



Theses and Dissertations

2017-12-01

Using Dr. Scratch as a Formative Feedback Tool to Assess Computational Thinking

Samuel Frank Browning
Brigham Young University

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>



Part of the [Educational Psychology Commons](#)

BYU ScholarsArchive Citation

Browning, Samuel Frank, "Using Dr. Scratch as a Formative Feedback Tool to Assess Computational Thinking" (2017). *Theses and Dissertations*. 6659.
<https://scholarsarchive.byu.edu/etd/6659>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Using Dr. Scratch as a Formative Feedback Tool to Assess Computational Thinking

Samuel Frank Browning

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Master of Science

Peter J. Rich, Chair
Royce Matthew Kimmons
Ross Allen Andrew Larsen
Gregorio Robles Martinez

Department of Instructional Psychology and Technology
Brigham Young University

Copyright © 2017 Samuel Frank Browning

All Rights Reserved

ABSTRACT

Using Dr. Scratch as a Formative Feedback Tool to Assess Computational Thinking

Samuel Frank Browning
Department of Instructional Psychology and Technology, BYU
Master of Science

Scratch is one of the most popular ways to teach younger children to code in K–8 throughout the U.S. and Europe. Despite its popularity, Scratch lacks a formative feedback tool to inform students and teachers of a student’s progress in coding ability. Dr. Scratch was built to fill this need. This study seeks to answer if using Dr. Scratch as a formative feedback tool accelerates the students’ progress in coding ability and Computational Thinking (CT). Forty-one 4th-6th grade students participated in a 1-hour/week Scratch workshop for nine weeks. We measured pre- and posttest results of the Computational Thinking Test (CTt) between control (n = 18) and treatment groups (n = 23) using three methods: propensity score matching (treatment = .575; control = .607; $p = .696$), information maximum likelihood technique (treatment effect = -.09; $p = .006$), and multiple linear regression. Both groups demonstrated significant increased posttest scores over their pretest (treatment = +8.31%; control = +5.43%), though which group improved the most varied depending on which test was run. We discuss the implications of using Dr. Scratch as a formative feedback tool and recommend further research on the use of such tools in elementary coding experiences.

Keywords: Computational thinking; Scratch; computational thinking test

ACKNOWLEDGEMENTS

I have so many who have supported me through my graduate degree. First and foremost I give great thanks to Peter Rich for the guidance, help, support, and laughs we've had together throughout my degree. I'd like to thank McKay Perkins and Cassidy Vela for their help, ideas, and insights in helping me to collect the data I would need and run the afterschool program. I'd like to thank Ross Larsen for all of his excellent help in making sense of the data I have collected. I'm grateful for friends and family who supported me through my whole degree. I want to thank my wife for the loving support and strength she has been and always will be in my life. I'm grateful for my Heavenly Father who has blessed me more than I can ever comprehend or deserve, and for the loving strength He has given me.

TABLE OF CONTENTS

ABSTRACT.....	ii
ACKNOWLEDGEMENTS.....	iii
TABLE OF CONTENTS.....	iv
LIST OF TABLES.....	vi
LIST OF FIGURES.....	vii
DESCRIPTION OF THESIS STRUCTURE.....	viii
Abstract.....	2
Introduction.....	3
Review of Literature.....	3
Computational Thinking Definitions.....	4
Lack of Measurement of CT.....	5
Methods for Measuring CT.....	6
Measurement as a Feedback Mechanism for Improving Computational Thinking.....	11
Research Questions.....	13
Method.....	14
Participants.....	14
Procedures.....	16
Instruments.....	16
Data Analysis.....	23

Findings	25
Question #1 Results	25
Question #2 Results	30
Discussion	35
Conclusion	38
References	42
APPENDIX A: Step-by-Step Lesson.....	47
APPENDIX B: Annotated Bibliography	49
APPENDIX C: Parental Permission for a Minor.....	68

LIST OF TABLES

Table 1. <i>Participants by Grade and Gender</i>	15
Table 2. <i>Competence Level for Each CT Concept</i>	18
Table 3. <i>Alignment of Computational Concepts Found in CTt with those Measured in Dr. Scratch</i>	21
Table 4. <i>After School Club Schedule</i>	23
Table 5. <i>Propensity Score Matching Frequencies Chart for the Control and Treatment Groups</i>	27
Table 6. <i>Linear Regression Model</i>	29
Table 7. <i>Dr. Scratch Week 4 CT Correlations</i>	35
Table 8. <i>Cluster Centers of Each of the Three Clusters of the Solution</i>	38

LIST OF FIGURES

Figure 1. Dr. Scratch score demonstrating a “Developing” rating.	17
Figure 2. Dr. Scratch Flow control scoring explained.	19
Figure 3. Dr. Scratch score of a “Master” rating.	19
Figure 4. Average pretest and posttest scores on CTt.	26
Figure 5. Mean scores on the “abstraction” construct as measured by Dr. Scratch	31
Figure 6. Mean scores on the “logical thinking” construct as measured by Dr. Scratch.	32
Figure 7. Mean scores on the “flow control” construct as measured by Dr. Scratch	33
Figure 8. Mean scores on the “data representation” construct as measured by Dr. Scratch.	34

DESCRIPTION OF THESIS STRUCTURE

Using Dr. Scratch as a Formative Feedback Tool to Assess Computational Thinking is written in article-format according to guidelines provided by the Department of Instructional Psychology and Technology. I propose submitting this article to either *Computers & Education* (Impact Factor 2.881) or *Computers in Human Behavior* (Impact Factor 2.880). Both journals have been actively publishing research articles on computational thinking and computer science in K-12 contexts over the past five years. As such, either journal appears to be an appropriate target for the content and intended audience of this study. Both journals accept original research manuscripts in the following APA 6th edition style and accept 8000–10000 word articles. Following the article, Appendix A, is an example lesson plan we used to teach both groups during the after school club. Appendix B presents an annotated bibliography, providing an overview of the research literature that informed this study. In Appendix C is the parental consent form used for this study.

Using Dr. Scratch as a Formative Feedback Tool to Assess Computational Thinking

Samuel F. Browning

Brigham Young University

Peter J. Rich

Brigham Young University

Ross Larsen

Brigham Young University

Abstract

Scratch is one of the most popular ways to teach younger children to code in K–8 throughout the U.S. and Europe. Despite its popularity, Scratch lacks a formative feedback tool to inform students and teachers of a student’s progress in coding ability. Dr. Scratch was built to fill this need. This study seeks to answer if using Dr. Scratch as a formative feedback tool accelerates the students’ progress in coding ability and Computational Thinking (CT). Forty-one 4th-6th grade students participated in a 1-hour/week Scratch workshop for nine weeks. We measured pre- and posttest results of the Computational Thinking Test (CTt) between control (n = 18) and treatment groups (n = 23) using three methods: propensity score matching (treatment = .575; control = .607; $p = .696$), information maximum likelihood technique (treatment effect = -.09; $p = .006$), and multiple linear regression. Both groups demonstrated significant increased posttest scores over their pretest (treatment = +8.31%; control = +5.43%), though which group improved the most varied depending on which test was run. We discuss the implications of using Dr. Scratch as a formative feedback tool and recommend further research on the use of such tools in elementary coding experiences.

Keywords: Computational thinking; Scratch; computational thinking test

Introduction

There is a growing need internationally for programmers due to increasing automation and digitization of everyday tasks. One study estimated that “around 47 percent of total US employment is...at risk – *i.e.*, jobs we expect could be automated relatively soon, perhaps over the next decade or two” (Frey & Osborne, 2013, p. 44). In response to this demand, schools around the world have been adopting and even requiring Computer Science (CS) and related concepts—often referred to as computational thinking (CT)—as part of their curriculum. CT and the fundamentals of coding are starting to be introduced in K–12 schools throughout the United States (Elahi, 2016; Grover & Pea, 2013; K–12 Computer Science Framework, 2016; Reppenning, Webb, & Ioannidou, 2010; Rich, Bly, & Leatham, 2014; Smith, 2016). In Europe, coding integration in the curriculum has seen a large adoption rate, “at the national, regional or local level” (Balanskat & Engelhardt, 2015, p. 9), including 16 countries with two more having plans to integrate coding into their core curriculum as of 2015. With so many schools adopting coding at earlier ages, there is a need to better understand more and less effective methods to assess the computing ability of younger students.

Review of Literature

To better situate a discussion on CT in education, it is helpful to first define what is meant by computational thinking, how it is measured, and what efforts have been made to evaluate the effectiveness of these different measurement methods. The following section introduces different attempts to define CT and reviews measures that have been developed to gauge student improvement in coding over time.

Computational Thinking Definitions

Establishing a general definition and an operational definition of CT allows for the creation of assessments to measure student progress in these concepts. One of the first definitions of CT was that CT, “involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” (Wing, 2006, p. 33). Realizing CT’s usefulness for a 21st-century education, the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) worked together with other experts in CS to create for an operational CT framework and definition, which they established as the following:

Computational thinking (CT) is a problem-solving process that includes (but is not limited to) the following characteristics: Formulating problems in a way that enables us to use a computer and other tools to help solve them; logically organizing and analyzing data; representing data through abstractions such as models and simulations; automating solutions through algorithmic thinking (a series of ordered steps); identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources; generalizing and transferring this problem solving process to a wide variety of problems. (CSTA & ISTE, 2011, p. 1)

This operational definition and others led to a more precise consensus on the following CT criteria as defined by Grover and Pea (2013):

Abstractions and pattern generalizations (including models and simulations); Systematic processing of information; Symbol systems and representations; Algorithmic notions of flow of control; Structured problem decomposition (modularizing); Iterative, recursive,

and parallel thinking; Conditional logic; Efficiency and performance constraints;
Debugging and systematic error detection. (pp. 39–40)

The general definition of CT has evolved over time to become more specific in meaning. A K-12 Computer Science Framework examined all past definitions as well as educational standards documents for computing from England and Australia (two countries that have made computing compulsory at every level). Additionally, the framework was reviewed by dozens of collaborators in three rounds of open-feedback, with a panel of experts refining the framework after each open-review session (which each lasted roughly two months). This definition presents computational thinking as, “the thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer” (K–12 Computer Science Framework, 2016, p. 68). Just recently another definition has emerged: “The conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts” (Shute, Sun, & Asbell-Clarke, 2017, p. 1). The goal of this definition was to be actionable by students and therefore assessable. For the purpose of this research, we will use the K–12 Computer Science Framework definition as our guiding framework for understanding and measuring CT ability and improvement over time.

Lack of Measurement of CT

Researchers have highlighted the lack of adequate measurements of CT and have encouraged their creation. Grover and Pea stated in 2013 that “among the most important questions that need answering” before CT is introduced into the curricula is “What . . . can we expect children to know or do better once they’ve been participating in a curriculum designed to develop CT and how can this be evaluated” (p. 42)? Part of what makes CT so difficult to

measure is that CT “concentrates on students performing a thought process, not on the production of artefacts or evidence” (Mueller, Beckett, Hennessey, & Shodiev, 2017, p. 21). One way researchers have addressed this difficulty is by establishing its convergent validity with related constructs. For example, researchers demonstrated that CT and three of the abilities-factors (fluid reasoning, visual processing, and short-term memory) from the Cattell-Horn-Carroll (CHC) model of intelligence (McGrew, 2009; Schneider & McGrew, 2012) were correlated in a recent cognitive study (Ambrosio, Xavier, & Georges, 2014). Others have compared CT to the mental process of problem solving (CSTA & ISTE, 2011; K–12 Computer Science Framework, 2016, p. 69). Thus, while CT may be difficult to operationalize due to its thought-based nature, we may be able to measure it by employing methods used with similar thought-based constructs, such as intelligence and problem-solving.

Methods for Measuring CT

In the following section, we review some of the recent attempts to quantify and describe students’ computational thinking ability, the artifacts examined in doing so, and the instruments developed in the process. The methods demonstrate wide variance on how to measure CT.

Recognizing computational thinking patterns through video. One study sought to measure students’ ability to recognize Computational Thinking Patterns (CTP) through the use of unrelated videos (Basawapatna, Koh, Repenning, Webb, & Marshall, 2011). Researchers ran a summer program teaching middle school teachers and community college students how to code various games, like *Frogger*, *Space Invaders*, *Pacman*, or *Sims*. Participants were then supposed to use the CTP they had learned and apply them in making science simulations. Afterwards, participants were given a test with unrelated examples of the coding patterns they used in the games. Questions were administered online and required participants to watch a short video clip

and then identify through free response what coding patterns they saw in the clip that relate to *Frogger* or another one of the games they created. Basawapatna et al. (2011) asserted the study successfully showed that Computational Thinking Patterns are learned and applicable in related situations. The effective use of video as the medium for presenting unrelated content could be effectively used in K–12 assessments.

