THE UNIVERSITY OF CHICAGO

BEYOND CHILD'S PLAY: UNDERSTANDING AND SUPPORTING PROGRAM COMPREHENSION IN YOUNG LEARNERS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
JEAN SALAC

CHICAGO, ILLINOIS
DECEMBER 2021

For my fellow scholars of color who were told that you weren't good enough. I see you.

I celebrate you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I'm grateful for my fiancé John Zhang for being my relentless cheerleader and my anchor throughout this storm of a PhD. Thank you for all your sacrifices, for all your emotional support, for your patience, for looking out for me, for being my sounding board, for reminding me of my worth, and for the beloved little family we build with Abby Salac Zhang. You went through the roller coaster of getting a PhD when you weren't getting one yourself. Needless to say, I wouldn't have gotten through this process in one piece without you. I couldn't have wished for a better partner.

I'm grateful for my parents, Diana Mansilungan and Jude Salac, and my sister, Kim Salac. Thank you for all the sacrifices you made in immigrating to the US, for providing for me and putting a roof over my head, and for your love and support. I'm grateful for my Tita Niña and Tito Ray Capulong, for being my home away from home in the Midwest. Thank you for your warmth and for welcoming me into your home. I'm grateful for my Tito Philip Canlas, for being a fellow nerd and for introducing me to a sport that helps me cope with the unpredictability of research and reminds me of my strength.

I'm grateful for my advisor, Prof. Diana Franklin. Thank you for your research mentorship and guidance, for advocating for me, and for modeling what work-life balance in academia can look like. I'm grateful for my committee, Prof. Amy J Ko, Prof. Tiffany Barnes, and Prof. Marshini Chetty. Thank you for taking the time to review this tome of a dissertation and for providing insightful feedback to improve it. I'm grateful for Donna Eatinger and Dr. Jennifer Tsan for helping me with my dissertation analysis and for encouraging and supporting me throughout the process, Merijke Coenraad for all the research feedback, Jen Palmer for your enthusiasm and expertise in Scratch Encore, and Carla Strickland for keeping it real. I'm grateful for Prof. Cathy Thomas for your mentorship, honesty, and emotional availability, and Chloe Butler for being the one who made the trains run on time in our projects. I'm grateful for all the undergraduate researchers I have worked with over the

years: Marco Anaya, Zach Crenshaw, Qi Jin, Zipporah Klain, Ashley Sanchez, Zené Sekou, Saranya Turimella, Ashley Wang, and Max White. Without you all, our projects wouldn't be where they are today and my summers would have been much duller. I'm grateful for Bryan Twarek and Bill Marsland, for helping me get started with my doctoral research.

I'm grateful for my fellow CS PhD students: Suhail Rehman, Francesca Falzon, Yulie Zamora, Pranav Gokhale, Saeid Barati, Yi Ding, Anne Farrell, Konstantinos Ameranis, Casey Duckering, Will Brackenbury, Michelle Aninye, Bruno Barbarioli, Jenna Cryan, Andrew McNutt, Kartik Singhal, Kavon Farvardin, Akshima, Alex Hoover, Tushant Mittal, Chengcheng Wan, Weijia He, Ivana Marincic, Valerie Zhao, Chris Jones, Emily Wenger, Muhammad Santriaji, and many others who have touched my life in the past four-ish years. Thanks for making an only-lab-child feel less alone, for the spontaneous office chats, for the lunch company, for the solidarity, for being sounding boards, and for the Chicago adventures. My PhD time would have been much less enjoyable without you all.

I'm grateful for the community I've built beyond my department: Jessy Morgan, Travis Bee, Bipul Pandey, Ram Itani, Airi Kawamura, Andrew McNeece, Huw Rees, Alexis Thomas, Katie Aracena, and Jake Higgins. Thank you for being a breath of fresh air, for being a nice break from shop talk, and for cultivating my interests beyond research.

I'm grateful for my community of computing education grad students, especially Joslenne Peña and Nick Lytle. I can say without exaggeration that meeting you both at the 2019 ICER doctoral consortium changed the trajectory of my PhD. Y'all filled a hole in my support network I didn't know I was missing. My PhD experience has been much richer with both of you by my side. I'm also grateful for Ethel Tshukudu, for being each other's cheerleaders across the pond. Paper writing would have been much less fun without you. I'm grateful for Melissa Perez, for your cheery disposition that never fails to make me smile, for Elizabeth Cole, for sharing your wisdom and expertise in early childhood education, and for Philipp Kather, for all our conference back-channeling.

I'm grateful for my oldest friends, Katie Silva, Rachel Pehrsson, and Alexandra Polchek. Thank you for all the conversations that still flow like we talk everyday and for being a steady force in my life all these years. I'm grateful for my Hoo Crew, Sam Havron, Salah Assana, Anant Kharkar, Andrew Norton, and Jenny Xing. Thank you for giving me the space to geek out and vent in our semi-monthly calls and for all your different perspectives over the years. I'm grateful for Prof. Rider Foley and Prof. Luther Tychnonievich. Thank you for getting me started in research and for your guidance and mentorship since my undergraduate days.

