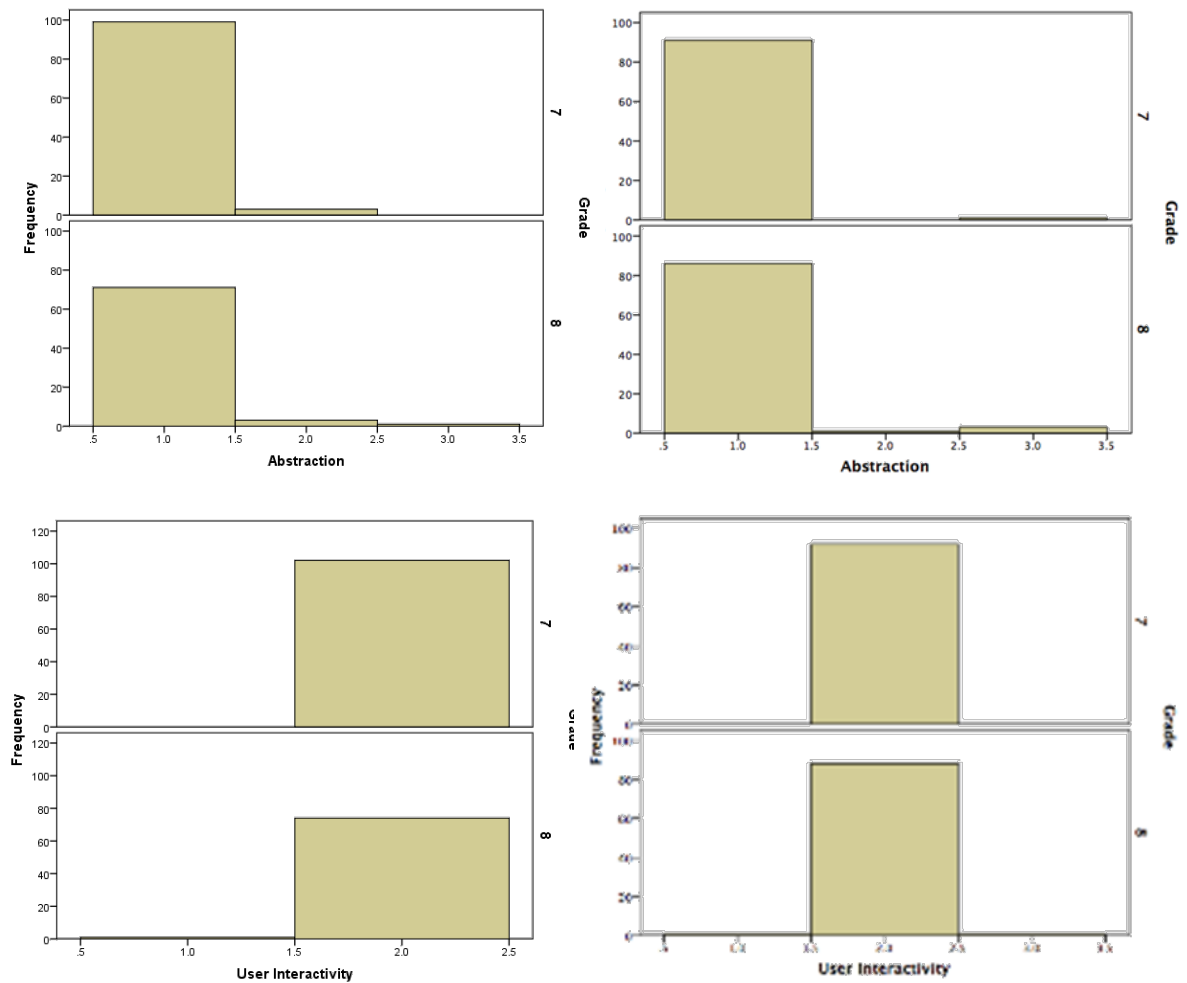


Figure 5. Histograms comparing seventh- and eighth-grade abstraction and user interactivity component scores from 2013 to 2014 (left) and 2014 to 2015 (right) school years.

components such as flow control, logic, synchronization, data representation, and parallelism, showed a fluctuation and a slight difference in their CT components scores, in which typically the seventh-grade students overall achieved lower scores compared to the eighth-grade students. It may be concluded that the eighth-grade students were more experienced in programming with Scratch compared to the seventh-grade students (see Appendix C).

When comparing the descriptive statistics of seventh- and eighth-grade students' projects, there was a similarity in students' CT average scores from 2013 to 2014 and 2014 to 2015 academic years. In both academic periods, a small standard deviation ($SD = 0.11$) occurred in user interactivity, which revealed little variance in user scores; however, logic scores varied greatly ($SD = 1.01$). From 2014 to 2015, students' project synchronization was highly variable ($SD = 0.89$), which suggested that they showed strong differentiation in synchronization and logic skills, but very little in user interactivity skills. From 2013 to 2014, students' project synchronization ($SD = 0.60$) was not as highly variable as in the later years, which suggested that they showed a somewhat strong differentiation in synchronization and logic skills, but little in user interactivity skills. The numbers were beginning to show intrinsic characteristics of certain CT elements.

Further analysis revealed common CT strengths and weaknesses. If such a common pattern existed, one would expect the strengths and weaknesses of seventh- and eighth-grade students to be similar, inasmuch as they are typically separated in age by only 1 year. The mean scores of seventh and eighth graders for each of the CT components within the entire dataset for the 2013 to 2015 school years are highlighted in

Tables A1 and B1 in Appendices A and B. Using the mean scores, I created similar web-like figures (see Figure 6), as shown in the *Analyze your Scratch Projects with Dr. Scratch* paper (Moreno-León & Robles, 2015) to show students' mean distributions among CT components. On the web-like figures, the mean scores of the CT components near the edge indicated higher levels of mastery, while mean scores near the middle indicated a lower level. Therefore, if there was a common pattern of strengths and weaknesses to detect, one would expect the webs of seventh and eighth graders to have a similar shape. As shown in Figure 6, the shapes were similar, indicating a common pattern of strengths and weaknesses. The student projects from 2014 to 2015 showed a better CT performance overall, as indicated by the web figure that was more spread out compared to the figure from the previous school year. Eighth-grade students showed higher scores in every category, except for user interactivity. Furthermore, the Mann-Whitney U test was also used to highlight significant similarities or differences between seventh- and eighth-grade students' Scratch projects.

The Mann-Whitney U test indicated that there were slight variations between the two school years. For the 2013 to 2014 school year, the test indicated that three of the seven CT components (parallelism, logic, and data representation) were greater for eighth graders than for seventh-grade students. Furthermore, in the 2014 to 2015 school year, the test indicated that five of the seven CT components (flow control, synchronization, parallelism, logic, and data representation) were greater for eighth-grade students than for seventh graders (both $p < 0.05$; also see Table 4 for results).

In the 2013 to 2014 school year, the lowest U scores occurred in data representation, logic, and parallelism, indicating the greatest level of variability between