Three approaches to recognizing CT: Computational concepts, practices, and perspectives. Brennan and Resnick (2012) developed a framework that is increasingly cited by those measuring CT. By studying the Scratch online community as well as Scratch workshops, Brennan and Resnick (2012) created a framework highlighting three key ways CT manifests in students: “computational concepts (the concepts designers employ as they program),” such as loops, conditionals, operators, data, etc. “computational practices (the practices designers develop as they program),” such as testing and debugging, abstracting and modularizing, etc. “and computational perspectives (the perspectives designers form about the world around them and about themselves)” (p. 3) such as expressing, connecting, questioning. Each of these are described in greater detail below.

Computational concepts. Brennan and Resnick (2012) identified seven computational concepts: (a) sequences (a set of instructions that can be followed sequentially by a computer), (b) loops (repeating the same set of instructions more than once), (c) parallelism (two or more sets of instructions all occurring at the same time), (d) events (“one thing causing another thing to happen” [p. 4]), (e) conditionals (e.g., programming a computer to be able to make decisions based on provided conditions), (f) operators (“support for mathematical, logical, and string expressions, enabling the programmer to perform numeric and string manipulations” [p. 5]), and (g) data (the use of things like variables or lists to store, retrieve, and update values).

Computational practices. The goal of computational practices is to “focus on the process of thinking and learning, moving beyond what you are learning to how you are learning” (Brennan & Resnick, 2012, p. 7). Brennan and Resnick defined “four main sets of practices: being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing” (p. 7).

Computational perspectives. The purpose of computational perspectives was to capture how “young designers describe evolving understandings of themselves, their relationships to others, and the technological world around them” (Brennan & Resnick, 2012, p. 10). Computational perspectives were defined as: Expressing (e.g., computation can be used a form of expression), connecting (e.g., “I can do different things when I have access to others” [p. 10] by creating with others or creating for others), and questioning (e.g., “Questioning involves interrogating the taken-for-granted, and, in some cases, responding to that interrogation through design” [p. 11]).

Assessing the framework. Three approaches were created to assess students’ development within Brennan and Resnick’s (2012) framework. The first approach was a project analysis; its goal was to mainly assess computational concepts. Brennan and Resnick used a Scratch project visualization tool called Scrape (<http://happyanalyzing.com/>) developed out of the College of New Jersey that allowed for a visual representation of every type of coding block used in a project. This associated specific Scratch blocks to specific computational concepts, which Brennan and Resnick used to place visualizations of a student’s projects next to each other to show the progression of code block usage over time. This approach is non-intrusive; it can be done without the student needing to perform any task beyond coding, such as taking a test or being interviewed. Brennan and Resnick acknowledge that this approach does not show if a

Scratch programmer truly understands the coding blocks used in their project, and therefore whether or not the programmer understands the underlying concepts Brennan and Resnick tied to that block.

Artifact-based interviews with a specific one-hour interview protocol was the approach Brennan and Resnick used next. The protocol involved four main areas of discussion: Background in Scratch, project creation, how the programmer participated in the online community, and looking forward to the future. During the interviews Brennan and Resnick found that when some Scratch programmers would copy and paste a chunk of code from another project that worked in a way they wanted, the programmers would understand generally what the chunk of code would do but did not understand what each block within the code specifically accomplished. A greater understanding of individual programmers' abilities was formed as well as what computational concepts they had a proficiency in.

Design scenarios were used as the third approach and consisted of three sets of Scratch projects presented across three interviews in a classroom setting. The students were told each project was something created by another young programmer. Students were then asked to do four tasks: "explain what the selected project does, describe how it could be extended, fix a bug, and remix the project by adding a feature" (p. 19). The tasks revealed a lot of what students did or did not understand and allowed for extensive exploration of students' abilities in real time verses through shared memories the students gave during the artifact-based interviews.

Brennan and Resnick (2012) concluded that the progression of all three assessments was very helpful in getting a more complete picture of students' capabilities and learning as mentioned above, but acknowledged that it is a significant amount of work and time to do all three; as well, Brennan and Resnick state none of their approaches effectively assessed "changes

in computational thinking perspectives” (p. 22). Thus, while description-rich, these approaches make measuring change in computational thinking over time difficult.

A comprehensively validated computational thinking test. The Computational Thinking Test (CTt) was developed in Spain to be a measure of CT concepts for middle-high school students that “[has] undergone a comprehensive psychometric validation process” (Román-González, Pérez-González, & Jiménez-Fernández, 2017, p. 3). The CTt was originally written and validated in Spanish but was translated to British English for use in measuring English students’ CT. Román-González et al. (2017) also sought to better understand CT by seeing how it would relate to “key psychological constructs” (abstract). In the following section, we explain the validation of the CTt and how it seeks to measure middle-high school students’ CT.

Sample and administration of the CTt. The CTt was taken by 1,251 Spanish students across 24 schools in 5th–10th grade (see Román-González, et al., 2017, for distribution by grade, age, and gender). “The CTt is administered collectively and on-line,” and can be taken on either computers or mobile devices (Román-González, et al., 2017, p. 4).

The CTt’s *internal reliability* was measured using Cronbach’s Alpha ($\alpha = 0.793$). Reliability increased in older grades. Researchers found reliability also increased in the 20.0% of the sample who took the CTt on a tablet instead of a computer (Román-González, et al., 2017, p. 9). In an effort to check for *content validity*, CTt was judged by 20 experts with varied experience as K-7 up through university CS teachers, job training CS teachers, app development contest winners, or members of various CS groups. These 20 experts agreed on the CTt’s content validity and narrowed it down from 40 to 28 multiple-choice questions (Román-González, 2015).

Criterion validity was measured by comparing results between the CTt, the Primary Mental Abilities (PMA) battery, and the RP30 problem-solving test (RP30), two well-used measures of related constructs. Each participant took the CTt and then either the PMA or the RP30. The correlation between CTt and PMA measured at ($r = 0.540$; $p < 0.01$) and between CTt and RP30 ($r = 0.669$; $p < 0.01$). Román-González, et al. (2017) conclude that the CTt's comparison to the PMA and RP30 is "powerful evidence of the criterion concurrent validity of [the] CTt" (pp. 5-6, 10).

Measurement as a Feedback Mechanism for Improving Computational Thinking

Every measurement method reviewed to this point has been summative. Measurement for the purpose of summative feedback provides a high-level understanding of a specific group for administrators, researchers, and teachers. However, none of these summative feedback methods help the individual student specifically. Measures can be much more powerful than simple population descriptors. Research shows that performance feedback measures are one of the most effective forms of intervention (Hattie, 2015, Table 1). With the goal of finding an answer to the question "What methods work best in education?," John Hattie (2009) published a study of data compiled from 800 meta-analyses and used it to develop a list of "138 influences that are related to learning outcomes from very positive effects to very negative effects." He "found that the average effect size of all the interventions he studied was 0.40" (Visible Learning, 2016). Hattie used this 'hinge point' as a way to measure how successful an intervention in education could be. As of 2015, Hattie's research data is pulled from nearly 1200 meta-analyses and his list has grown to 195 influencers on student achievement. Feedback consistently ranks as important in John Hattie's meta analyses, currently ranking 15th largest in effect size on student achievement ($ES = 0.73$; Hattie, 2015, Table 1; Visible Learning, 2016).

How feedback relates to CT. Feedback in its broadest definition may be defined as: “the means by which the learner, or any other agent directing the learning process, ascertains whether or not progress is being made toward the end goal, and whether or not the goal has been reached” (Weibell, 2011, p. 361). The goal in measuring progress and providing feedback in CT can be looked at in two ways: (a) the completion of or progress toward solving the problem to which computational thinking processes are applied, or (b) the ability of the student/learner to apply CT processes correctly to any given task/problem. Feedback towards the goal of solving the problem can be given by teachers and peers, or even by the way the student/learner’s artifact(s) (e.g., a program) interact with the problem (e.g., student’s program does not solve or partially solves the given problem). Feedback towards the goal of applying CT processes correctly would most likely require a way to understand how the learner was thinking during the creation process. One way to do this would be to analyze the artifact(s) a student/learner creates and what CT processes would have been needed to create those artifacts.

Dr. Scratch as a formative feedback tool. Now that there is greater consensus on what computational thinking is and there are measures to assess CT among younger students, the question remains, how can we use a reliable and valid measure as a feedback mechanism for improving students’ CT ability? According to a recent study, Scratch (<https://scratch.mit.edu/>) is one of the most popular ways to teach coding in K-8 throughout the world (Rich et al., 2016; de Araujo, Andrade, & Serey-Guerrero, 2016). Despite its popularity, Scratch lacks a formative feedback tool to inform students and teachers of a student’s progress in coding ability. Dr. Scratch was built to fill this need. Dr. Scratch seeks to give feedback on the application of CT processes a student/learner uses to solve a problem. It does this by analyzing a Scratch project’s artifact(s) (e.g., coding blocks, sprites, variables) created by the student/learner in their attempt to

solve their problem. The idea behind Dr. Scratch's analysis is students who correctly use complex programming blocks will most likely have used the CT concepts and processes required to understand the function of those blocks. For example, a student's understanding of data representation would be categorized as follows: one point for using blocks that modify a sprite's attributes such as and orientation & position, two points for using variables, and three points for the use of lists (Moreno-León, Robles, & Román-González, 2015). Dr. Scratch gives a score like this in seven different categories (see Table 2). Dr. Scratch combines the scores from each of the seven categories into one CT score that is shown to the students along with the points given in each section. A study in 2016 found that Dr. Scratch's CT score has "moderate to strong correlations" that are positive and significant with McCabe's Cyclomatic Complexity and Halstead's metrics (Moreno-León, Robles, & Román-González, 2016, p. 1044), common complexity metrics used in software development.

In an effort to test further if Dr. Scratch successfully performs as a formative feedback tool, the purpose of this study is to use Dr. Scratch as a formative measure of students' computational thinking over time.

Research Questions

This study will address the following research questions:

1. Does feedback from Dr. Scratch as a formative feedback tool result in higher scores in computational thinking as measured by the Computational Thinking Test (CTt)?
2. What are the differences in students' created programs as measured by Dr. Scratch when they have used Dr. Scratch as a formative feedback tool and when they have not?

Method

In the following section I will discuss participants, setting, instruments (Dr. Scratch and CTt), and procedures.

Participants

This research took place at a large private university in the western United States. The study consisted of a control and a treatment group. After receiving IRB approval, students were recruited through emails sent to administrators in the three school districts nearest to the university campus. These districts were chosen due to their proximity and the likelihood of parents being able to drive their children to and from campus once/week for an hour-long lesson. Parents/legal guardians were asked to sign a consent form the first day of the afterschool. The goal was to create two groups of 24 students from 5th or 6th grade, split evenly between males and females, if possible. Participants were assigned to either the control or treatment conditions based on which day they signed up for. Students participating in Day A were assigned to the control condition and students in Day B were assigned to the treatment condition. We have found in our experiences teaching CS that boys are typically more interested than girls and that if girls are in a 5:1 ratio of more boys than girls, they tend to drop out because of the lack of other girls; thus we were interested in recruiting a fair number of girls for each group. We were limited to 24 participants in each group due to the size of the computer lab and available computers. Fortunately, we had a near equal number of girls sign up for each day as boys (see Table 1).

Table 1

Participants by Grade and Gender

Gender/Grade	Treatment	Control	Total
Boy Grade 4	3	2	5
Girl Grade 4	4	2	6
Boy Grade 5	6	3	9
Girl Grade 5	4	5	9
Boy Grade 6	2	5	7
Girl Grade 6	3	1	4
Boy Grade 7	1	0	1
Girl Grade 7	0	0	0
Boy Total	12	10	22
Girl Total	11	8	19
Total Participants	23	18	41

Note. Only participants who attended at least 2/3rds of all classes are represented.

We used Propensity Score Matching (PSM) to match students from the Control and Treatment groups based on the following characteristics provided on the form filled out by parents/guardians when they signed up their students for the after school club: grade, gender, father or guardian's highest level of education, mother or guardian's highest level of education, qualify for free or reduced meals, interest in video games, how many hours a week video games are played, Blockly proficiency, exposure to other coding languages, prescore, and post score. We understand that allowing students to self-select into each group reduces randomization and introduces bias into our sample. We attempted to address this bias and reduction in randomization by using PSM and the CTt as a pre/posttest. In our case, we could not force a student to come one day over the other. Using PSM allowed us to find like pairs in both groups and compare how much they improved from pre to post on the CTt.

Procedures

On the first day, each group took the CTt prior to any coding instruction (which began on week 2). Both groups were presented with the same curriculum (see section “The after school club” for a detailed explanation of the curriculum), provided by the same teacher(s), and faced the same coding challenges and projects. The only difference between groups was the use of Dr. Scratch by the treatment group students as a method of self-evaluation throughout the course. At the end of each lesson, after participants left for the day, researchers ran all students’ projects for that day through Dr. Scratch as a way to track progress over time on the same tasks.

Treatment group. The treatment group was introduced to the analysis tool Dr. Scratch as a way to receive feedback while they code. Students were taught in week 2 how to use Dr. Scratch as a way to measure their computational thinking skills. Instructors then referred to it frequently and encouraged its use in class. The same 2–3 instructors were present for each lesson.