I'm grateful for my therapist, Linda Bartoli and my 2020-21 therapy group: facilitator Cristen Brossok, Hannah Kessler-Jones, Lexi Lalungo, Laura Merchant, and other group members who choose to remain anonymous for privacy but I'm equally grateful for nonetheless. Thank you for helping me unpack old and new traumas, grow in my vulnerability and self-reflection, and process the joys and hardships of life.

The projects in this dissertation were funded by National Science Foundation (NSF) Grants No. 1660871 and 1738758, and the NSF Graduate Research Fellowship Program under Grant No. DGE-1746045.

ABSTRACT

Worldwide, many countries are integrating Computer Science (CS) and Computational Thinking (CT) instruction into elementary school curricula. This push for CS/CT instruction in younger ages increases the need to better understand how young learners come to a comprehension of programs and how they can be effectively supported in their learning. However, research into this age group (ages 9-15) is relatively thin compared with research into university-age learners. Research for university-age students is unlikely to directly translate to younger learners. Further, the context in which young learners learn computing differ greatly from the university setting, making direct translation difficult, if not impossible.

This dissertation outlines a series of studies that illuminate the contributing factors to program comprehension, the different types of program comprehension achieved, and the strategies that support program comprehension in young learners. We have found that societal factors, such as school environment, gender, and under-represented minority status, and academic factors, such as reading and math proficiency, contribute to various aspects of program comprehension. Further, our studies revealed that students frequently use code that they only demonstrate a functional, not a structural, understanding of, when learning in an open-ended curriculum. To bridge these gaps in elementary program comprehension, we developed two strategies: (1) TIPP&SEE, inspired by previewing and navigating strategies in reading, and (2) diagramming, inspired by similar strategies used in university CS and elementary math. Exploratory studies have shown TIPP&SEE, a mnemonic for students to remember the steps in exploring a new Scratch project, to be associated with positive performance differences in summative CT assessments and in project completion rates, as well as with narrowed gaps between students with and without academic challenges. In contrast, a diagram-based implementation of TIPP&SEE was linked to similar performance as a text-based implementation of TIPP&SEE, with implications for the design of such diagrams for young learners. Taken together, this body of work deepens our collective

understanding of program comprehension in young learners for more effective and equitable implementations of elementary CS/CT curricula.

CHAPTER 1

INTRODUCTION

With the launch of the CS for All initiative in the US, many American school districts, including San Francisco, Chicago, and New York City, are integrating Computer Science and Computational Thinking instruction at the pre-university level [7]. This trend is not unique to the United States; countries such as New Zealand, Israel, and India have implemented similar programs [103]. With the growing spread of elementary and secondary CS and CT instruction worldwide, it is imperative that such instruction is effective for a broad set of learners.

In spite of the recent push for elementary CS/CT instruction, research on how young learners learn to read and write code is comparatively thin, especially with respect to research on university-aged learners. At the university level, scholars have studied a whole host of factors that contribute to success in introductory programming, including performance in other subjects [35, 250], cognitive and metacognitive skills [15, 51, 86, 235], belief systems [15, 250], and prior experience [31, 92, 189, 228, 248]. In contrast, factors at the elementary school level are relatively under-explored. Existing studies have identified english and math performance, prior computing experience, and extracurricular technology activities as factors that contribute to success in CS learning [89, 135, 187]. Similarly, there is a wealth of research in programming learning strategies at the university level. Strategies include reading and tracing code [138], sketching [49, 137], and other self-regulated learning strategies [64]. In comparison, at the elementary school level, only two strategies are widely employed: Use→Modify→ Create [127] and PRIMM [217].

In this dissertation, I present a body of work that investigate the contributing factors to program comprehension, the different types of comprehension achieved, and the strategies that support program comprehension in young learners. I pursued the following overarching research questions across all my studies:

- Which factors, from academic skills to demographics, are associated with program comprehension in a formal school setting?
- What kinds of comprehension do young learners demonstrate after open-ended, exploratory instruction?
- How can we support the development of program comprehension in young learners?

Our contributions include:

- determining how different academic and non-academic factors are linked to performance in formal introductory CS/CT instruction,
- identifying the different levels of comprehension students achieve through open-ended instruction,
- developing the TIPP&SEE learning strategy, a mnemonic for students to remember the steps in exploring a new Scratch project, that is associated with improved performance and narrowed gaps between students with and without academic challenges, and
- exploring diagramming as a scaffold for teachers and students to learn decomposition, with implications for diagram design in elementary computing.

The rest of this dissertation is structured as follows. In the next chapter, I outline relevant literature on three aspects of program comprehension which are the focus of this dissertation: (1) the various factors that influence comprehension, (2) the different forms of comprehension, and (3) the strategies that can support its development. In chapter 3, I present the theories that framed the design of our studies and the interpretation of our studies. I follow with a description of the instructional context in which our studies take place in chapter 4. Chapters 5 through 10 delineate the methodologies, results, and implications of several studies that delve into each of the three aspects of program comprehension. Finally, I reflect on my dissertation as a whole in chapter 11.