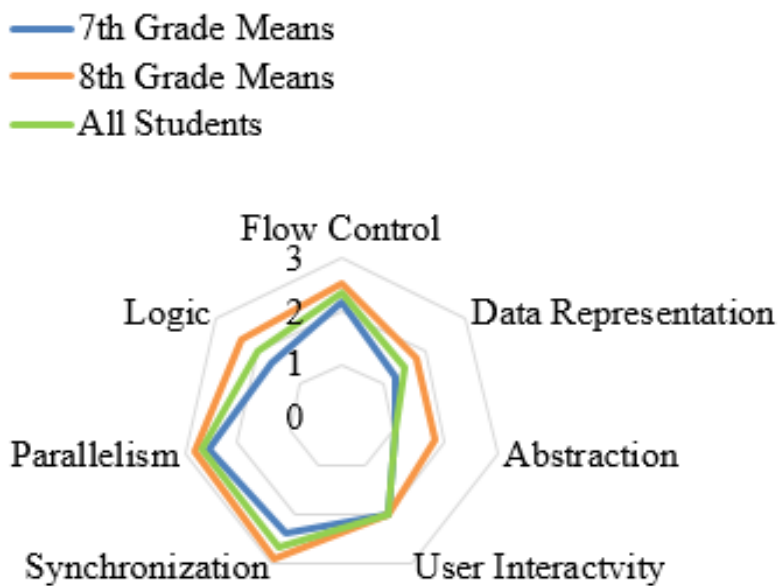
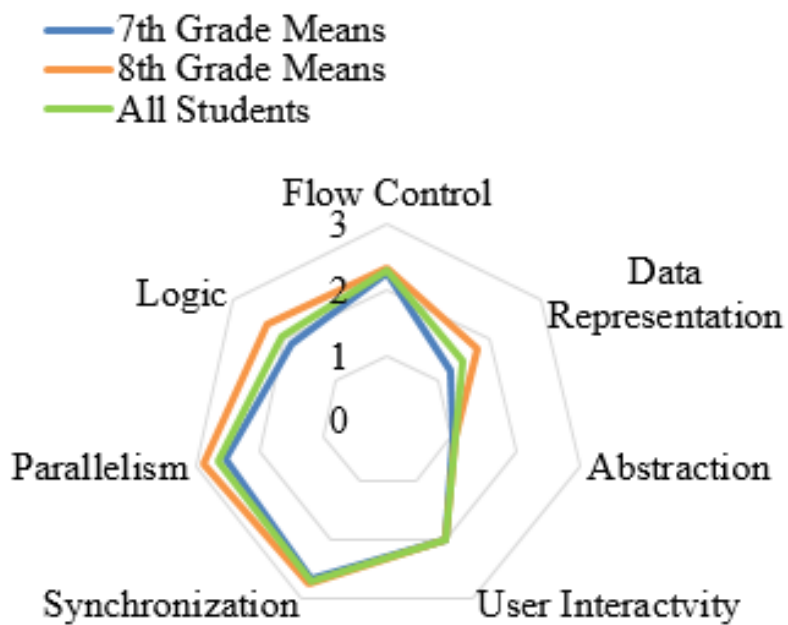


Figure 6. CT skill development for seventh and eighth graders from 2013 to 2014 (top) and 2014 to 2015 (bottom) school years.

Table 4

Mann-Whitney U Scores Across CT Components

School year	CT components						
	Flow control	Abstraction	User interactivity	Synchronization	Parallelism	Logic	Data representation
2013 to 2014							
<i>U</i>	3586.5	3732	3774	3596.5	3027.5	2951.5	1818
<i>p</i>	.378	.413	.244	.368	.001*	.003*	.001*
2014 to 2015							
<i>U</i>	2780.5	4001.5	4140	3375	3401	2635	2034
<i>p</i>	.001*	.169	1.000	.001*	.003*	.001*	.001*

* $p < .05$.

seventh- and eighth-grade students. In other words, eighth-grade student projects resulted in significantly better CT scores for those three CT components. The remaining CT components indicated that differences for flow control, abstraction, user interactivity, and synchronization were not significant.

The lowest U scores were exhibited in data representation, logic, and flow control components, indicating the greatest level of variability between seventh- and eighth-grade students. In other words, eighth-grade student projects exhibited significantly better CT scores for flow control, synchronization, parallelism, logic, and data representation, with the greatest differences being in flow control, logic, and data representation. User interactivity and abstraction mean differences were not significant.

The findings illustrate potential strengths and weaknesses in CT for students in grades 7 and 8. The findings also suggest that CT does not develop evenly, as a unified construct, but rather, certain elements (e.g., data representation and logic) likely develop more drastically than others.

Affordances and Constraints of Dr. Scratch

The second part of the research questions involved understanding some of the affordances and constraints of the Dr. Scratch tool. Currently, few research studies on CT and Scratch have used Dr. Scratch as a tool to help measure users' CT skills and abilities, inasmuch as Dr. Scratch itself is a relatively new tool. Thus, this study examined Dr. Scratch and explored its affordances and constraints as a CT tool.

In this section, two examples were used to examine the affordances and constraints of Dr. Scratch. The examples are provided from seventh- and eighth-grade projects from 2013 to 2015 school years, in the developing and master categories. The developing group consisted of students who scored 8 to 14, and the master group included those who scored 15 to 21. No students in the seventh- and eighth-grade classes from either school year scored 7 or below, which was considered the basic category.

Developing

When comparing students' projects from the 2013 to 2015 school years in the developing category, the results indicated that there were more students in the seventh-grade group ($n = 134$), compared to those in the eighth-grade group ($n = 49$) who were in the same category. The majority of the seventh-grade students in the category received a score of 13 ($n = 63$), while the majority of the eighth-grade students received a score of 14 ($n = 19$). From the data analysis, the average CT scores across seven CT components indicated that there were similar trends in which both seventh- and eighth-grade students were low in three different CT areas (see Table 5): *data representation* ($\mu = 1.10$, $\mu =$

Table 5

CT Average Scores for Seventh- and Eighth-Grade Students in the Developing Category

Grade	Flow control	Data representation	Abstraction	User interactivity	Synchronization	Parallelism	Logic
Seventh							
CT avg. score	2.03	1.10	1.01	2.00	2.40	2.37	1.23
SD	.17	.31	.12	.00	1.08	.84	.62
Eighth							
CT avg. score	2.04	1.41	1.01	1.96	2.73	2.57	1.06
SD	.41	.50	.14	.20	.57	.71	.56

1.41, respectively), *abstraction* ($\mu = 1.01$ for both grades), and *logic* ($\mu = 1.23$, $\mu = 1.06$, respectively). In Figure 7, an example is provided of one of the seventh-grade student's projects in the developing category. Also shown is what they saw on their dashboard once their project was assessed by Dr. Scratch.

When a student received a score that placed him or her in the developing stage, Dr. Scratch would only yield two types of information that could be used to improve their Scratch project: sprite naming and sprite attributes. Dr. Scratch developers, Moreno-León and Robles (2014), considered these two types of information to be bad programming habits. When a student starts to program with Scratch, it is typical to leave the naming of the sprite with the default name. When Scratch users have a few sprites, it is easy to know the name of each of the characters; however, when the number of sprites increases, it is more complicated to detect errors in their Scratch codes. Therefore, it is a good practice to name each individual sprite in a project differently, because the programs may then be read more efficiently.

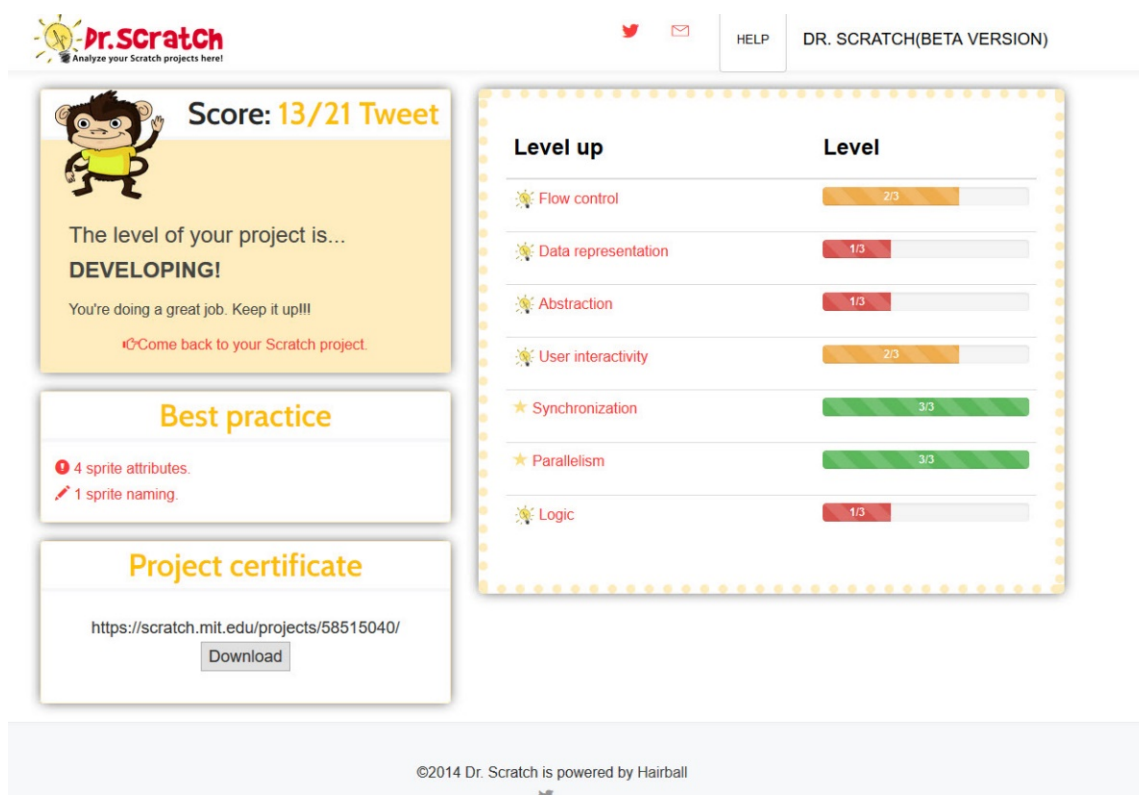


Figure 7. Example dashboard for one seventh-grade student in the developing category.