Control group. Dr. Scratch was not taught to control students or mentioned at all. Unknown to students, at the end of each day, students’ projects were analyzed in Dr. Scratch as a way to track their progress and compare scores on the same assignments with the treatment group. Students received the same lessons as the treatment group. In both cases (control and treatment), the lessons were designed to teach programming fundamentals.

Instruments

The following section will explain the instruments used in the study, how the CTt was administered, and how the after school club was run.

Computational Thinking Test. The Computational Thinking Test (CTt) is a 28 multiple-choice question test designed for middle and high school students. Twenty experts judged the CTt test and agreed on its content validity (Román-González, 2015). The CTt’s internal reliability was measured using Cronbach’s Alpha ($\alpha = 0.793$; Román-González, et al., 2017). Criterion validity was measured by comparing results between the CTt, the Primary Mental Abilities (PMA) battery, and the RP30 problem-solving test (Román-González, et al., 2017). “The CTt is administered collectively and on-line, and it can be performed both via non-mobile or mobile electronic devices” (Román-González, et al., 2017, p. 4). The CTt was originally written and validated in Spanish. Researchers shared a British English version with us, which we used with American students. Cases of confusion over cultural phrasing were explained to students upon request (e.g., “how do you get along with computers”).

Dr. Scratch. Dr. Scratch is a free tool found at <http://drscratch.org>. Scratch projects can be analyzed by url or by uploading a Scratch project file.

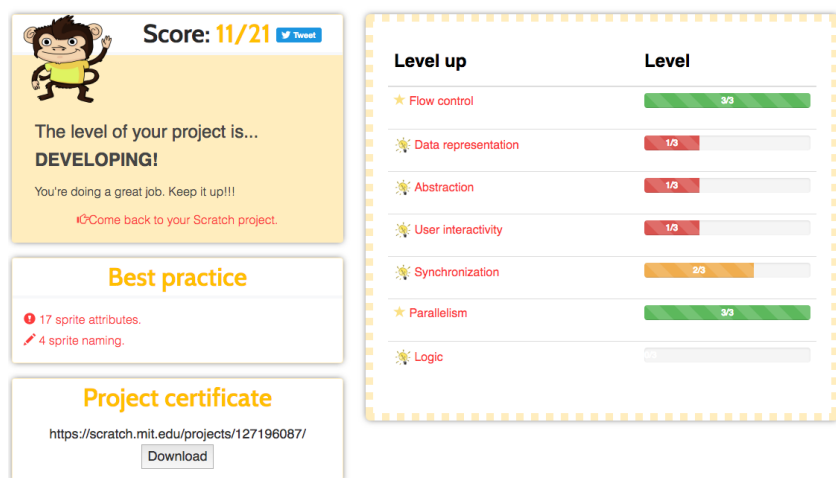


Figure 1. Dr. Scratch score demonstrating a “Developing” rating.

Projects are analyzed according to seven criteria that align with the K-12 CS framework: Flow control, data representation, abstraction, user interactivity, synchronization, parallelism, and

logical thinking. Each category is worth three points, for a total of 21 possible points (see Figure 1). Points are assigned by the use of specific coding blocks as seen in Table 2 found in Moreno-León, et al. (2015, p. 6).

Table 2

Competence Level for Each CT concept (Moreno-León, et al., 2015)

CT Concept	Competence Level			
	Null (0)	Basic (1 point)	Developing (2 points)	Proficiency (3 points)
Abstraction and problem decomposition	-	More than one script and more than one sprite	Definition of blocks	Use of clones
Parallelism	-	Two scripts on green flag	Two scripts on key pressed, two scripts on sprite clicked on the same sprite	Two scripts on when I receive message, create clone, two scripts when %s is > %s, two scripts on when backdrop change to
Logical thinking	-	If	If else	Logic operations
Synchronization	-	Wait	Broadcast, when I receive message, stop all, stop program, stop programs sprite	Wait until, when backdrop change to, broadcast and wait
Flow control	-	Sequence of blocks	Repeat, forever	Repeat until
User interactivity	-	Green flag	Key pressed, sprite clicked, ask and wait, mouse blocks	When %s is >%s, video, audio
Data representation	-	Modifiers of sprites properties	Operations on variables	Operations on lists

When a project is analyzed, its total score is given as well as a mastery rating (i.e., “basic,” “developing,” or “master”). Individual scores in each category are also given.

Pr.Scratch
Analyze your Scratch projects here!

DR. SCRATCH(BETA VERSION)

Flow control

Logic | Data representation | Parallelism | Synchronization | User interactivity | Flow control | Abstraction

Duplicated scripts | Incorrect names | Dead code | Attribute initialization

Instructions related to algorithmic notions of flow control can help you to **control the behavior of your characters**, making, for example, certain blocks that are repeated a number of times, or until a situation arises.

If you get 0 points...

The most basic way to control the behavior of your characters is creating a **program composed of a set of blocks are executed one after another**, as we see in the picture:

```

when clicked
say I'm a dancer cat for 2 secs
move 10 steps
turn 15 degrees
say (3.3 secs)
turn 15 degrees
move 10 steps
  
```

How does this program work? When the user click on the green flag run behind one another all blocks that have been included in the program. Start with the first block 'say I'm a cat dancing for 2 seconds'; then run 'move 10 steps' then 'turn 15 degrees right'; and so on to the last block of the program.

Figure 2. Dr. Scratch Flow control scoring explained.

Each category explains to the user how points are earned (e.g., by clicking on “Flow control” a user is taken to a separate web page and can read explanations of how to improve as well as see example code; see Figure 2)

Score: 16/21 [Tweet](#)

The level of your project is...
MASTER!
You're the master of the universe!!!
[Come back to your Scratch project.](#)

Best practice

- 0 duplicated scripts.
- 0 sprite naming.
- 0 dead code.
- 3 sprite attributes.

Project certificate
<https://scratch.mit.edu/projects/129385222/>
[Download](#)

Level up	Level
★ Flow control	3/3
★ Data representation	2/3
★ Abstraction	1/3
★ User interactivity	2/3
★ Synchronization	2/3
★ Parallelism	3/3
★ Logic	3/3

Figure 3. Dr. Scratch score of a “Master” rating.

Coding best practices (e.g., dead code, sprite attributes, sprite naming, variable naming; see Figure 3) are also checked for and reported in a “Best practice” section. Best practices do not

contribute to a project's total score. Dr. Scratch provides a snapshot of where a project scores at the time the snapshot was taken. It is possible to take as frequent snapshots as a user would like of their project to see how their project develops (Moreno-León, et al., 2015). As such, it is possible for a student to use Dr. Scratch as a formative measure for identifying the complexity of their project and for guidance on how to make it more advanced.

In the treatment group, we would teach students about the project they would be doing that day. As we walked through the daily project, we showed students the Dr. Scratch score they should try and get. For example, in the “step-by-step” lesson students were asked to try to earn at least one point in flow control, data representation, and user interactivity. We encouraged treatment students while they coded to check their Dr. Scratch Score to see if they had reached their goal and then discussed it with them.

How Dr. Scratch relates to the CTt. The correlating CT concepts between the CTt & Dr. Scratch are shown in Table 3 below. The numbers shown are the points a Scratch project would receive from Dr. Scratch for using the coding blocks tied to the CT concepts tested in the CTt. Scratch does not have a *while* conditional block, and the CTt has a section that tests the while conditional. The while conditional is similar enough to a *repeat until* conditional that we decided it aligned with the flow control category like a “repeat until” conditional.

Table 3

Alignment of Computational Concepts Found in CTt with those Measured in Dr. Scratch

CTt Computational concepts addressed	Dr. Scratch CT Concepts			
	Abstraction	Logical Thinking	Flow Control	Data Representation
Basic directions and sequences (4 items)			1	1
Loops–repeat times (4 items)			2	
Loops–repeat until (4 items)			3	
If–simple conditional (4 items)		1		
If/else–complex conditional (4 items)		2		
While conditional (4 items)			(3)	
Simple functions (4 items)	2			

The CTt had nothing we could find that correlated strongly with Dr. Scratch’s parallelism, synchronization, and user interactivity categories (see Tables 2–3). Therefore, we only report on the correlation of the sub constructs measured by both assessments.

Administration of CTt. On the first day of the after-school experience, participants in both groups took the CTt as a pre-assessment. Inasmuch as the test is administered online, each student was told to pick a computer to sit at in the computer lab where the test was already loaded in full-screen mode. Students completed the test on their own without help from classmates or teachers, except for clarifications on language or instructions. There was one parent who tried to help their student on the test. They were asked to stop and the student ended up dropping out of the study. Students were not told what they scored or how well they did. The test was never referred to again during the duration of after school club. On the final day of the

after school club, students took the CTt again as a posttest. Similar to how the pretest was set up, students were asked to pick a computer in the lab to take the test on, with the CTt loaded in full-screen on each machine. We told students the posttest would have no effect on them but that we would appreciate it if they tried to do their best. At the end of the posttest, we let each student pick a large candy bar. Students could see there was more candy than students, so they did not feel like they needed to hurry through the test to get a candy bar of their choice. We chose to follow recommendations for doing a pretest and then a posttest to check if an intervention has caused an increase in skill or cognitive ability in a subject area; namely, to do the posttest between 6–12 weeks after the pretest. Reasoning for doing it within that timeframe are if a student performs the posttest too close to the pretest then students may recall of the answers they put on the pretest while they take the posttest, and to not wait more than 12 weeks to give the posttest to avoid change in a student's score due to maturation (Brown, Irving, & Keegan, 2014).

The after school club. Lessons for the after school club were based on the *Creative Computing Curriculum Guide* (CCCG) by Brennan, Balch, & Chung (2011). The after school club began each day with an opening challenge that required the use of CT concepts to solve (e.g., debugging activities, logic activities). Then there was a short lesson to teach concepts based on the seven Dr. Scratch criteria (See Table 4 below). During weeks 1–4 (during week 1 they did the Scratch Surprise lesson with the remaining time after they finished the CTt) students completed projects from the CCCG. In week 5, students were given the challenge to make a Choose Your Own Adventure (CYOA) and spent this day creating the story on paper. Students worked on coding this single project during the remaining three weeks.

Table 4

After School Club Curriculum Schedule

Week	Lessons ^a	Concepts Taught
Week 1	Pre CTt, Scratch Surprise	
Week 2	Step-by-step	Basic Directions and Sequences
Week 3	It's Alive!	Loops/Repeat Blocks
Week 4	Cloning	Costumes and Speech
Week 5	CYOA Work ^b	If/then; If/Else
Week 6	CYOA Work	Operations
Week 7	CYOA Work	Function Blocks
Week 8	CYOA Work	
Week 9	Post CTt	

^a Using Brennan, Balch, & Chung, 2011

^b Participants used Week 5 to create the plan for their CYOA on paper, they did not code on the computers.

Data Analysis

In this section we will discuss our use of the following techniques to answer our first question: Post-Propensity Score Matching (PSM), Information Maximum Likelihood Technique (FIML), and multiple linear regression. To answer our second question we used a Latent Growth Curve Model (LGCM).

Question #1 data analysis. To answer the question “Does feedback from Dr. Scratch as a formative feedback tool result in higher scores in computational thinking as measured by the Computational Thinking Test (CTt)?” we used Post-Propensity Score Matching (PSM). This method is used to match like pairs from both groups after the fact on observable characteristics (see Table 5). If like pairs can be found, it allows for a comparison with good statistical power, which would normally be lost due to participants not being randomly assigned to each group.

Participants in our study self-selected the group they participated in according to the availability in their schedules. Each student pair had a PSM score within 0.2 of their match. If a student had no match in the other group with a PSM score within 0.2 then their data was thrown out. This left us with 20 students after the PSM comparison (nine students in treatment, 11 in control).

With a sample size below 30 ($p = .696$), we decided to look at our missing data (we had eight students in each group who took the pretest and not the posttest) in M-Plus using the Full Information Maximum Likelihood Technique (FIML; Little & Rubin, 2002). Instead of imputing missing data, this method “estimates parameters directly using all the information that is already contained in the incomplete data set” (Dong & Peng, 2013, p. 8). Finally we ran our data through a Multiple Linear Regression model as a further way to check if we were accounting for the missingness of the data accurately in our FIML (see the Findings & Discussion below).

Question #2 data analysis. To answer the question “What are the differences in students’ created programs as measured by Dr. Scratch when they have used Dr. Scratch as a formative feedback tool and when they have not?” we ran a Dr. Scratch Analysis on all participants’ projects in both the treatment and control group at the end of each day we met. We used a statistical approach from the structural equation modeling (SEM) framework called Latent Growth Curve Modeling to look at our repeated measure. The Latent Growth Curve Model (LGCM) “attempt[s] to estimate between-person differences in within-person change” (Curran, Obeidat, & Losardo, 2010, p. 122). The purpose of the LGCM of the students’ programs is to show growth over time of both individuals and the group as a whole in scores received by Dr. Scratch. The LGCM allowed us to dig in further and map out students’ programs’ growth in each CT concept individually as defined by Dr. Scratch (e.g., students’ overall growth in abstraction or logic). It is important to note that Dr. Scratch only analyzes the code existing inside students’

Scratch projects at the time the analysis takes place. This means that the growth shown in a LGCM will show us what evidence of CT processes exist in a student's project at the times it was analyzed.

Findings

In this section we will discuss the findings for both of our research questions.

Question #1 Results

Initial descriptive statistics help us to understand how each group performed on the CTt pre/post measure as well as weekly Dr. Scratch scores on individual projects (see Figures 4–8 below). Due to differing circumstances, eight

students in each group were not present on the final day, therefore affecting overall scores in both groups. Figure 4 shows initial raw CTt scores for both the treatment and control groups, as well as the sub-group of students who completed both tests. Pretest scores indicated no statistical difference between groups (Treatment = 57.68% or Mean = 16.15 vs. Control = 57.59% or Mean = 16.13), suggesting the groups were equivalent to begin with. On an intriguing side note, potentially because of interest in the programming and self-selection into the program, the groups in our study obtained scores more closely matching 7th & 8th grades in the initial study (Mean = 16.24 or 58%) than their own age group of 5th & 6th grades (Mean = 13.76 or 49.14%) as measured in previous studies (Román-González, et al., 2017, Table 4).

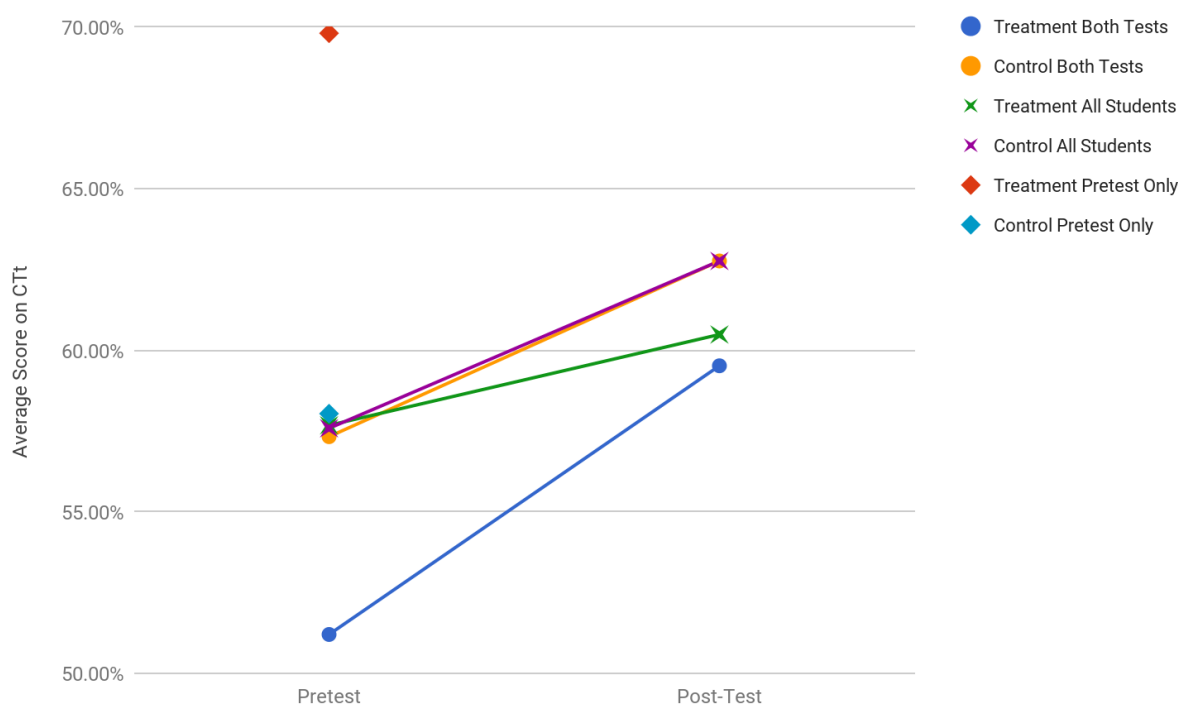


Figure 4. Average pretest and posttest scores on CTt. *Note:* “Treatment all Students” would have had the same posttest score as "Treatment Both Tests" but there was one student in the treatment group who took the posttest without taking the pretest.

After conducting a post PSM using the pretest, we compared how students in both groups performed on the CTt posttest (see Table 5). Only 20 students remained after the PSM comparison (9 in the treatment group; 11 in control) students were matched with an equivalent pair within .2 of a standard deviation of the propensity score (treatment = .575; control = .607; $p = .696$). This showed that our first prediction was met—both groups demonstrated significant increased posttest scores over their pretest.

Table 5

Propensity Score Matching Frequencies Chart for the Control and Treatment Groups

	Control Group						
	N		Mean	Median	Std. Deviation	Minimum	Maximum
	Valid	Missing					
Grade	24	0	5	5	0.722	4	6
Father/Guardian's Highest Level of Education	24	0	3.958	4	1.627	1	6
Mother/Guardian's Highest Level of Education	24	0	3.583	4	1.059	2	7
Qualify for Free or Reduced Meals	24	0	1.666	2	0.481	1	2
Interest in Video Games	24	0	4.5	6	1.865	1	6
How many Hours a week does the student play video games	24	0	1.291	1	0.550	0	2
Blockly Proficiency	24	0	0.083	0	0.408	0	2
Exposure to other programming languages	24	0	0.375	0	0.494	0	1
Pretest Score	22	2	0.575	0.607	0.187	0.19	0.86
Posttest Score	14	10	0.627	0.660	0.169	0.33	0.93
Boy(1: Boy 0: Girl)	24	0	0.583	1	0.503	0	1
	Treatment Group						
	N		Mean	Median	Std. Deviation	Minimum	Maximum
	Valid	Missing					
Grade	26	0	4.961	5	0.870	4	7
Father/Guardian's Highest Level of Education	26	0	4.538	5	1.207	2	7
Mother/Guardian's Highest Level of Education	26	0	4.269	4	1.218	2	7
Qualify for Free or Reduced Meals	26	0	1.769	2	0.429	1	2
Interest in Video Games	26	0	4	3	1.811	1	6
How many Hours a week does the student play video games	26	0	1.230	1	0.651	0	3
Blockly Proficiency	26	0	0.538	0	1.028	0	4
Exposure to other programming languages	26	0	0.423	0	0.503	0	1
Pretest Score	23	3	0.576	0.5	0.179	0.33	0.93
Posttest Score	16	10	0.604	0.607	0.178	0.36	0.89
Boy (1: Boy 0: Girl)	26	0	0.538	1	0.508	0	1

We decided to look into how attendance may have affected each group's score. When looking at the 16 students (8 from each group) who missed the final CTt compared to the students who took both tests in the same groups, we found the treatment group students who took just the pretest scored an average of +18.59% higher than those in the treatment group who took both tests! By contrast, control group students who only took the pretest scored similarly to those in the control group who took both tests (+0.71%; see Figure 4). Thus, there was a very stark difference between the effect of attrition in the treatment group vs. the control group. We were unable to find any reason for why the difference was so stark. We suspect that attrition was so high due to the date for the posttest coinciding with most of the students' last week of school before summer. With those in the treatment group who only took the pretest having such a high average pretest score, the score of the students in the treatment group who took both the pretest and the posttest had a much lower pretest average (-6.12%) than the control group. When only looking at students who took both tests, the treatment group had a higher improvement from their initial score (+8.31%) than the control group did from theirs (+5.43%).

We ran our results in M-Plus to account for our missing data through FIML (Little & Rubin, 2002). This gave us a significant negative treatment effect of = -.09 ($p = .006$). If our FIML accurately accounted for the missing data, then the treatment was detrimental, though, when we ran the data through FIML using just those who completed both tests, the significance went away ($p = .157$; $B = -.088$).

Next, we ran a linear regression model to check to see if we were accounting for the missingness of the data well in our FIML. The assumptions in multiple linear regression are: linearity, independence, normality, equality of variance, and lack of multicollinearity. We checked these diagnostic methods with a variety of diagnostic methods including: (a) residual

plots, (b) histograms of the residuals, and (c) the variance inflation factors (VIFs). No assumption of violations were noted except for a high VIF in Scratch Proficiency, which was thus eliminated and those results are reported (see Table 6).

Table 6

Linear Regression Model

Factor	B	S.E.	Two-Tailed P-Value	β
Treatment	-0.09	0.033	0.006	-0.221
Gender (Boy)	-0.097	0.046	0.033	-0.237
Grade	0.007	0.033	0.824	0.028
Father/Guardian's Highest Level of Education	-0.018	0.022	0.419	-0.123
Mother/Guardian's Highest Level of Education	0.107	0.012	0	0.613
Qualify for Free or Reduced Meals	-0.073	0.036	0.04	-0.161
Interest in Video Games	0.006	0.017	0.714	0.055
How many Hours a week does the student play video games	-0.065	0.04	0.1	-0.19
Blockly Proficiency	0.049	0.018	0.007	0.195
Exposure to other programming languages	0.029	0.038	0.452	0.069
Pre Score	0.734	0.148	0	0.637

Having accounted for missing data with FIML and the treatment showing up as statistically significant with a negative effect, we came to the following conclusions: (1) We may not understand the missing data mechanism for the treatment and control groups who only took the pretest. This is a weak claim as it looks like we are predicting the missingness well in our linear regression models (see Table 6). We are more inclined to believe the PSM results, because the students were matched with an equivalent pair within .2 of a standard deviation of the propensity score ($p = .696$). But a limitation is small sample size, reducing statistical power.

(2) Our groups seem to be heavily affected by selection bias, which weakens any causal claim we can make with this experiment. It could be there was just a different type of student in the treatment group. (3) It is a possibility that using Dr. Scratch as a formative feedback tool does not increase students' CT ability as measured by the CTt, but it may increase other skills as measured by Dr. Scratch.

On the other hand, when we compare scores for those for whom we have the full data, there is a statistically significant result in the other direction, suggesting that the treatment did indeed have a positive effect, which may have been evident had attrition not occurred. Had the scores from the high performers in the treatment group been included, the rate of increase may have also been smaller (2/8 of these students scored ~93% to begin with, which means there was less room for overall growth than the other students). Due to this ambiguity between tests, we cannot determine with certainty that using Dr. Scratch as a formative feedback tool either did or did not positively affect students' CT as measured by the CTt. Further research would be required in this area. Our exploration of similar Scratch projects using Dr. Scratch may help to clarify the difference that using it as a feedback tool may have had.

Question #2 Results

To answer our second question, we decided to run a LGCM on the four Dr. Scratch CT categories that correlate with the CTt (see Table 3, Figures 5–8). Both groups worked on the same one-lesson projects from the CCCG for the first four weeks of lessons (weeks 1–4). After that, students were given a 4-week Choose Your Own Adventure (CYOA) project to work on for the remaining weeks. These projects varied in scope enough that we decided it would be best to make a comparison on the one-lesson CCCG projects and not the CYOA projects. In Figures 5–8 you can see a large drop off in the graph between week 4 and week 6 in all scores, which is

because students spent time planning and creating their own stories to use for the CYOA project during week 5 and did not code on the computers. During week 6 they started working on their CYOA projects (including time spent creating sprites, backdrops, sounds, etc.). Week 8 is included to show truncatedly where their CYOA projects' scores ended on the last day they had to work on their projects. Few students completed their projects. Only data from weeks 1–4 are used in the following analysis.

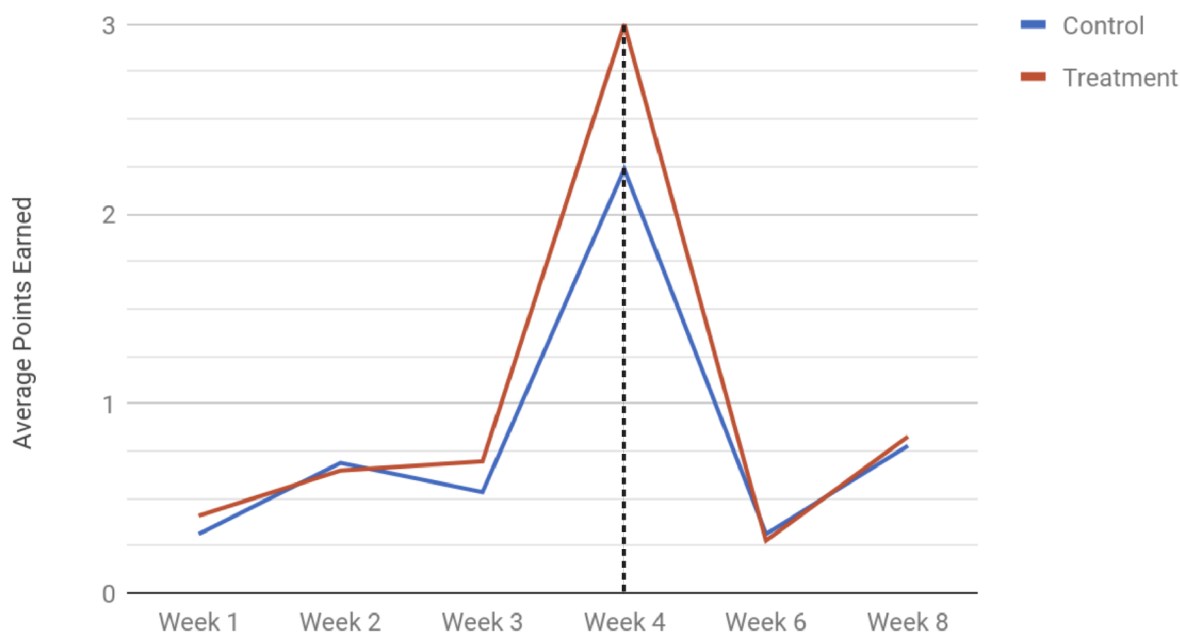


Figure 5. Mean scores on the “abstraction” construct as measured by Dr. Scratch. *Note:* The LGCM only considered weeks 1–4.