The second bad programming habit, according Moreno and Robles (2014), is attribute initialization. One of the mistakes that many programmers repeat when they learn to program is to initialize incorrectly the objects' attributes. Sprite attributes are the characters' features that can be modified in the execution of a project, for instance, their position, size, color, and orientation.

When students have blocks that modify the features of a character, they should always assign the value of their starting point. For example, students should have the block "go to" under the block "when green flag clicked," in order to place the character in its initial position (see Figure 8).

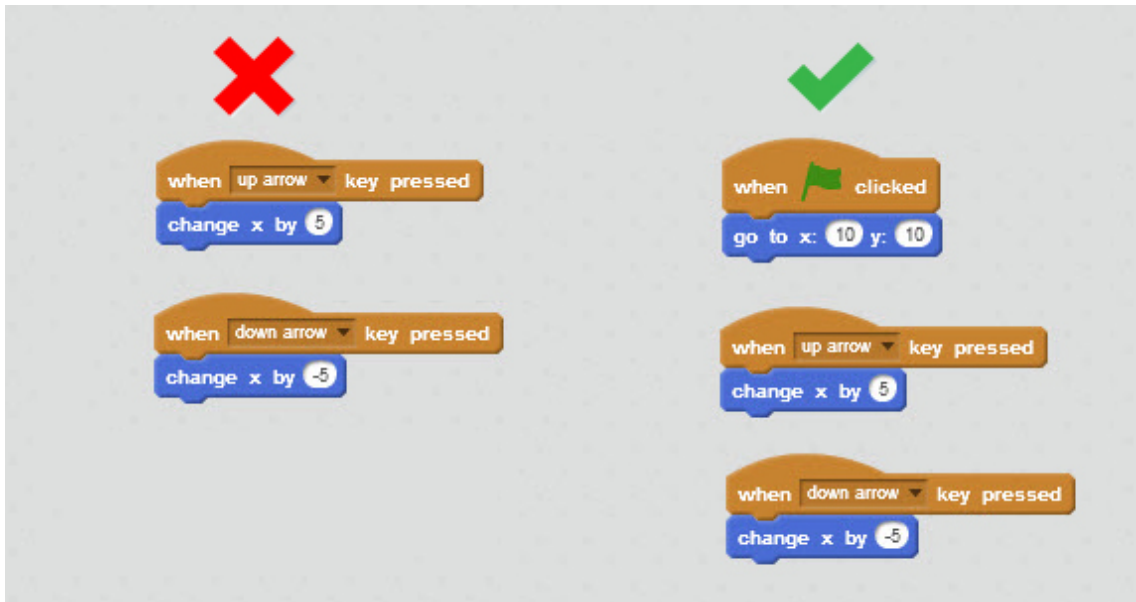


Figure 8. Example of an incorrect attribute initialization (left) and a correct attribute initialization (right).

Master

Unlike the results in the developing category, when comparing master category seventh- and eighth-grade students' projects from 2013 to 2015 school years, the results indicated that there were more students in the eighth-grade group ($n = 116$) compared to students in the seventh-grade group ($n = 61$). The majority of the eighth-grade students received a score of 17 ($n = 50$), while the majority of the seventh-graders received a score of 16 ($n = 31$). It should be pointed out that most of the seventh-grade students from the previous school year (2013 to 2014) were in the eighth grade currently and had become better at making their Scratch projects more complex (as shown by the improvement in their CT average score), compared to the previous year. From the data analysis shown in Table 6, the average CT scores across the seven CT components indicated that the seventh-grade students in the 2013 to 2015 academic years were low in data

Table 6

Computational Thinking Average Scores for Seventh- and Eighth-Grade Students in the Master Category

Grade	Flow control	Data representation	Abstraction	User interactivity	Synchronization	Parallelism	Logic
Seventh							
CT avg. score	2.67	1.57	1.05	2.00	2.84	2.93	3.00
SD	.47	.50	.28	0.00	.37	.25	0.00
Eighth							
CT avg. score	2.59	1.92	1.09	2.01	2.90	2.96	2.91
SD	.49	.30	.40	.09	.31	.24	.41

representation ($\mu = 1.57$) and abstraction ($\mu = 1.05$), while eighth-grade students from the same period were low in only abstraction ($\mu = 1.09$). In Figure 9, an example is provided of one of the eighth-grade student's projects scored by Dr. Scratch as being in the master category.

In addition to changing the sprite names and attributes, Dr. Scratch also added two more bad programming habits for students in the master category: duplicated scripts and dead codes. As a novice programmer, it is typical for students to duplicate their Scratch scripts to do the same tasks repeatedly. In this scenario, it was recommended that students make their own block to define the behavior and to use the new block in all programs where needed. Thus, if students wanted to change the outcome, they merely went to the block they defined (see Figure 10).

The fourth bad programming habit, according Moreno-León and Robles (2014), occurs when students put a dead code in their Scratch scripts. Dead codes are parts of programs that are never executed. Typically, dead codes are formed when students forget to include an events block (e.g., when the green flag or the sprite are clicked) or when

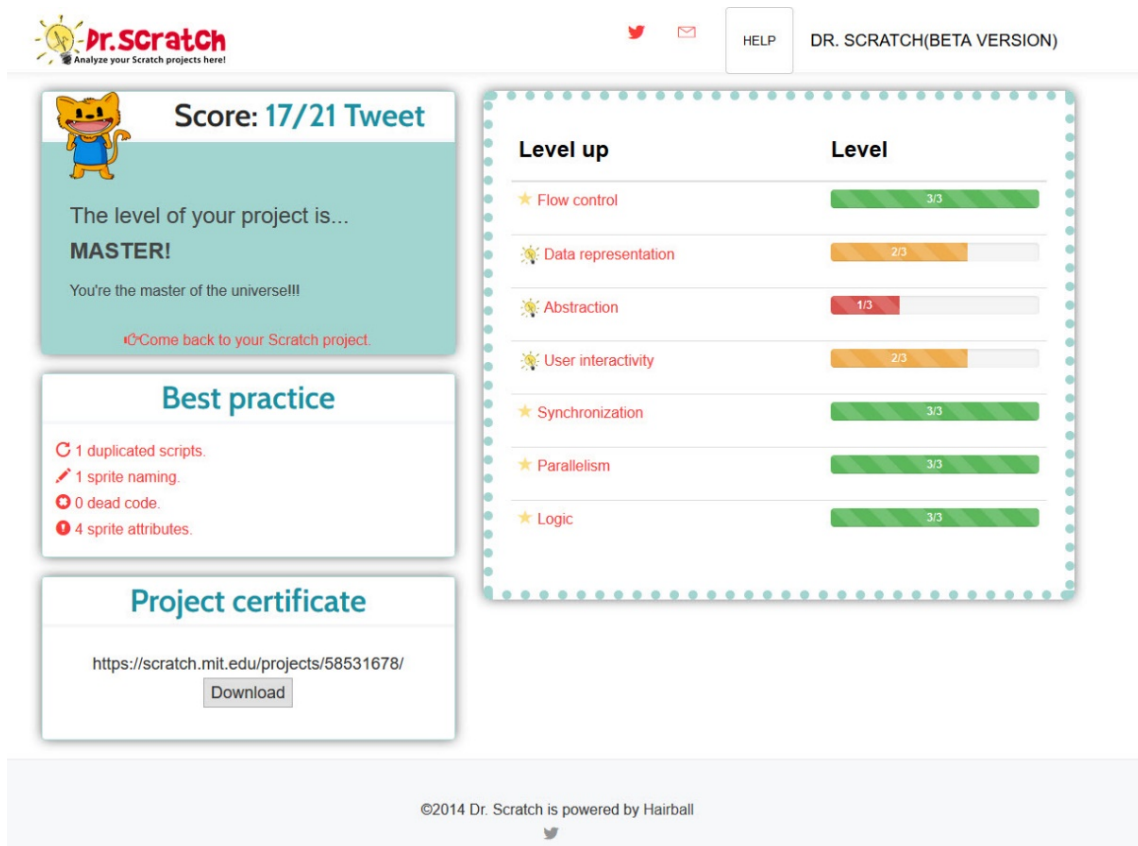


Figure 9. Example dashboard for one eighth-grade student in the master category.