Dr. Scratch’s measure of abstraction showed a .75 increase in the treatment group more than the control group, which was statistically significant ($p = .0021$; see Figure 5). We were surprised to note that in week 4 all students in the treatment group scored three points in abstraction, which is the highest score given by Dr. Scratch.

In contrast, control group students scored a mean of 2.25, a respectable but lower average. Both groups were presented with the same lesson.

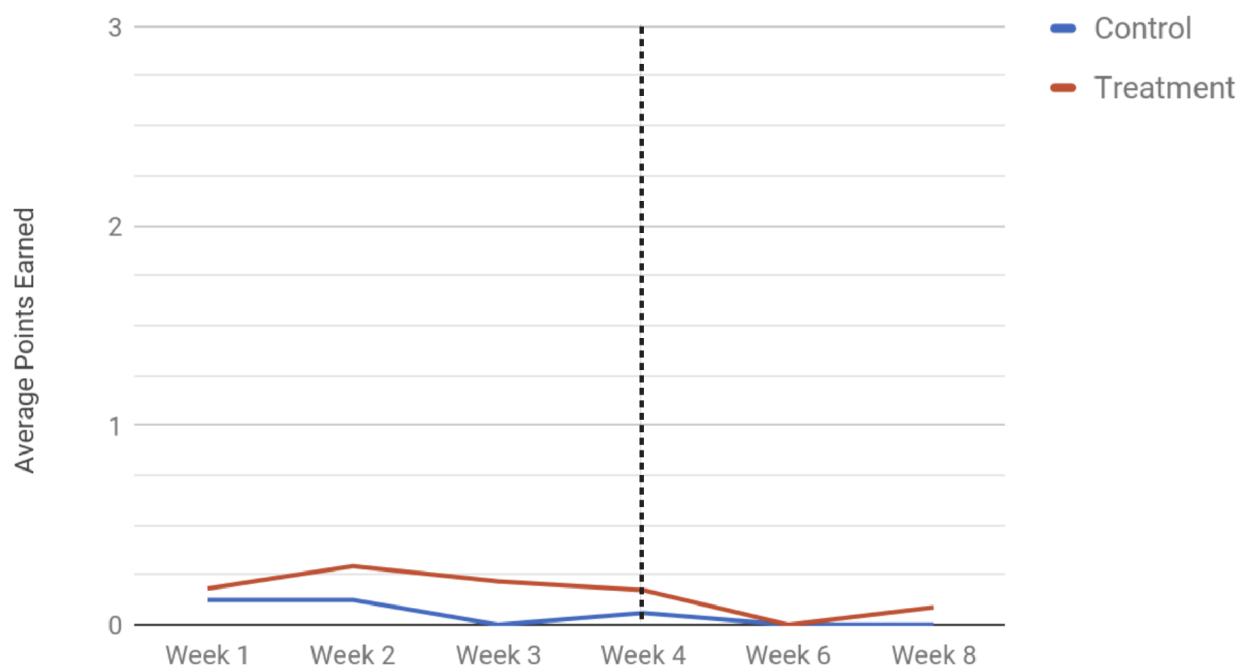


Figure 6. Mean scores on the “logical thinking” construct as measured by Dr. Scratch. *Note:* The LGCM only considered weeks 1–4.

For Dr. Scratch’s measure of logical thinking, both visually and statistically there is nothing that points to evidence of the treatment’s effect on the treatment group (see Figure 6). Looking back on the lessons we taught there was little to no use of if, if/else, or logic operators in the one-week projects they completed from the CCCG (see Table 2, Table 4). As a result, hardly any logical thinking scores were obtained by students. It could be if more lessons were given that required these types of blocks to complete the project there would have been something more interesting to see.

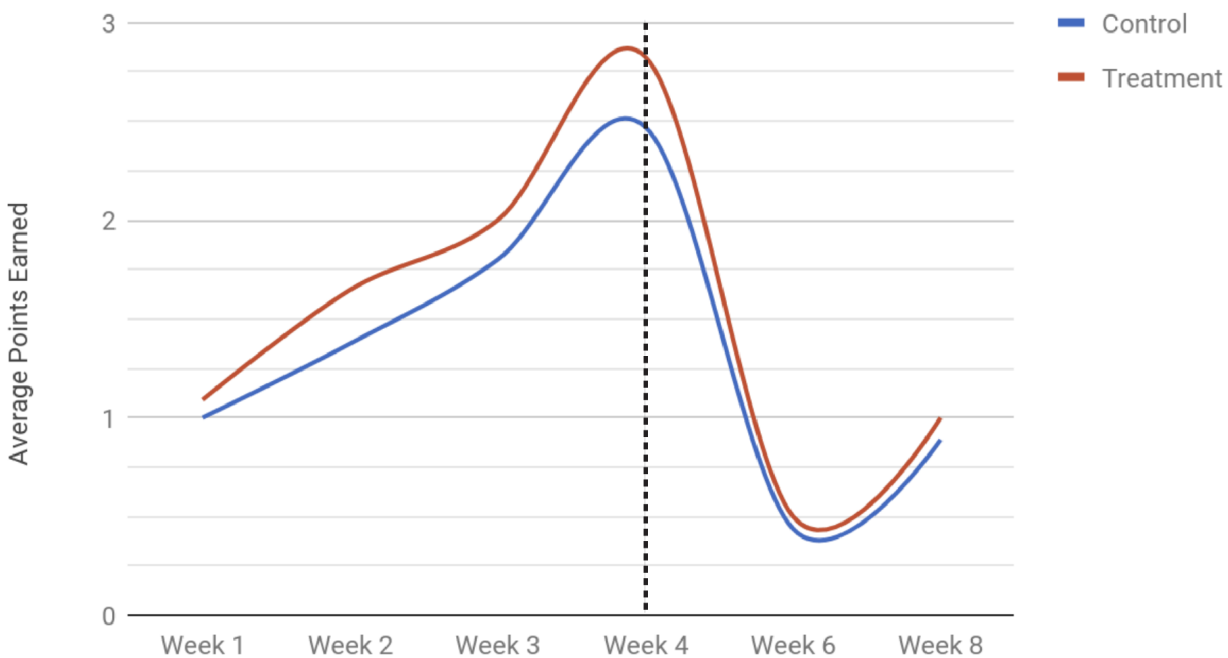


Figure 7. Mean scores on the “flow control” construct as measured by Dr. Scratch. *Note:* The LGCM only considered weeks 1–4.

Dr. Scratch’s measure of flow control did not reveal a linear relationship like the models above (see Figure 7). As such, we needed to represent these results using a non-linear model. While a visual glance might suggest a difference, we could not establish that this was anything more than chance variation ($p = 0.812$). It is possible that we may have found statistical significance with a larger sample size.

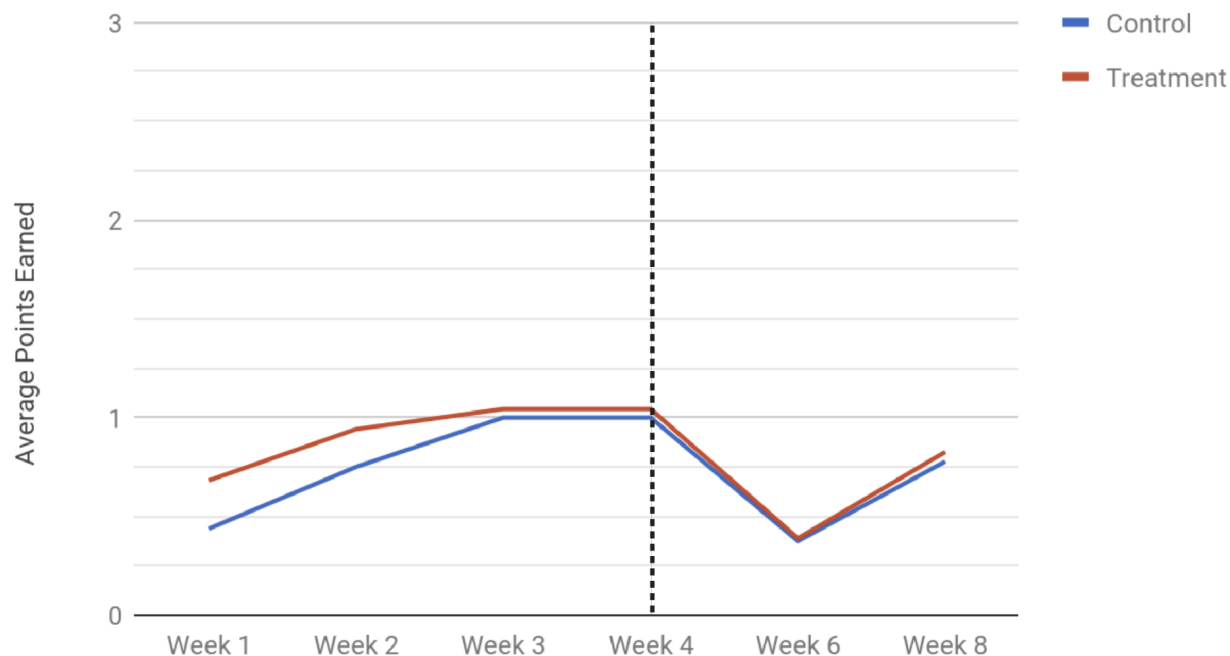


Figure 8. Mean scores on the “data representation” construct as measured by Dr. Scratch. *Note:* The LGCM only considered weeks 1–4.

Dr. Scratch’s measure of data representation does not show much visually (see Figure 8). As well, the statistical model did not converge probably because of low sample size. Both groups were surprisingly consistent with each other in this category.

We further compared all four Dr. Scratch constructs we measured with each other to see if there was any relationship in score between categories. Each week followed the same general pattern: when there was a significant correlation, it was positive, with the highest correlation occurring between abstraction and flow control (Table 7 shows what a typical week looked like).

Table 7

Dr. Scratch Week 4 CT Correlations

		Abstraction	Flow Control	Data Representation	Logic
Abstraction	Pearson Correlation				
	Sig. (2-tailed)				
	N				
Flow control	Pearson Correlation	.537**			
	Sig. (2-tailed)	0.000			
	N	40			
Data Representation	Pearson Correlation	0.066	-0.208		
	Sig. (2-tailed)	0.685	0.197		
	N	40	40		
Logic	Pearson Correlation	-0.036	0.091	.424**	
	Sig. (2-tailed)	0.825	0.576	0.006	
	N	40	40	40	

** . Correlation is significant at the 0.01 level (2-tailed).

Discussion

The results of this study show that the participants in both groups increased their CT ability after only two months of learning how to code regardless of the intervention. While there is little empirical evidence that CT and coding are related, a large number of studies in CT make the implicit assumption that coding and computational thinking are related; often this is done by using coding as the primary activity that CT concepts are applied to (Basawapatna et al., 2011; Brennan & Resnick, 2012; Grover & Pea, 2013; K–12 Computer Science Framework, 2016; Román-González, et al., 2017). This study strengthens the connection between CT and coding, demonstrating that students who regularly engage with coding increase their computational thinking ability.

Block-based coding languages—the most popular being Scratch—are most frequently used by K–12 teachers to teach coding to students (Rich et al., 2016). There is evidence that learning Scratch first helps students to make the transition to text-based coding faster (Armoni, Meerbaum-Salant, & Ben-Ari, 2015). The CTt uses its own unique method of conveying what is happening in its code in a way that removes the need to know a programming language to take the assessment (Román-González, 2015). Thus, as teachers shift away from block-based coding languages, teachers can continue to use the CTt to measure student growth independent of the coding language or platform being taught.