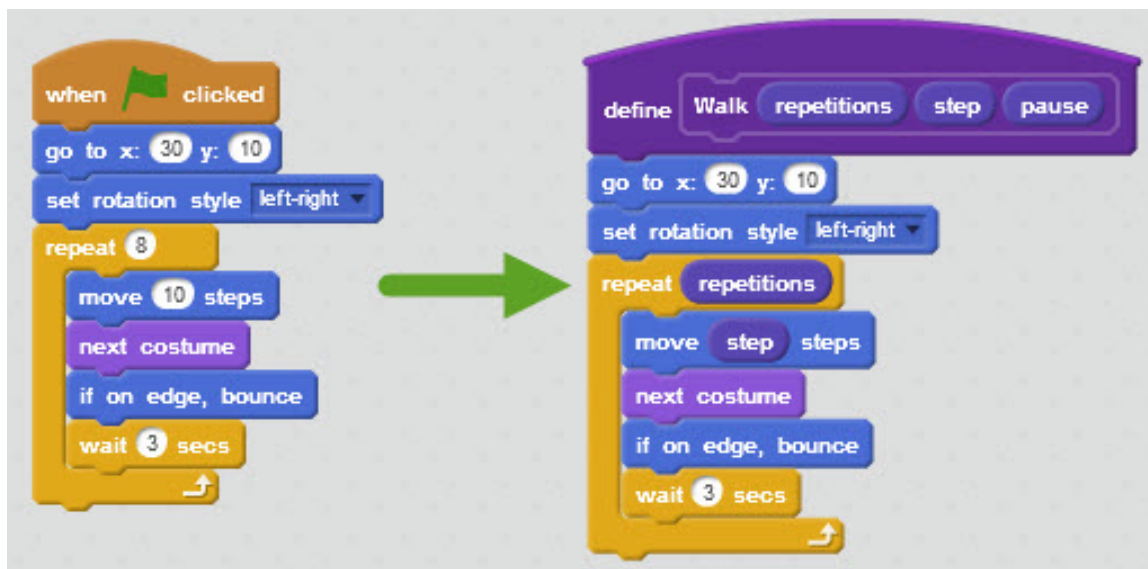


Figure 10. Example of efficient script duplication.

there is a program that is waiting for a message that is never sent (see Figure 11). The presence of a dead code could cause the Scratch project to not work as expected or to not run efficiently.

Summary

Results of the study yielded common CT strengths and weaknesses among the 360 seventh- and eighth-grade students analyzed. An analysis of the students' Scratch projects showed that both groups were strong in synchronization, parallelism, and flow control, and relatively weak in data representation and abstraction. Translated into a day-to-day context, it may be concluded that the students were likely to understand and reason through complex information with which they were presented (synchronization), to focus their thinking process in more than one direction (parallelism), and to create plans that allow them to succeed when presented with both known and unknown events (flow control). The results also indicated that the students were unlikely to be proficient at filtering out what information was necessary to solve a problem (abstraction), and were

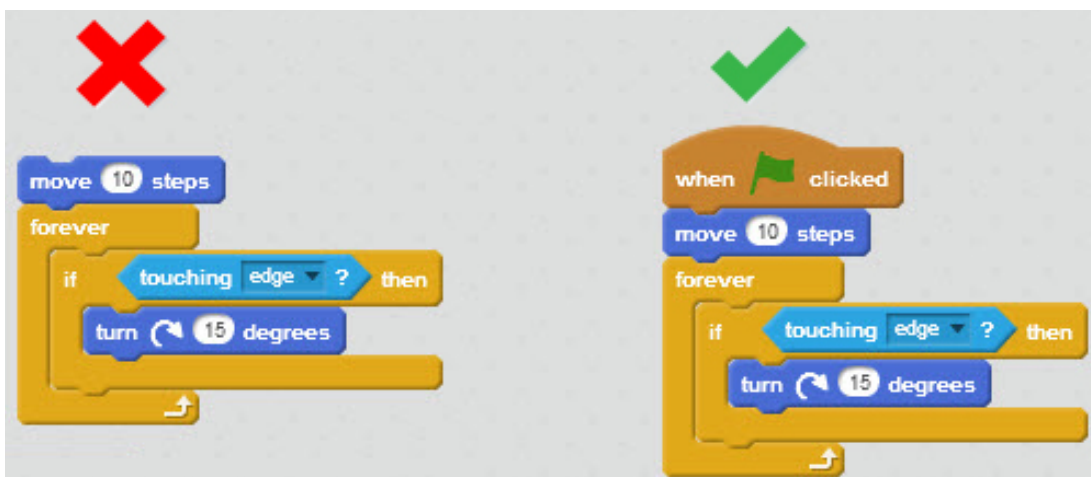


Figure 11. Examples of a “dead code” (left) and correct code initialization (right).

less likely to demonstrate the ability to prioritize in a way that allows them to solve their problems efficiently (data representation).

Affordances and constraints of Dr. Scratch as a tool were also apparent during the data analyses and interpretation process. One of the affordances of Dr. Scratch is that it can help inform the users of their degree of CT development in a particular Scratch project by giving a score and to show a programming profile (i.e., beginner, developing, or master) to the users. This information could become extremely useful for teachers to identify students who might excel or struggle in particular CT skills. Another benefit of Dr. Scratch is that it can help users to identify bad programming habits (i.e., sprite naming, sprite attributes, duplicated scripts, and dead codes). The identification can help teachers to assist their students to become more efficient and better programmers, not just in Scratch but also when using other computer programs in the future.

There are a few constraints of Dr. Scratch. One is that it can only analyze one Scratch project at a time. Currently there is no simple way for teachers to analyze students' Scratch projects as a group. Thus, for teachers to see patterns in students' CT skills strengths and weaknesses, they must analyze their students' Scratch project one by one using Dr. Scratch. A second drawback is that there were several occurrences in which the Scratch project looked great visually, and when it was analyzed by Dr. Scratch, the result showed that the student received a good score on each of the CT components as indicated by a high total score (i.e., 17); however, when I played the game on the Scratch website, I noted that the game did not work as I expected or there were glitches in the game made by the students. As shown in Figure 12, someone else (a user) also commented that the game created by a particular student was not working.

v444

Mouse: Hits 0

Mouse: Level 1

HELP THE MOUSE GET AWAY FROM THE CAT
WHILE AVOIDING THE DOG! AS YOU NAVIGATE
THROUGH THE ROOMS MAKE SURE YOU GRAB
THE FOOD TO GET TO THE NEXT LEVEL!
GOOD LUCK!

PRESS SPACE TO BEGIN

Instructions

Use the arrow keys to move

Notes and Credits

Shared: 13 Mar 2014

★ 1 ♥ 2

Comments (1)

Sorry, comment posting has been turned off for this project.

There are some glitches that need to be fixed.
2 years ago

Figure 12. Example of one eighth-grade student who received a total score of 17 and had Scratch code glitches.

CHAPTER V

DISCUSSION AND CONCLUSIONS

Discussion

The purpose of this pilot study was to explore how Dr. Scratch functions as an assessment for CT. Two research questions were developed in order to assess the effectiveness of Dr. Scratch as a tool to assist programming learners: (a) among the seven CT components used in student's projects, what were common areas of strength and weakness for the seventh- and eighth-grade students? and (b) what were some affordances and constraints of Dr. Scratch as a CT tool? Valuable insights can be gleaned from the results of this study to not only guide practitioners, but also to direct future research. In the next section I provide recommendations for improving students' CT skills, suggestions for instructors implementing CT in the classroom, suggestions for researchers, and conclusions of the study.

Supporting Student CT

The rationale for introducing computing in K-12 in order to improve students' CT skills is indeed compelling (Grover & Pea, 2013; Grover, Pea, & Cooper, 2016; Stephenson, Gal-Ezer, Haberman, & Verno, 2005; Wing, 2006). While the need to introduce CT is currently prioritized at the high school level, there is a growing belief that it should start at an earlier age. Many researchers (e.g., Grover et al., 2016; Tai, Liu, Maltese, & Fan, 2006) have suggested the middle school level as the optimum period to introduce computing. Middle school years are formative and key for cognitive and social

development in the K-12 schooling journey, especially with regard to future engagement with STEM fields (Tai et al., 2006).

This study focused on 360 seventh- and eighth-grade students. Overall, the eighth-grade students performed better in terms of their CT skills than did the seventh-grade students. This may be because many of the eighth-grade students had received Scratch training from the previous school year. The analysis results indicate that many of the seventh- and eighth-grade students particularly excelled in synchronization, parallelism, and flow control, were moderate in logical thinking and user interactivity, but struggled with abstraction and data representation.

Results indicate that students tend to be missing out on two important CT skills—abstraction and data representation, which is the ability to simplify a task and identify what is important and also the ability to solve problems that they are facing more efficiently. These two CT skills are considered by many as possibly two of the most important problem solving skills that each of us needs to have (Barr, Harrison, & Conery, 2011; Lee et al., 2011; Wing, 2008).

Currently there are competing explanations as to why many of the students struggled with these two particular CT skills. One possibility for their difficulty could be due to their age, meaning that they have yet to understand the need to use abstraction and data representation in their daily life, or it could also be because the task that they were given in class did not require the two CT skills (i.e., abstraction and data representation). For this study I used information that was publicly available on the Internet, thus there are limits to the information that I could get to give detailed explanations of these two areas of weakness, such as information about the Scratch instructor and the type of

assignment given in class. Specific information about the students and the project they were given might have helped provide context for the findings.

Furthermore, there are a few suggestions that I will provide to teachers on improving students CT skills. Many researchers' believe that repetitive training, doing the same activity day after day, and getting multiple feedback from teachers are key to students' successful acquisition of knowledge and integration of skills (Bosse et al., 2015; Brookes et al., 2012). Teachers should continue creating similar instructions that can help students retain the CT skills they feel like they have mastered, in this case synchronization, parallelism, and flow control. Furthermore, teachers need to also think about creating instructions that foster a student's logical thinking, user interactivity, and especially in the two CT components where students in this study struggled the most—abstraction and data representation. The first step that could help them improve their instructions in those key areas, is to understand what these components mean. On their website, Dr. Scratch provides a clickable link that allow teachers to see the definitions on each of the CT components and also provides examples on how to improve students' skills on the CT components that students are still currently lacking. Teachers can use this information to create new assignments that consist of abstraction and data representation in the future.

Dr. Scratch as a Tool to Evaluate Scratch

As a relatively new tool to evaluate users' Scratch projects, Dr. Scratch did an impressive job in breaking down Scratch projects into their respective CT components. Even with its impressiveness, the tool is by no means perfect and teachers should not rely

solely on it to evaluate students' CT skills. Findings indicate that Dr. Scratch is especially well-suited to inform users of the degree of their CT development within Scratch by presenting scores from 0 to 3 in each CT component. The scores can also help teachers in identifying students' strengths and weaknesses in particular CT components. Dr. Scratch can also identify bad programming habits in students. The bad programming habits (i.e., sprite naming, attribute initialization, duplicated scripts, and dead codes) can be used to inform users of ways they can improve their Scratch projects, and to guide teachers in helping their students to become more efficient and better programmers.

Despite its strengths, Dr. Scratch has a number of limitations. First, Moreno-León and Robles (2015) mentioned that the tool is currently still in Beta mode; thus, the current Dr. Scratch is unstable. There were several occurrences where Dr. Scratch did not produce evaluation results. Due to its inconsistency, it could be unreliable to evaluate users' Scratch projects. Second, there are no user accounts built into the website. Therefore, students and teachers are unable to keep a log of their Scratch analyses. The log could be useful for the students to look at their progress over time and also for teachers to keep track of their students' progress. Third, as mentioned in the Dr. Scratch paper, the developers were also uncertain whether the use of particular Scratch blocks or a group of blocks was enough to confirm student fluency on certain CT concepts. Dr. Scratch could not measure some of the key CT components that Grover and Pea (2013) described, such as: debugging, efficiency and recursive thinking, and pattern generalization. There was no specific reason why the CT components could not be implemented and measured in the current version of Dr. Scratch. In a future version, Moreno-León and Robles noted that they planned to provide more information on how to

improve each of the aspects where there is room for improvement by the learner. Fourth, Dr. Scratch cannot differentiate between individual and group work. There needs to be a way to indicate in Scratch that students are switching users, so that Dr. Scratch can track who contributed what code. This is not necessarily for fair division of labor, but for a more accurate measure of student CT abilities. Last but not least, Dr. Scratch cannot differentiate between the different Scratch projects, that is, original or a remix.

In Scratch, the term *remixing* refers to the creation of any new version of a Scratch program by adding, removing, or changing the programming blocks, images, or sounds. Nonetheless, one may contend that remixing can be conducive to creating a better product. This stems essentially from “borrowing” from the expertise and know-how of other colleagues and reaping the fruits of their design reflections, which is implicit in their final products. The decisions that go into the design (content, examples, structure) are implicit testimony to the design process. When someone engages in remix, he or she is borrowing not only the content, but also the expertise and the thought process that is embedded in design. Thus, the act of remixing from an existing Scratch project could result in misinterpretation of students’ CT abilities by the teachers. Furthermore, by creating an original Scratch project, teachers will be able to assess individual students CT skills more accurately when the project is analyzed using Dr. Scratch. By having more accurate CT scores, teachers will also be able to tailor a specific instructional plan to the student to address their CT weaknesses.

Even considering its above-mentioned flaws. I would still recommend this tool to K-12 educators because of its intuitive website that is organized and easy to understand through the layout and figures along with useful information to support users in

improving their CT skills. Nonetheless, when using this tool, teachers must be careful in assigning the task (e.g., no remixing allowed) and conservative in evaluating their students' work. By creating an in-depth and well thought out lesson plan, hopefully teachers can effectively help students improve their CT skills. Furthermore, with the development of new, additional tools and the enhancement of Dr. Scratch teachers will be able to better assess each student's strengths and weaknesses in each of the CT components. Hopefully more innovations that could address the aforementioned limitations will result in improvements in CT education and future research.

Conclusions

Scratch is a free web tool that allows teachers to introduce computer programming to K-12 students, and it provides a way to analyze Scratch projects. This allows educators and researchers to automatically assign a CT score to student projects, as well as to detect potential bad programming habits. The aim is to help learners to develop CT skills as well as an interest in CS, and to support educators in evaluating outcomes from their instruction.

The development of CT skills has the potential to improve students' self-efficacy in relation to the field of CS, and to prepare them for greater success in the 21st century workplace. It is anticipated that this research will provide educators, students, and researchers with a better understanding of why CT skills are so important, and specific things they can do to help improve CT. It is also hoped that the findings will support other researchers interested in improving strategies for assessing CT in K-12 schools through the use of learning analytics tools, such as Dr. Scratch.