As pointed out by Selby, Dorling, and Woollard (2014), to properly assess CT, there must be an “understanding that computational thinking concepts can be demonstrated in multiple ways” (p. 6). Studies such as Brennan and Resnick (2012) demonstrates that students were able to show different proficiencies in CT when any of the three different assessment techniques were used. Because the nature of Scratch projects varies greatly, and Dr. Scratch assigns points to specific block types as the indicator of CT concepts, Dr. Scratch may be more helpful for some types of projects over others or for certain coding proficiency levels. For example, in this study, Dr. Scratch was more helpful for measuring CT in some areas (e.g., abstraction) more than others (e.g., conditional logic). Some students found creative ways to achieve their goal without using blocks measured by Dr. Scratch. The quick quantitative analysis judged these to be of lesser value, whereas our own qualitative analysis of the project revealed that students were simply demonstrating the same abstract thinking and problem solving without using the same blocks as other coders.

We further recognized the shortcomings of measuring computational thinking by the presence (or lack) of bricks when we looked at students’ final projects. While reviewing the

CYOA projects, we concluded that our students were showing use of logic in their code that was not captured by Dr. Scratch. To wit, to receive Dr. Scratch points in logic requires the use of “if,” “if/else,” or operator blocks (see Table 2). Our students created the functionality of an “if” block by creating button sprites combined with the “when this sprite is clicked” block.

Depending on the purpose, either method may be more effective in achieving a desired result, though the “if/then” logic is more abstracted in the “if” block than in the alternative our students created. This lead us to conclude that a single project may not be a good indicator of what a student’s true CT ability is as measured by Dr. Scratch. According to the research done by Moreno-León, et al., (2017) the CCCG projects we had the participants do fell into the art and animations categories, which are mostly represented in the first and second clusters (Moreno-León, et al., 2017, p. 5, Figure 6). A cluster is a grouping of Scratch projects organized by difficulty level and Dr. Scratch score. Clusters raise in difficulty with three being the highest. As shown in Table 8 below, the average project in clusters one and two demonstrate almost no use of logic and only one point (out of three) in data representation or abstraction. Due to one of our CCCG activities requiring the use of clones in week 4 (see Table 2; Table 4) our students’ projects had a higher level of abstraction than was considered average for that type of project.

Table 8

Cluster Centers for Each of the Three Clusters of the Solution (Moreno-León, et al., 2017, Table 3)

	Cluster		
	1	2	3
Logic	0	0	3
Abstraction	1	1	2
User Interactivity	1	1	2
Data Representation	1	1	2
Parallelism	1	2	2
Synchronization	1	2	3
Flow Control	2	2	3
Mastery	7	9	17

This research on project progression was presented at computing education conferences toward the end of this research. In the future, it might be wise to create curricula demonstrating progressions of increasing complexity as proposed by Moreno-León et al. (2017). Likewise, assessment of computational thinking ability from a single project provides an incomplete picture of a student's ability—both for the student and the teacher. Furthermore, provided that there was no significant difference in students' CTt scores, we question Dr. Scratch's utility to students as a formative tool to improve their coding.

Conclusion

In this paper we compared the difference in pre and posttest CTt scores of two groups of 4th-6th-grade students who were taught how to code in Scratch using the same curriculum and lesson plans. The treatment group used Dr. Scratch as a formative feedback tool to assess how their projects scored. Our propensity score matching of the CTt pre and post comparison left us

with a small sample size of 20 ($p = .696$). While the limitation of the small sample size reduces statistical power, we trust this result the most, though the score difference between missing students in the treatment group on the pretest raises serious questions. The LGCM of the first four weeks shows a positive statistically significant treatment ($p = .002$) in abstraction for those using Dr. Scratch as a formative feedback tool. Flow control may have potentially shown significance with a larger sample size. Logic was not adequately represented in the chosen Scratch projects, and data representation showed little variation between groups. While the results of our study are not as conclusive as we would like, we can see that both groups did improve overall in their CTt scores. Given that both groups started above the average CTt score for their age group (as compared with 5th–6th grade students in Román-González, et al., 2017), this increase bodes well, suggesting that engaging in coding activities in Scratch enabled students to increase their computational thinking ability over a 2-month period. Beyond CT, all of our students increased in their ability to code, though that was not something we measured in this study. Our research and findings lead us to several conclusions.

First, computational thinking is still being defined. As shown earlier in this paper, new definitions are created all the time. One of the primary reasons CT is so difficult to define is that doing so is an attempt at defining a mental thought process; that becomes all the more difficult to measure when the focus of the measurement is on whether “students perform[ed] a thought process” (Mueller, et al. 2017, p. 21). One of the most common ways researchers have attempted to measure CT is the evaluation of artifacts created through the CT process; yet, without a considerable amount of time discussing the CT thought processes used by students, it can be difficult to tell if a student actually was successful in engaging in CT during the artifact’s creation process. Two different students could theoretically create the same program with

exactly the same code but get to that result through entirely different mental thought processes; one may use CT to thoroughly think it through and figure out the code needed, while another may create the code because someone else had informed them at some point that coding the program in that way was the best solution. As Brennan and Resnick (2012) found, some students will even grab a snippet of code from somewhere else knowing it does what they want without ever actually thinking through the purpose of each individual piece within that code. Finding ways to address these concerns are still needed. Despite these shortcomings, we were interested in knowing how using an evaluative feedback tool (Dr. Scratch) might influence students' CT.

The CTt and Dr. Scratch did not necessarily measure the same constructs (see Table 3). Both measures have their own strengths and weaknesses. The CTt is great in its accessibility and usability. Because of its format as a digital multiple-choice test, it is quick for students to take and quick for educators to measure the results. It excels in assessing students' ability to think abstractly and to recognize and follow patterns. But its very nature as a multiple-choice test restricts students from creating original solutions to the problems presented.

Dr. Scratch works very effectively as an evaluation tool for educators (Moreno-León, et al., 2017), but would need to have some changes made to it to become an effective formative feedback tool for students. From our experience, some students were motivated by trying to achieve higher scores, but many were not. Very few students in our study actively tried to learn more from any of the information that Dr. Scratch had to offer. Changes that could help improve Dr. Scratch as a feedback tool might include: a more user friendly interface or more examples of what code would look like at different point levels (e.g., three example projects that all achieve the same purpose, with one receiving one point in abstraction, another two points, and the third receiving three points).

In conclusion, Dr. Scratch, though a useful tool for indicating the relative complexity of a Scratch program (Moreno-León, et al., 2017), may not provide any more benefit as a formative feedback tool than students otherwise get from coding in Scratch. In future studies using Dr. Scratch, we would ensure the projects we have students create have a better balance of helping students learn to code and require use of blocks that require the CT concept we are looking at (e.g., projects that require more use of the logic construct as measured by Dr. Scratch), perhaps following the natural progression of project complexity as promoted by Moreno-León, et al., (2017). Efforts to understand computational thinking are just emerging. The CTt and Dr. Scratch provide two attempts at measuring students' computational thinking abilities. However, perhaps their use is better served in a diagnostic or summative function than in a formative one, filling one important need of computing education. There remains a need for tools that will measure students' formative progress of learning to think computationally.

References

- Ambrosio, A. P., Xavier, C., & Georges, F. (2014). Digital ink for cognitive assessment of computational thinking. In *Frontiers in Education Conference (FIE), 2014 IEEE* (pp. 1–7). <http://dx.doi.org/10.1109/FIE.2014.7044237>.
- Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to “real” programming. *ACM Transactions on Computing Education, 14*(4), 1-15.
doi:10.1145/2677087
- Balanskat, A., & Engelhardt, K. (2015). Computing our future: Computer programming and coding: Priorities, school curricula and initiatives across Europe. *European Schoolnet (EUN Partnership AISBL)*, 1–87. Retrieved from http://www.eun.org/c/document_library/get_file?uuid=3596b121-941c-4296-a760-0f4e4795d6fa&groupId=43887
- 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—SIGCSE '11*, 245–250.
<http://doi.org/10.1145/1953163.1953241>
- Brennan, K., Balch, C., & Chung, M. (2011). Creative computing curriculum guide. *Harvard Graduate School of Education*, 1–149. Retrieved from <http://scratched.gse.harvard.edu/guide/files/CreativeComputing20141015.pdf>
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting, Vancouver, BC, Canada*, 1–25. Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

- Brown, G. T., Irving, S. E., & Keegan, P. J. (2014). *An introduction to educational assessment, measurement and evaluation: improving the quality of teacher-based assessment* (3rd ed.). Auckland, NZ: Dunmore Publishing.
- CSTA, & ISTE. (2011). *Operational definition of computational thinking for K–12 education*. 1–1. Retrieved from:
<http://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CompThinkingFlyer.pdf?hSearchTerms=%22comphinkingflyer%22>
- Curran, P. J., Obeidat, K., & Losardo, D. (2010). Twelve frequently asked questions about growth curve modeling. *Journal of Cognition and Development, 11*(2), 121–136.
<http://doi.org/10.1080/15248371003699969>
- de Araujo, A., Andrade, W., & Serey-Guerrero, D. (2016). A systematic mapping study on assessing computational thinking abilities. In *2016 IEEE frontiers in education conference (FIE)* (pp. 1–9). IEEE. Retrieved from
<http://ieeexplore.ieee.org/document/7757678/>.
- Dong, Y., & Peng, C. Y. J. (2013). Principled missing data methods for researchers. *SpringerPlus, 2*(222), 1–17. <http://doi.org/10.1186/2193-1801-2-222>
- Elahi, A. (2016, March 1). CPS to roll out computer science requirement. *Chicago Tribune*. Retrieved from <http://www.chicagotribune.com/bluesky/originals/ct-computer-science-graduation-cps-bsi-20160225-story.html>
- Frey, C. B., & Osborne, M. A. (2013). The future of employment: How susceptible are jobs to computerisation? *Oxford Martin Programme on Technology and Employment*. 1–72. Retrieved from
http://www.oxfordmartin.ox.ac.uk/downloads/academic/The_Future_of_Employment.pdf

- Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43. <http://doi.org/10.3102/0013189X12463051>
- Hattie, J. (2015). The applicability of visible learning to higher education. *Scholarship of Teaching and Learning in Psychology*, 1(1), 79–91.
- K–12 Computer Science Framework (2016). 1–297. Retrieved from <http://www.k12cs.org>
- Little, R. J. A., & Rubin, D. B. (2002). *Statistical analysis with missing data* (2nd edition). Hoboken, NJ: John Wiley & Sons. <https://doi.org/10.2307/1533221>
- McGrew, K. S. (2009). CHC theory and the human cognitive abilities project: Standing on the shoulders of the giants of psychometric intelligence research. *Intelligence*, 37(1), 1–10. <http://dx.doi.org/10.1016/j.intell.2008.08.004>.
- Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of Scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, 15(46), 1–23. <http://doi.org/10.6018/red/46/10>
- Moreno-León, J., Robles, G., & Román-González, M. (2016). Comparing computational thinking development assessment scores with software complexity metrics. *IEEE Global Engineering Education Conference, EDUCON*, 10–13 (April), 1040–1045. <https://doi.org/10.1109/EDUCON.2016.7474681>
- Moreno-León, J., Robles, G., & Román-González, M. (2017). Towards data-driven learning paths to develop computational thinking with Scratch. *IEEE Transactions on Emerging Topics in Computing*, PP(99)1–1. <https://doi.org/10.1109/TETC.2017.2734818>
- Mueller, J., Becket, B., Hennesey, E., & Shodiev, H. (2017). Assessing computational thinking across the curriculum. In P. Rich & C. Hodges (Eds.), *Emerging research, practice, and policy in computational thinking*. (pp. 251–268) Cham, CH: Springer.

- Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. *SIGCSE 10 Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 265–269. <http://doi.org/10.1145/1734263.1734357>
- Rich, P. J., Bly, N., & Leatham, K. R. (2014). Beyond cognitive increase: Investigating the influence of computer programming on perception and application of mathematical skills. *Journal of Computers in Mathematics and Science Teaching*, 33(1), 103–128.
- Rich, P. J., Yoshikawa, E., Belikov, O., Perkins, M., Browning, S., & Shoop, T. (2016). *Coding in K-8*. Presentation given at the annual meeting for the Association for Educational Communications and Technology. Las Vegas, NV.
- Román-González, M. (2015). Computational Thinking Test: Design guidelines and content validation. *Proceedings of EDULEARN15 Conference*, (July), 2436–2444. <http://doi.org/10.13140/RG.2.1.4203.4329>
- Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, 72, 678-691. <http://doi.org/10.1016/j.chb.2016.08.047>
- Schneider, W. J., & McGrew, K. S. (2012). The Cattell-Horn-Carroll model of intelligence. In D. Flanagan, & P. Harrison (Eds.), *Contemporary intellectual assessment: Theories, tests, and issues* (3rd ed., pp. 99–144). New York, NY: Guilford.
- Selby, C., Dorling, M. and Woollard, J., (2014) Evidence of assessing computational thinking. *Author's original*, 1-11.

- Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*. <https://doi.org/10.1016/j.edurev.2017.09.003>
- Smith, M. (2016). *Computer science for all*. Retrieved April 15, 2016, from <https://www.whitehouse.gov/blog/2016/01/30/computer-science-all>
- Visible Learning. (2016). *Hattie ranking: 195 influences and effect sizes related to student achievement*. Retrieved from <http://visible-learning.org/hattie-ranking-influences-effect-sizes-learning-achievement/>
- Weibell, C. J. (2011). *Principles of learning: A conceptual framework for domain-specific theories of learning* (Doctoral dissertation). Retrieved from BYU ScholarsArchive: All Theses and Dissertations, (2759), 1–495.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <http://dx.doi.org/10.1145/1118178.1118215>.

APPENDIX A: Step-by-Step Lesson

Before Class Challenge (Before class for those who arrive early, but carry over five min MAX into class time): Can the farmer get his Fox, Hen, and bag of corn over the river? He has a boat that is too small and will only allow him to take one item at a time. If left alone, the Fox will eat the hen, and the hen will eat the corn. How do you get them across safely without something being eaten?

Lesson (10-15 minutes): Have students write down instructions for how to make a peanut butter and jelly sandwich. Have a teacher follow the instructions exactly (something *always* goes wrong, and students find it hilarious). Explore with the students using Scratch on the classroom projector how to use blocks in the motion and looks categories of Scratch, show an example of how to do the Step-by-step activity shown on the handout below. For the advanced students we would have them synchronize their dance to the music for greater difficulty.

Treatment Group only (1-2 minutes): Show Dr. Scratch and explain that students want to get at least three total points, with at least one point in flow control, data representation, and user interactivity for the Step-by-step project.

Project Time (40–45 minutes): Give students the Step-by-step handout below from *Creative Computing Curriculum Guide* (Brennan, Balch, & Chung, 2011) and have them work individually on computers on the activity. Rove and answer questions as needed.

STEP-BY-STEP

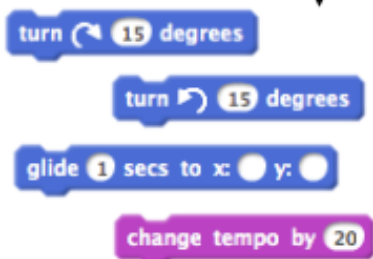
NEW TO SCRATCH? CREATE YOUR FIRST SCRATCH PROJECT!

In this activity, you will follow the Step-by-Step Intro in the Tips Window to create a dancing cat in Scratch. Once you have completed the steps, experiment by adding other Scratch blocks to make the project your own.

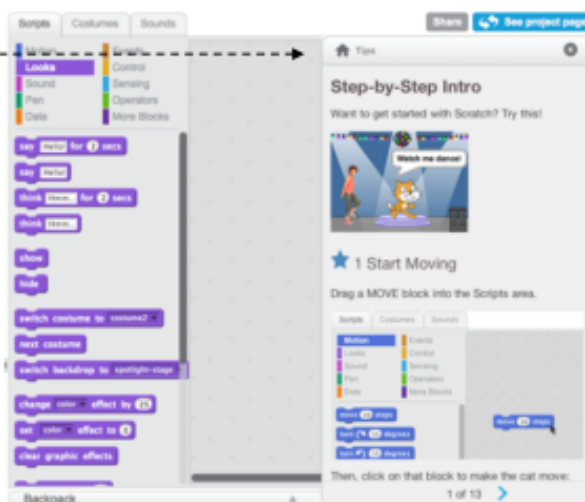


START HERE

- Follow the Step-by-Step Intro in the Tips Window.
- Add more blocks.
- Experiment to make it your own!



What blocks do you want to experiment with?



THINGS TO TRY

- Try recording your own sounds.
- Create different backdrops.
- Turn your project into a dance party by adding more dancing sprites!
- Try designing a new costume for your sprite.

FINISHED?

- + Add your project to the Step-by-Step Studio: <http://scratch.mit.edu/studios/475476>
- + Challenge yourself to do more! Play with adding new blocks, sound, or motion.
- + Help a neighbor!
- + Choose a few new blocks to experiment with. Try them out!

APPENDIX B: Annotated Bibliography

Introduction

Computational thinking (CT) is a term that refers to the mental process of problem solving with technology used by computer scientists. The term was popularized by Jeanette Wing (2006) in which she demonstrated that CT is a skill used by and needed across a variety of disciplines. As more of the world around us requires computer science (CS) and computer literacy, CT processes will become more valuable in our schools, employment, and homes. The main purposes of this annotated bibliography are to: (a) provide a background on the nature of computational thinking, how it is affecting our schools, and how students can be assessed in their CT ability; and (b) provide support and background for the following research questions: Does feedback from Dr. Scratch as a formative feedback tool result in higher scores in computational thinking as measured by the Computational Thinking Test (CTt)? What are the differences in students' created programs as measured by Dr. Scratch when they have used Dr. Scratch as a formative feedback tool and when they have not?

The scope includes research in the following categories: coding in K–12 schools, computational thinking definitions, assessing computational thinking, data analyzing, feedback, and Dr. Scratch.

Literature Search Methodology

To identify articles selected for inclusion in my annotated bibliography I looked up “computational thinking” and “k-12” in the Institute of Electrical and Electronics Engineers (IEEE), Google Scholar, and the Association for Computing Machinery (ACM) databases. IEEE is a database common for use in engineering circles, which resulted in many computational thinking conference presentations. The ACM is a widely-used database for computing professionals. This source likewise provided many conference presentations, but also provided some of the more recently published manuscripts on computational thinking. Searching Google Scholar allowed me to find sources that may not have appeared in these engineering and computing-centric sources, such as those represented in education journals. Finally, I also included a few journal articles from popular media, which demonstrated evidence for recent computational thinking (CT) initiatives. This bibliography is not meant to be a comprehensive review of all research on CT. Rather, it demonstrates reasonable support for the questions posed in this study. There may be other relevant manuscripts not included in the following list.

Coding in K–12 Schools

The articles in this section express the need for computer science (CS)/CT to become part of the public school curriculum (Frey & Osborne, 2013; Grover & Pea 2013; Repenning, Webb, & Ioannidou, 2010), give updates on adoption of CS/CT into public school curriculum (Balanskat, & Engelhardt, 2015; Elahi, 2016; Smith 2016), and provide support for educators making this change (Balanskat, & Engelhardt, 2015; Brennan, Balch, & Chung, 2011; K–12 Computer Science Framework, 2016; Rich et al., 2016).

Balanskat, A., & Engelhardt, K. (2015). Computing our future: Computer programming and coding: Priorities, school curricula and initiatives across Europe. *European Schoolnet (EUN Partnership AISBL)*, 1–87. Retrieved from http://www.eun.org/c/document_library/get_file?uuid=3596b121-941c-4296-a760-0f4e4795d6fa&groupId=43887

This study was conducted by the European Schoolnet in which they reviewed the current state and future plans of CS education in public schooling within the European Union. Some of the key questions this study answers include: Which countries are formally teaching coding? How do coding skills relate to other skills? How is coding integrated into the curriculum? How is coding being assessed coding? and How are teachers being prepared to teach coding?

Brennan, K., Balch, C., & Chung, M. (2011). Creative computing curriculum guide. Harvard Graduate School of Education, 1–149. Retrieved from <http://scratched.gse.harvard.edu/guide/files/CreativeComputing20141015.pdf>

This curriculum guide was created by the same group responsible for creating Scratch. The guide was built to be flexible for a wide range of educators (e.g., K–12, universities, museums). It contains an instructor guide and student worksheets for each lesson. Lessons build on each other to some degree while teaching students how to use Scratch and navigate its interface. The guide encourages creativity and exploring everything Scratch has to offer.

Elahi, A. (2016, March 1). CPS to roll out computer science requirement. *Chicago Tribune*.

Retrieved from <http://www.chicagotribune.com/bluesky/originals/ct-computer-science-graduation-cps-bsi-20160225-story.html>

This news article highlights how CS is becoming a required subject in Chicago Public Schools district starting with the class of 2020. The article then explains how one of the other three largest school districts in the country (Los Angeles) also requires CS, but New York City's school district does not. The article also explains that in the Chicago Public Schools district only 41 out of 176 high schools are ready to offer acceptable computer science courses.

Frey, C. B., & Osborne, M. A. (2013). The future of employment: How susceptible are jobs to computerisation? *Oxford Martin Programme on Technology and Employment*. 1–72.

Retrieved from

http://www.oxfordmartin.ox.ac.uk/downloads/academic/The_Future_of_Employment.pdf

This article seeks to estimate how many of the 702 occupation types it studies from the US labor market are likely to be automated by computers in the future. Frey & Osborne estimate that “about 47 percent of total US employment is at risk.” They then seek to show how educational attainment and wages have a strong negative relationship with their job automation estimates.

Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43. <http://doi.org/10.3102/0013189X12463051>

This article framed the then current state of CT in the field, elaborating on how CT had progressed and developed since Jeannette Wing first published on CT in 2006. One section of Grover & Pea's article discusses how CT/CS was introduced in schools, and highlights some NSF initiatives that have encouraged CT/CS to enter secondary level of schools. The initiatives include the NSF CS10K initiative which aimed to bring 10,000 new CS teachers into schools by 2015. The article also mentions that despite existing NSF initiatives, there is actually a general lack of funding from NSF in the CT/CS area.

K–12 Computer Science Framework (2016). 1–297. Retrieved from <http://www.k12cs.org>

This framework is an example of how seriously schools are taking CS/CT integration into public education. It was built as a framework for U.S. states to use to build their computer science teaching curriculum. The framework built on similar frameworks from the UK and Australia and was vetted by an advisory board and three rounds of open comment, each lasting a month. It gives definitions of computational thinking, practices or core concepts of CT, guidance for standard developers, and implementation guides.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. *SIGCSE 10 Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 265–269. <http://doi.org/10.1145/1734263.1734357>

This article recognizes the need for more members of the workforce to have a background in CS and IT, and suggests ways to get CS into public schools by “scal[ing] up from after school and summer programs into required curriculum.” It seeks to interest students in CS by teaching them how to design and create games.

Rich, P. J., Yoshikawa, E., Belikov, O., Perkins, M., Browning, S., & Shoop, T. (2016). *Coding in K-8*. Presentation given at the annual meeting for the Association for Educational Communications and Technology. Las Vegas, NV.

This study sent out a snowballing 15–20 minute survey to CS teachers and teachers of CS to students ages 5–14 worldwide. The survey asked many open-ended response questions that were then categorized and coded with like answers. A total of 303 Respondents from 22 countries responded; most relevant to my thesis is that Scratch is by far the most commonly used coding language worldwide.

Smith, M. (2016). Computer science for all. Retrieved April 15, 2017, from <https://www.whitehouse.gov/blog/2016/01/30/computer-science-all>

This White House Press release is about President Obama’s initiative “to empower all American students from kindergarten through high school to learn computer science and be equipped with the computational thinking skills.” This initiative among other things provided \$4 billion in federal funding to states and \$100 million directly for school districts for the purpose of training new CS teachers, obtaining instructional materials, and building regional partnerships.

Computational Thinking Definitions

The articles in this section create precise definitions of the meaning of computational thinking (CSTA & ISTE, 2011; K–12 Computer Science Framework, 2016; Shute, Sun, & Asbell-Clarke, 2017, Wing, 2006), and/or delve into the subjects and principles that are aspects of computational thinking (CSTA & ISTE, 2011; Grover & Pea, 2013; Shute, Sun, & Asbell-Clarke, 2017).

CSTA, & ISTE. (2011). *Operational definition of computational thinking for K–12 education*. 1–

1. Retrieved from:

<http://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CompThinkingFlyer.pdf?hSearchTerms=%22compthiningflyer%22>

The International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) worked together with other experts in CS to create for an operational CT framework and definition, which they established as the following:

“Computational thinking (CT) is a problem-solving process that includes (but is not

limited to) the following characteristics: formulating problems in a way that enables us to use a computer and other tools to help solve them; logically organizing and analyzing data; representing data through abstractions such as models and simulations; automating solutions through algorithmic thinking (a series of ordered steps); identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources; generalizing and transferring this problem solving process to a wide variety of problems.”

Grover, S., & Pea, R. (2013). Computational Thinking in K–12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43. <http://doi.org/10.3102/0013189X12463051>

This article provided an operational definition of concepts found in CT: “Abstractions and pattern generalizations (including models and simulations); Systematic processing of information; Symbol systems and representations; Algorithmic notions of flow of control; Structured problem decomposition (modularizing); Iterative, recursive, and parallel thinking; Conditional logic; Efficiency and performance constraints; Debugging and systematic error detection” (pp. 39–40).

K–12 Computer Science Framework (2016). 1–297. Retrieved from <http://www.k12cs.org>

The K–12 Computer Science Framework examined all past definitions as well as educational standards documents for computing from England and Australia (two countries that have made computing compulsory at every level). Additionally, the

framework was reviewed by dozens of collaborators in three rounds of open-feedback, with a panel of experts refining the framework after each open-review session (which each lasted roughly two months). This definition presents computational thinking as, “the thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer” (K–12 Computer Science Framework, 2016, p. 68).

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*. <https://doi.org/10.1016/j.edurev.2017.09.003>

This article puts forward another working definition of CT: “The conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts.” The goal of this definition was to be actionable by students and therefore assessable. It also categorizes CT “into six main facets: decomposition, abstraction, algorithm design, debugging, iteration, and generalization.”