REFERENCES

- Allan, V., Barr, V., Brylow, D., & Hambruch, S. (2010). Computational thinking in high school courses. *Proceedings of the 41st ACM technical symposium on computer science education* (pp. 390-391). New York, NY: ACM.
- Armoni, M. (2011). The nature of CS in K-12 curricula: The roots of confusion. *ACM Inroads*, 2(4), 19-20.
- Ater-Kranov, A., Bryant, R., Orr, G., Wallace, S., & Zhang, M. (2010). Developing a community definition and teaching modules for computational thinking: Accomplishments and challenges. *Proceedings of the 2010 ACM conference on information technology education* (pp. 143-148). New York, NY: ACM.
- Bagiati, A., Yoon, S. Y., Evangelou, D., & Ngambeki, I. (2010). Engineering curricula in early education: Describing the landscape of open resources. *Early Childhood Research & Practice*, 12(2), 1-15.
- Barendsen, E., Mannila, L., Demo, B., Grgurina, N., Izu, C., Mirolo, C., ... Stupurienė, G. (2015). Concepts in K-9 computer science education. *Proceedings of the 2015 ITiCSE on working group reports* (pp. 85-116). New York, NY: ACM.
- Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, 38(6), 20-23.
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48-54.
- Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). Recognizing computational thinking patterns. *Proceedings of the 42nd ACM technical symposium on computer science education* (pp. 245-250). New York, NY: ACM.
- Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Educational Technology Research and Development*, 59(6), 765-782.
- Bell, S., Frey, T., & Vasserman, E. (2014). Spreading the word: Introducing pre-service teachers to programming in the K12 classroom. *Proceedings of the 45th ACM technical symposium on computer science education* (pp. 187-182). New York, NY: ACM.

- Berland, M., & Lee, V. R. (2011). Collaborative strategic board games as a site for distributed computational thinking. *International Journal of Game-Based Learning*, 1(2), 65-81. doi:10.4018/ijgbl.2011040105
- Boe, B., Hill, C., Len, M., Dreschler, G., Conrad, P., & Franklin, D. (2013). Hairball: Lint-inspired static analysis of scratch projects. *Proceedings of the 44th ACM technical symposium on computer science education* (pp. 215-220). New York, NY: ACM.
- Bosse, H. M., Mohr, J., Buss, B., Krautter, M., Weyrich, P., Herzog, W., ... Nikendei, C. (2015). The benefit of repetitive skills training and frequency of expert feedback in the early acquisition of procedural skills. *BMC Medical Education*, 15(1), 1-10.
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 annual meeting of the American Educational Research Association* (pp. 1-25). Vancouver, Canada.
- Brookes, S. D., Donner, M., Driscoll, J., Mauldin, M., Pausch, R., Scherlis, W. L., & Spector, A. Z. (2012). *The Carnegie-Mellon curriculum for undergraduate computer science*. New York, NY: Springer Science & Business Media.
- Burke, Q., & Kafai, Y. B. (2010). Programming and storytelling: Opportunities for learning about coding & composition. In *Proceedings of the 9th international conference on interaction design and children* (pp. 348-351). New York, NY: ACM.
- Burke, Q., & Kafai, Y. B. (2012). The writers' workshop for youth programmers: Digital storytelling with Scratch in middle school classrooms. *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 433-438). New York, NY: ACM
- Bybee, R. W., & Fuchs, B. (2006). Preparing the 21st century workforce: A new reform in science and technology education. *Journal of Research in Science Teaching*, 43(4), 349-352.
- Chaiken, S., & Ledgerwood, A. (2012). A theory of heuristic and systematic information processing. In P. Van Lange, A. Kruglanski, & E. Higgins (Eds.), *Handbook of theories of social psychology* (vol. 1, pp. 246-267). London, England: Sage. doi: <http://dx.doi.org/10.4135/9781446249215.n13>
- Clark, J., Rogers, M. P., Spradling, C., & Pais, J. (2013). What, no canoes? Lessons learned while hosting a Scratch summer camp. *Journal of Computing Sciences in Colleges*, 28(5), 204-210.

- Close, K., Janisiewicz, P., Brasiel, S., & Martin, T. (2015). What do I do with all this data? How to use the FUN! tool to automatically clean, analyze, and visualize your digital data. *Proceedings from Games and Learning Society 11 conference*, Madison, WI.
- Computer Science Teacher Association. (2003). *CSTA K-12 computer science standards*. Retrieved October from <http://csta.acm.org>
- Computer Science Teacher Association. (2010). *CSTA K-12 computer science standards*. Retrieved from <http://csta.acm.org>
- Computer Science Teacher Association. (2012). *CSTA K-12 computer science standards*. Retrieved from <http://www.csta.acm.org/Curriculum/sub/K12Standards.html>
- Curzon, P., McOwan, P. W., Plant, N., & Meagher, L. R. (2014). Introducing teachers to computational thinking using unplugged storytelling. *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 89-92). New York, NY: ACM.
- DeJarnette, N. (2012). America's children: Providing early exposure to STEM (science, technology, engineering and math) initiatives. *Education*, 133(1), 77-84.
- 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. (2009). The profession of IT beyond computational thinking. *Communications of the ACM*, 52(6), 28-30.
- Dr. Scratch, (n.d.). *Analyze your Scratch projects here!* Retrieved from <http://drscratch.programamos.es/learn/Parallelism/>
- FACT SHEET: President Obama Announces Over \$240 Million in New STEM Commitments at the 2015 White House Science Fair. (2015). Retrieved from <https://www.whitehouse.gov/the-press-office/2015/03/23/fact-sheet-president-obama-announces-over-240-million-new-stem-commitmen>
- FACT SHEET: President Obama Launches New TechHire Initiative. (2016). Retrieved from <https://www.whitehouse.gov/the-press-office/2016/03/09/fact-sheet-white-house-announces-doubling-techhire-communities-and-new>
- Fields, D. A., Giang, M., & Kafai, Y. (2014). Programming in the wild: Trends in youth computational participation in the online scratch community. *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 2-11). New York, NY: ACM.

- Fields, D. A., Searle, K. A., Kafai, Y. B., & Min, H. S. (2012). Debugging to assess student learning in e-textiles. *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 699). New York, NY: ACM.
- Furber, S. (2012). *Shutdown or restart? The way forward for computing in UK schools*. The Royal Society, London, England.
- Goode, J. (2007). If you build teachers, will students come? The role of teachers in broadening computer science learning for urban youth. *Journal of Educational Computing Research*, 36(1), 65-88.
- Goode, J., Margolis, J., & Chapman, G. (2014). Curriculum is not enough: The educational theory and research foundation of the exploring computer science professional development model. *Proceedings of the 45th ACM technical symposium on computer science education* (pp. 493-498). New York, NY: ACM.
- Google for Education (n.d.). What is CT? Retrieved from <https://www.google.com/edu/resources/programs/exploring-computational-thinking/index.html#!what-is-ct>
- Grover, S. (2014). *Foundations for advancing computational thinking: Balanced designs for deeper learning in an online computer science course for middle school students*. Retrieved from <http://purl.stanford.edu/cc869py7832>
- Grover, S., & Pea, R. (2013). Computational thinking in K–12. A review of the state of the field. *Educational Researcher*, 42(1), 38-43.
- Grover, S., Pea, R., & Cooper, S. (2014). Promoting active learning & leveraging dashboards for curriculum assessment in an OpenEdX introductory CS course for middle school. *Proceedings of the first ACM conference on learning@ scale conference* (pp. 205-206). New York, NY: ACM.
- Grover, S., Pea, R., & Cooper, S. (2016). Factors influencing computer science learning in middle school. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 552-557). New York, NY: ACM.
- Guzdial, M. (2011). *A definition of computational thinking from Jeannette Wing*. Computing Education Blog.
- Holdren, J. P., Lander, E. S., & Varmus, H. (2010). *Prepare and inspire: K-12 education in science, technology, engineering, and math (STEM) for America's future. Executive Report*. Washington, DC: President's Council of Advisors on Science and Technology.
- Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, 82, 263-279.