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <http://dx.doi.org/10.1145/1118178.1118215>.

This seminal article encouraged the focus on computational thinking as a mental process for problem solving. Wing also defines computational thinking as: Computational

thinking “involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” (p. 33).

Assessing Computational Thinking

The articles in this section look at how computational thinking is being assessed (de Araujo, Andrade, & Serey-Guerrero, 2016); or assessments created (Mueller, et al., 2017; Román-González, 2015; Román-González, et al., 2017) using video (Basawapatna, et al., 2011), or how to design interviews or scenarios intended to measure or describe computational thinking concepts and practices (Brennan & Resnick, 2012).

de Araujo, A., Andrade, W., & Serey-Guerrero, D. (2016). A systematic mapping study on assessing computational thinking abilities. In *2016 IEEE frontiers in education conference (FIE)* (pp. 1–9). IEEE. Retrieved from <http://ieeexplore.ieee.org/document/7757678/>.

The goal of this study was to “identify and classify” different ways CT is “promoted” and assessed. Interesting findings relevant to my paper include: Scratch is the most commonly used coding language; looking at students’ code and multiple choice assessments are the two most common ways that CT ability is measured; and the most frequently assessed CT concepts are solving problems, algorithms, and abstraction.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting, Vancouver, BC, Canada*, 1–25. Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

This article developed a framework that is increasingly cited by those measuring CT. By studying the Scratch online community as well as Scratch workshops, the authors created a framework highlighting three key ways CT manifests in students: “computational concepts (the concepts designers employ as they program),” e.g., loops, conditionals, operators, data, etc.; “computational practices (the practices designers develop as they program),” e.g., testing and debugging, abstracting and modularizing, etc.; “and computational perspectives (the perspectives designers form about the world around them and about themselves)” (p. 3) e.g., expressing, connecting, questioning.

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—SIGCSE '11*, 245–250.

<http://doi.org/10.1145/1953163.1953241>

This study uses videos unrelated to CT to see if students can recognize computational thinking patterns (defined as: cursor control, generations, absorption, collision, transportation, push, pull, diffusion, and hill climbing) in unassociated contexts. To do this, they created a short answer assessment that was paired with videos that could be

interpreted as showing computational thinking patterns (e.g. showing a sledder sliding into someone as an example of collision). According to their results, most students in their study were able to recognize patterns in unrelated contexts.

Mueller, J., Becket, B., Hennesey, E., & Shodiev, H. (2017). Assessing computational thinking across the curriculum. In P. Rich & C. Hodges (Eds.), *Emerging research, practice, and policy in computational thinking*. (pp. 251–268) Cham, CH: Springer.

This book chapter states there is not a reliable tool to assess computational thinking, and states that part of what makes CT so difficult to measure is that CT “concentrates on students performing a thought process, not on the production of artefacts or evidence” (p. 21). It then suggests “a workable tool to assess CT based on existing literature.”

Román-González, M. (2015). Computational Thinking Test: Design guidelines and content validation. *Proceedings of EDULEARN15 Conference*, (July), 2436–2444.
<http://doi.org/10.13140/RG.2.1.4203.4329>

This article describes the creation of the Computational Thinking Test (CTt). The purpose of this assessment is to be a measure for computational thinking. The CTt started as a 40 item assessment, went through a content validation process involving 20 experts in teaching CS, and ended with the 28 best questions. Possible limitations of the test are also discussed.

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior, 72*, 678-691.
<http://doi.org/10.1016/j.chb.2016.08.047>

This study administers the Computational Thinking Test (CTt) to 1,251 students across 5th–10th grade in Spain. The results—including reliability—are reported. Of the same students, half then took the Primary Mental Abilities (PMA) battery and the other half took the RP30 problem-solving test. Results from all three tests were then compared.

“Results show statistically significant correlations at least moderately intense between CT and: spatial ability ($r = 0.44$), reasoning ability ($r = 0.44$), and problem-solving ability ($r = 0.67$).”

Data Analyzing

This section of the bibliography focuses on two of the main types of data analyses used in this manuscript, namely latent growth curve modelling (Curran, Obeidat, & Losardo, 2010) and full information maximum likelihood (Dong & Peng, 2013; Little & Rubin, 2002).

Curran, P. J., Obeidat, K., & Losardo, D. (2010). Twelve frequently asked questions about growth curve modeling. *Journal of Cognition and Development, 11*(2), 121–136.
<http://doi.org/10.1080/15248371003699969>

This article seeks to address the difficulty that developmental psychology researchers face when trying to get a broad understanding of literature surrounding growth curve modeling. The article offers 12 frequently asked questions about growth curve modeling. Questions they pose include: “What is growth curve modeling?,” “What are the data requirements to use growth models?,” “Can growth models be estimated with partially missing data?,” etc. The article also provides further reading relating to each question.

Dong, Y., & Peng, C. Y. J. (2013). Principled missing data methods for researchers.

SpringerPlus, 2(222), 1–17. <http://doi.org/10.1186/2193-1801-2-222>

This article compares three different statistical methods for dealing with missing data: multiple imputation, full information maximum likelihood, and expectation-maximization algorithm. It gives a background on how these different methods work and why you would want to use them. It then uses all three methods on a real world data set and contrasts the results of each method with the complete data set.

Little, R. J. A., & Rubin, D. B. (2002). *Statistical analysis with missing data* (2nd edition).

Hoboken, NJ: John Wiley & Sons. <https://doi.org/10.2307/1533221>

One of the main assumptions of this book is that “missingness indicators hide true values that are meaningful for analysis” (p. 8). Little and Rubin thoroughly explore different methods that account for missing data, including FIML. They then take these methods and apply them to various missing data sets.

Feedback

This section of the bibliography focuses on why feedback is an important method in helping students learn (Hattie, 2015), what feedback is, how feedback fits into different learning theories (Weibell, 2011), how feedback is used in various teaching models (Askew, & Lodge, 2000), how feedback affects language learning (Nakata, 2017), and how students' view of feedback affects them academically (Brown, Peterson, & Yao, 2016).

Askew, S., & Lodge, C. (2000). Gifts, ping-pong and loops – linking feedback and learning. In S. Askew (Eds.), *Feedback for Learning* (pp. 1–18). London: RoutledgeFalmer.

This book chapter asserts that feedback promotes learning and then proceeds to examine how different models of teaching approach feedback differently: in the Receptive–transmission model feedback is considered a “gift” from teacher to student, in the Constructive model feedback is “a two-way process (ping pong),” and in the Co-constructive model feedback “is a dialogue, formed by loops connecting the participants” (p. 4, Table 1.1).

Brown, G. T. L., Peterson, E. R. and Yao, E. S. (2016), Student conceptions of feedback: Impact on self-regulation, self-efficacy, and academic achievement. *British Journal of Educational Psychology*, 86(4), 606–629. doi:10.1111/bjep.12126

This article explores how a student views feedback (i.e. its role and purpose) affects student's self-reported self-regulation (SRL) and academic self-efficacy (ASE), and grade point average (GPA). Significant findings include that the students who reported 'I use feedback' had a positive correlation in all three categories: SRL ($\beta = .44$), GPA ($\beta = .45$), and ASE ($\beta = .15$). The authors conclude "The model indicates the centrality of believing that feedback exists to guide next steps in learning and thus contributes to SRL, ASE, and increased GPA."

Hattie, J. (2015). The applicability of visible learning to higher education. *Scholarship of Teaching and Learning in Psychology*, 1(1), 79–91.

This article looks at how higher education teachers can use John Hattie's 1200 meta-analyses synthesis to look not at what "works," but rather "what works best." Ways the article seeks to apply the synthesis include: teachers seeking evidence of their impact on students, appropriate levels of challenge and feedback, surface and deep learning. This article shows that feedback is an important form of intervention.

Nakata, M. (2017). Lingua Anglia: Bridging language and learners: Strike while the error is hot: Enhancing language learning through meaningful oral feedback. *English Journal*, 106(6), 81–83. Retrieved from <http://www.ncte.org.erl.lib.byu.edu/library/NCTEFiles/Resources/Journals/EJ/1066-july2017/EJ1066Lingua.pdf>

This article discusses how giving feedback immediately in the situation an error is made quickly helps the learner to apply that feedback and learn from it in language learning. For effective feedback it must be: clear, specific, timely, encouraging, ongoing, develop an awareness, and get buy-in.

Weibell, C. J. (2011). *Principles of learning: A conceptual framework for domain-specific theories of learning* (Doctoral dissertation). Retrieved from BYU ScholarsArchive: All Theses and Dissertations, (2759), 1–495.

This study proposes a shift away from learning theories towards learning principles. This study does an extensive review of major works in behavioral, cognitive, constructive, human, and social learning theories. 14 principles are identified: repetition, time, step size, sequence, contrast, significance, feedback, context, engagement, agency, potential, target, change, and practice.

Dr. Scratch

The articles in this section describe the creation, goals, and methods behind Dr. Scratch (Moreno-León, et al., 2015); how Dr. Scratch compares to McCabe’s Cyclomatic Complexity and Halstead’s metrics (Moreno-León, et al., 2016); and how Dr. Scratch has been used to differentiate levels of computational thinking ability based on project type ((Moreno-León, et al., 2017).

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of Scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, 15(46), 1–23. <http://doi.org/10.6018/red/46/10>

This article is the first one that introduces Dr. Scratch. It explains how Dr. Scratch works and the goals behind it, including: supporting teachers' evaluation of student programs and "to act as a stimulus to encourage students to keep on improving their programming skills." The article then explains a study to test if Dr. Scratch meets these goals, and conclude that their results show Dr. Scratch helped students "increase their CT score" and "[improve] their coding skills."

Moreno-León, J., Robles, G., & Román-González, M. (2016). Comparing computational thinking development assessment scores with software complexity metrics. *IEEE Global Engineering Education Conference, EDUCON*, 10–13 (April), 1040–1045. <https://doi.org/10.1109/EDUCON.2016.7474681>

This article compares the score Dr. Scratch gives in computational thinking with McCabe's Cyclomatic Complexity and Halstead's metrics. The study finds that the correlations between Dr. Scratch and the two metrics are "positive, significant, [and] moderate to strong." The article states this "could be therefore considered as a validation of the complexity assessment process of Dr. Scratch."

Moreno-León, J., Robles, G., & Román-González, M. (2017). Towards data-driven learning paths to develop computational thinking with Scratch. *IEEE Transactions on Emerging Topics in Computing*, *PP(99)*1–1. <https://doi.org/10.1109/TETC.2017.2734818>

This article uses Dr. Scratch to analyze 250 Scratch projects from five categories: games, animations, music, art, and stories. Each project was also analyzed by a researcher. Using this information, the researchers were able to show different categories of Scratch projects require different levels of computational thinking ability.

APPENDIX C: Parental Permission for a Minor

Parental Permission for a Minor

Introduction

My name is Geoff Wright, Ph.D. I am a professor from Brigham Young University. I am conducting a research study about the impact teaching programming and robotics has on students understanding of mathematics. I am inviting your child to take part in the research because (he/she) stated they were interested in participating in a programming and robotics course.

Procedures

The research project includes your child taking a short mathematics inventory survey, and then participating in a programming/robotics course, and then taking a short mathematics post inventory survey. The course will last approximately 1 month (depending on classroom availability at your school). We will use the data collected from the pre and post math surveys to measure the impact of teaching programming and robotics has on student understanding of math. All data will be anonymous. Videos will be filmed documenting the teaching and learning process. The videos will only be used for research purposes and will not be made public. See video release form for further questions.

Risks

Experimental Discomfort: Although this course focuses on teaching programming skills, we are interested in studying the relationship generated between these skills and mathematics in general. To measure this, we will administer problem-solving items for the balanced assessment project (<http://balancedassessment.concord.org/>), prior to and following the experience. We will choose purposefully items above students' mathematical ability level. These tests may cause students some anxiety, as they may not normally associate programming with mathematics assessment.

Confidentiality

The research data will be kept in a secure location (password protected and encrypted) and only the researcher will have access to the data. At the conclusion of the study, all identifying information will be removed and the data will be kept in a locked cabinet or office.

Benefits

There are benefits to participating in this study. The most obvious is that your student will be learning about programming and robotics. In addition, we will be directly teaching some mathematics, which may help them in their math courses at school.

Compensation

There will be no compensation for participation in this project.

Questions about the Research

Please direct any further questions about the study to Geoff Wright, Ph.D., at geoffwright@byu.edu, 801-44-7804.

Questions about your child's rights as a study participant or to submit comment or complaints about the study should be directed to the IRB Administrator, Brigham Young University, A-285 ASB, Provo, UT 84602. Call (801) 422-1461 or send emails to irb@byu.edu.

You have been given a copy of this consent form to keep.

Participation

Participation in this research study is voluntary. You are free to decline to have your child participate in this research study. You may withdraw you child's participation at any point without affecting your child's grade/standing in school, treatment, or benefits, etc.

Child's Name: _____

Parent Name: _____ Signature: _____ Date: _____