- Kafai, Y. B., & Burke, Q. (2013). The social turn in K-12 programming: Moving from computational thinking to computational participation. *Proceedings of the 44th ACM technical symposium on computer science education* (pp. 603-608). New York, NY: ACM.
- Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37(2), 83-137.
- Kwon, D. Y., Kim, H. S., Shim, J. K., & Lee, W. G. (2012). Algorithmic bricks: A tangible robot programming tool for elementary school students. *IEEE Transactions on Education*, 55(4), 474-479.
- Lamagna, E. A. (2015). Algorithmic thinking unplugged. *Journal of Computing Sciences in Colleges*, 30(6), 45-52.
- Lambert, L., & Guiffre, H. (2009). Computer science outreach in an elementary school. *Journal of Computing Sciences in Colleges*, 24(3), 118-124.
- Lee, I., Martin, F., & Apone, K. (2014). Integrating computational thinking across the K-8 curriculum. *ACM Inroads*, 5(4), 64-71.
- Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, 2(1), 32-37.
- Lin, J. M. C., Yen, L. Y., Yang, M. C., & Chen, C. F. (2005). Teaching computer programming in elementary schools: A pilot study. *Proceedings of the national educational computing conference*. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.9555&rep=rep1&type=pdf>
- 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.
- Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational Thinking in K-9 Education. *Proceedings of the working group reports of the 2014 conference on innovation & technology in computer science education* (pp. 1-29). New York, NY: ACM.
- Margolis, J., Goode, J., & Ryoo, J. (2014). Democratizing computer science knowledge. *Educational Leadership, STEM for All*, 72(4), 48-53.
- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with Scratch. *Computer Science Education*, 23(3), 239-264.

- Mishra, S., Balan, S., Iyer, S., & Murthy, S. (2014). Effect of a 2-week Scratch intervention in CS1 on learners with varying prior knowledge. *Proceedings of the 2014 conference on innovation & technology in computer science education* (pp. 45-50). New York, NY: ACM.
- Moreno, J., & Robles, G. (2014). Automatic detection of bad programming habits in scratch: A preliminary study. *Proceedings of the Frontiers in Education conference (FIE)*, 2014 IEEE (pp. 1-4). New York, NY: IEEE.
- Moreno-León, J., & Robles, G. (2015). Dr. Scratch: A Web tool to automatically evaluate Scratch projects. *Proceedings of the workshop in primary and secondary computing education* (pp. 132-133). New York, NY: ACM.
- National Research Council (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: The National Academies Press.
- Nickerson, H., Brand, C., & Repenning, A. (2015). Grounding computational thinking skill acquisition through contextualized instruction. *Proceedings of the 11th annual international conference on international computing education research* (pp. 207-216). New York, NY: ACM.
- Partovi, H. (2014). *What % of STEM should be computer science?* Retrieved 2015, from <https://www.linkedin.com/pulse/20140618231442-1287-too-much-stem-not-enough-computer-science>
- Preston, J. (2015). *Large companies game H-1B Visa program, costing the U.S. Jobs*. Retrieved from <http://www.nytimes.com/2015/11/11/us/large-companies-game-h-1b-visa-program-leaving-smaller-ones-in-the-cold.html>
- Pulimood, S. M., Pearson, K., & Bates, D. C. (2016). A study on the impact of multidisciplinary collaboration on computational thinking. *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 30-35). New York, NY: ACM.
- Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. *Proceedings of the 41st ACM technical symposium on computer science education* (pp. 265-269). New York, NY: ACM.
- Repenning, A., Webb, D. C., Koh, K. H., Nickerson, H., Miller, S. B., Brand, C., ... Repenning, N. (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*, 15(2), 11.

- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11), 60-67.
- Riversound Media (n.d.). Scrape Tool. Retrieved from <http://happyanalyzing.com/downloads/>
- Settle, A., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., & Wildeman, B. (2012). Infusing computational thinking into the middle-and high-school curriculum. *Proceedings of the 17th ACM annual conference on innovation and technology in computer science education* (pp. 22-27). New York, NY: ACM.
- Siegfried, R. M., Chays, D., & Herbert, K. (2008). Will There Ever Be Consensus on CS1? *Proceedings of the 2008 international conference on frontiers in education: computer science and computer engineering* (pp. 18-23). Semantic Scholar.
- Smith, M. (2016). *Computer science for all*. Retrieved from <https://www.whitehouse.gov/blog/2016/01/30/computer-science-all>
- Stephenson, C., Gal-Ezer, J., Haberman, B., & Verno, A. (2005). *The new educational imperative: Improving high school computer science education*. New York, NY: Computer Science Teachers Association (CSTA). Retrieved from <http://csta.acm.org/Publications/CSTAWhitePaperNC.pdf>
- Tai, R. H., Liu, C. Q., Maltese, A. V., & Fan, X. (2006). Planning early for careers in science. *Life science*, 1, 0-2.
- Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education*, 12(2), 1-29.
- Werner, L., Denner, J., & Campe, S. (2015). Children programming games: A strategy for measuring computational learning. *ACM Transactions on Computing Education*, 14(4), 24.
- Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: Measuring computational thinking in middle school. *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 215-220). New York, NY: ACM.
- Wilson, A., & Moffat, D. C. (2010). Evaluating Scratch to introduce younger schoolchildren to programming. *Proceedings of the 22nd annual psychology of programming interest group* (pp. 1-12). Leganés, Spain: Universidad Carlos III de Madrid.

- Wilson, C., Sudol, L. A., Stephenson, C., & Stehlik, M. (2010). *Running on empty: The failure to teach K-12 computer science in the digital age*. New York, NY: Association for Computing Machinery. Computer Science Teachers Association. Retrieved from <http://www.acm.org/runningonempty/fullreport.pdf>
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35. doi:10.1145/1118178.1118215
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society*, 366, 3717-3725.
- Wing, J. M. (2011). *Computational thinking*. Retrieved from <https://csta.acm.org/Curriculum/sub/CurrFiles/WingCTPrez.pdf>
- Wolz, U., Hallberg, C., & Taylor, B. (2011, March). *Scrape: A tool for visualizing the code of Scratch programs*. Pposter presented at the 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX.
- Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011). Introducing computational thinking in education courses. *Proceedings of the 42nd ACM technical symposium on computer science education* (pp. 465-470). New York, NY: ACM.
- Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)*, 14(1), 1-16.

APPENDICES

Appendix A

Summary of Computational Thinking Average Scores for each Component
from 2013 to 2014 School Year

Table A1

Summary of Computational Thinking Average Scores for each Component from 2013 to 2014 School Year

CT components	Grade 7 ($n = 102$)	Grade 8 ($n = 75$)	All students ($n = 177$)
Flow Control			
CT avg. scores	.44	.52	.48
<i>SD</i>	2.26	2.32	2.29
Data representation			
CT avg. scores	1.24	1.76	1.46
<i>SD</i>	.43	.43	.50
Abstraction			
CT avg. scores	1.03	1.07	1.05
<i>SD</i>	.17	.30	.30
User interactivity			
CT avg. scores	2.00	1.99	1.99
<i>SD</i>	0.00	.12	.11
Synchronization			
CT avg. scores	2.66	2.75	2.69
<i>SD</i>	.65	.52	.60
Parallelism			
CT avg. scores	2.52	2.87	2.67
<i>SD</i>	.78	.38	.66
Logic			
CT avg. scores	1.89	2.35	2.08
<i>SD</i>	.99	.97	1.01

Appendix B

Summary of Computational Thinking Average Scores for each Component
from 2014 to 2015 School Year

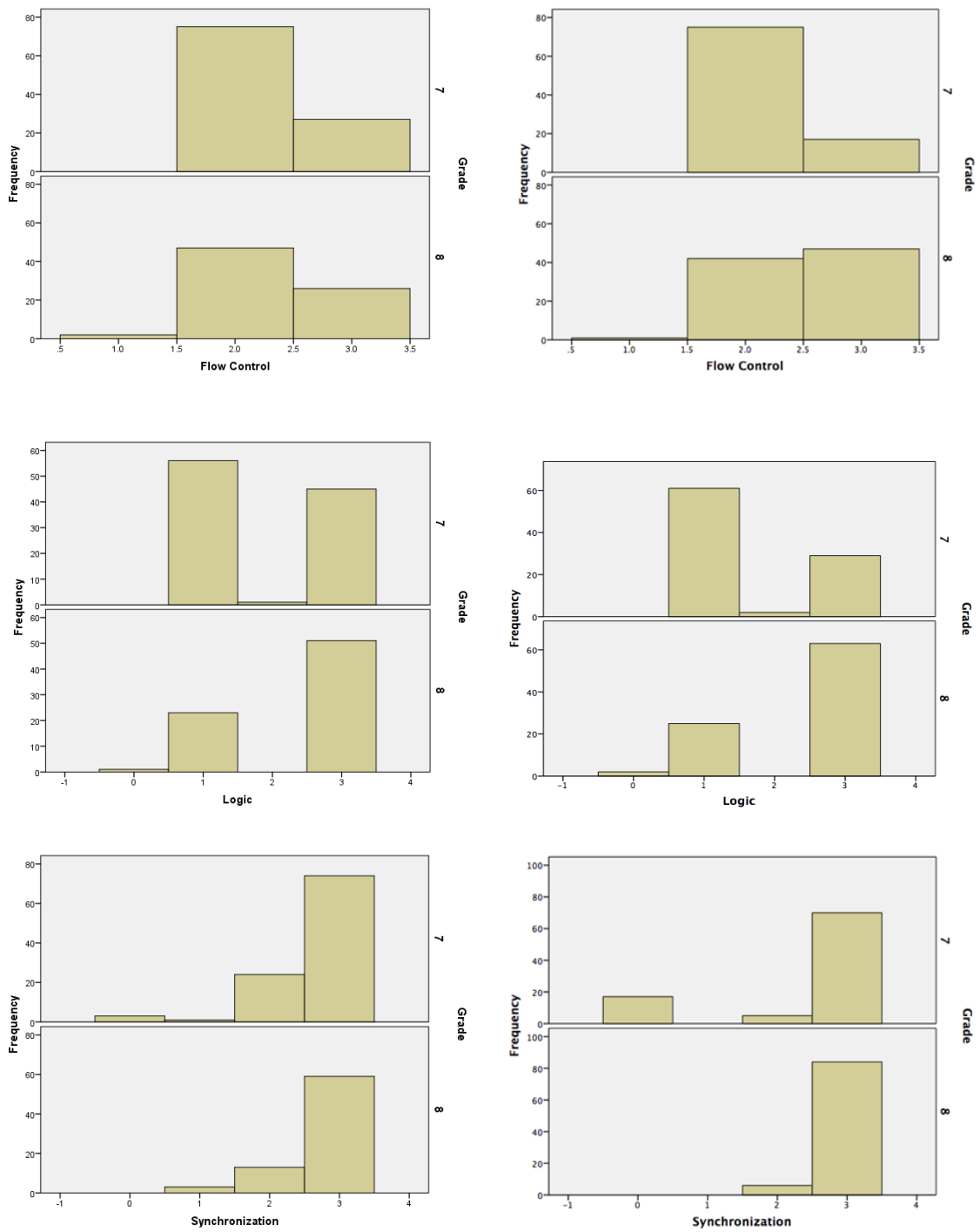
Table B1

Summary of Computational Thinking Average Scores for each Component from 2014 to 2015 School Year

CT components	Grade 7 ($n = 93$)	Grade 8 ($n = 90$)	All students ($n = 183$)
Flow Control			
CT avg. scores	2.19	2.51	2.35
<i>SD</i>	.39	.52	.49
Data representation			
CT avg. scores	1.27	1.78	1.52
<i>SD</i>	.44	.44	.51
Abstraction			
CT avg. scores	1.02	1.08	1.05
<i>SD</i>	.21	.37	.30
User interactivity			
CT avg. scores	2.00	2.00	2.00
<i>SD</i>	0.00	.15	.11
Synchronization			
CT avg. scores	2.40	2.93	2.66
<i>SD</i>	1.17	.25	.89
Parallelism			
CT avg. scores	2.57	2.82	2.69
<i>SD</i>	.73	.53	.65
Logic			
CT avg. scores	1.67	2.38	2.01
<i>SD</i>	.93	.97	1.01

Appendix C

Histograms Comparing Seventh- and Eighth-Grade CT Component Scores from
2013 To 2014 (Left) And 2014 To 2015 (Right) School Years



(figure continues)

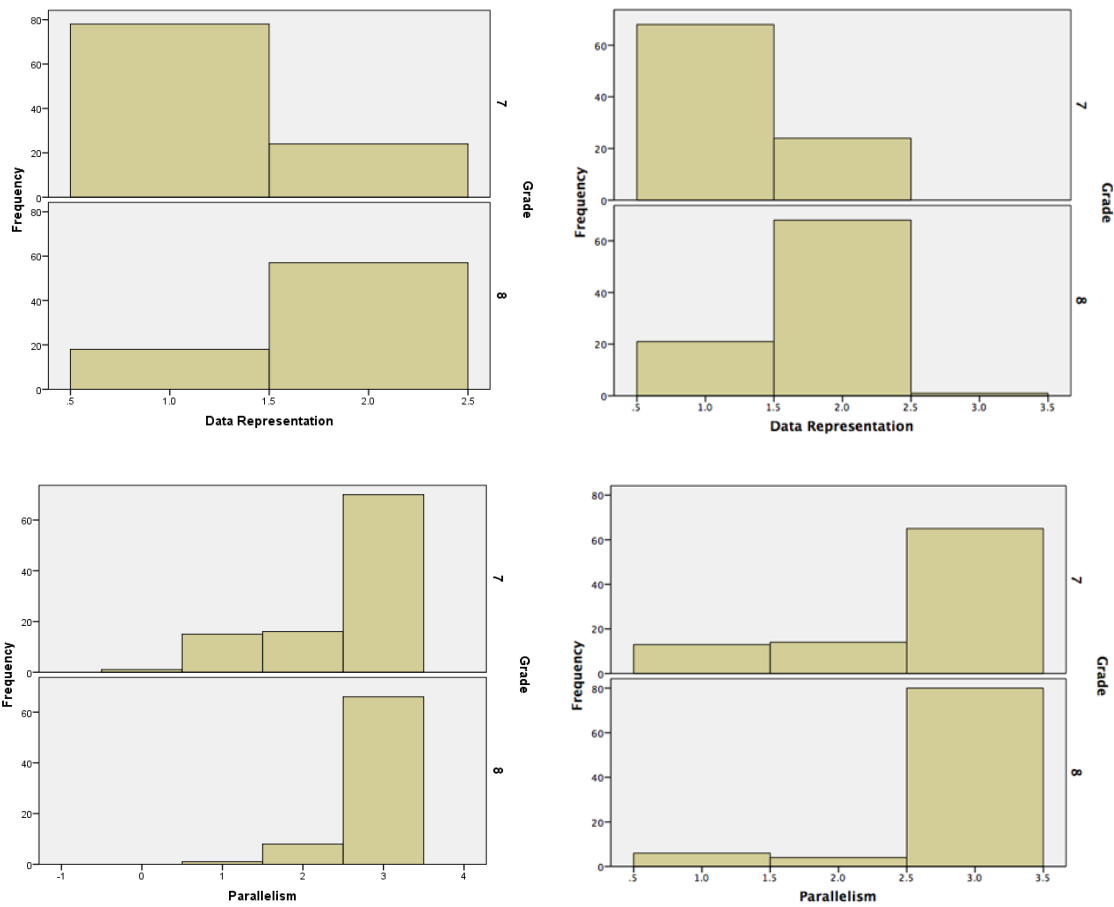


Figure C1. Histograms comparing seventh- and eighth-grade CT component scores from 2013 to 2014 (left) and 2014 to 2015 (right) school years.

Appendix D
IRB Exemption Letter

Institutional Review Board

USU Assurance: FWA#00003308



Request for Determination of Non-human Subjects Research



Approved

FROM:

Melanie Domenech Rodriguez, IRB Chair

Nicole Vouvalis, IRB Administrator

To: Sarah Brasiel, Kevin Lawanto
 Date: October 19, 2015
 Protocol #: 7088
 Title: Exploring Changes In Middle School Students' Computational Thinking

Based on the information provided to USU's IRB, it has been determined that this project does not qualify as human subject research as defined in 45 CFR 46.102(d) and (f) and is not subject to oversight by USU's IRB.

4460 Old Main Hill Logan, UT 84322-4460
 PH: (435) 797-1821 Fax: (435) 797-3769
 WEB: irb.usu.edu EMAIL: irb@usu.edu