CHAPTER 2

RELATED WORK

In this chapter, we discuss related work in three key areas of program comprehension. The first section covers the factors contributing to comprehension, where we describe prior work in both societal and academic factors influencing computing instruction. The second section covers the different types of comprehension demonstrated by students, where we discuss the kinds of comprehension that can be revealed through different forms of assessments. The last section covers the strategies that support the development of program comprehension in young learners, drawing inspiration from existing strategies in computing and strategies from other discipline-based education research fields with a longer history of research in young learners.

2.1 Factors Contributing to Comprehension

We present two bodies of work that our studies on this topic build upon—equity in CS education and factors contributing to success in an introductory CS curriculum. We adopted a lens towards equity in investigating the factors critical to learning CS as many of these factors are tied to structural and societal inequities. Our studies extend prior work in the following ways. First, we identify performance differences across school-level and student-level factors in a formal school setting, thus providing a more holistic view of the (in)equities at the elementary school level. Second, we break down our results by specific learning goals and skills to provide a more nuanced picture of CT learning.

2.1.1 Equity in CS Education

The catalyst for research into the equity of CS education might be the seminal work by Jane Margolis [146], *Stuck in the Shallow End*, where she found an insidious "virtual segregation"

that maintains inequality. She identified structural barriers and curriculum and professional development shortcomings in the integration of CS education in Los Angeles. By tracing the relationships between school structures (e.g. course offerings and student-to-counselor ratios) and belief systems (i.e. teachers' assumptions about their students and students' assumptions about themselves), Margolis posits that the race gap in CS exemplifies the way students of color are denied a wide range of occupational and educational opportunities. Her research spawned numerous studies into the obstacles to equitable CS education, both in formal and informal settings.

Opportunities in Informal Settings

Informal CS education, such as after-school programs and summer camps, can be inaccessible and unaffordable to disadvantaged students. In an analysis of two nationally representative datasets of participation in out-of-school activities, Bouffard et al. [23] found that disadvantaged youth (youth from families with lower incomes and less education) were less likely to participate in out-of-school activities than their peers. If they participated at all, they were participating in *fewer* activities compared to their peers who came from wealthier and more educated families. They also found that Black and Hispanic youth participated with less intensity in some out-of-school activities, including lessons.

Furthering the connection between out-of-school participation and parental education, DiSalvo et al. [57] found that the search terms commonly used by parents to find such out-of-school opportunities yielded poor quality results, because they did not have the privilege of education and technical experience when searching for learning opportunities for the children. Among the 840 results from 42 computer related searches, the most powerful and free informal learning tools, such as Scratch and Alice, and free classes and tutorials, such as Khan Academy and Udacity, were completely absent. Instead, the results were filled with summer camps and fee based distance-learning programs. These activities may be

prohibitively expensive for students from low-income households.

Access in Formal Settings

Inequities persist into formal education, as well. In a US-wide survey by Google and Gallup [85], a majority of parents (84%), teachers (71%), principals (66%), and superintendents (65%) say that offering CS is just as important as, *if not more important than*, required courses like math and science. However, only 40% of principals report having at least one CS class where students can learn programming or coding. In addition, male students were more likely than female students to be told by a teacher that they would be good at CS (39% vs 26%). Likewise, Black students (47%) were less likely than White students (58%) to have classes dedicated to CS at the school they attend [84].

Studies at the school district level have found similar trends of inequity. In a study of CS course offerings in New York City public schools, Fancsali et al. [65] revealed that schools that did have CS courses served fewer Black and Latinx students and more White and Asian students. The schools offering CS also served fewer students in poverty and fewer students receiving special education services, and had higher average academic performance and graduation rates. Fancsali et al. [65] also identified a lack of funding, a lack of staffing, and competing academic priorities as barriers to offering CS or implementing it well. Additionally, in a study from the state of Florida, Century et al. [41] found an association between completing more Code.org courses and higher literacy, math and science scores.

2.1.2 Factors contributing to Success in Introductory Computing

Pea et al. [176] proposed the following cognitive prerequisites to programming from existing literature at the time: (a) math ability, (b) memory capacity, (c) analogical reasoning skills, (d) conditional reading skills, and (e) procedural thinking skills. Since then, there have been many studies analyzing the factors that contribute to success in a CS curriculum, most of

which have been at the college level.

At the university level, several studies have cited performance in other subjects as a factor leading to CS success. A study of first-year programming courses by Byrne et al. [35] revealed that scores on a math and science standardized test were strongly correlated with performance in the course, suggesting that CS may require a structure and approach with which science students have some experience and similar cognitive skills used in math. Wilson et al. [250] also found that the number of semesters of high school math courses were predictive of performance on a midterm in an introductory CS class.

Others have attributed success in introductory courses to various cognitive and metacognitive skills. Goold et al. [86] found personal learning strategies and problem-solving skills to be important to success. In a separate study [235], spatial visualization skills were also found to be associated with the success of students, suggesting that different navigational strategies may affect the way in which programmers are able to navigate programming code and form a conceptualization of its major features. Drawing from work by Leppink et al. [132], Morrison and colleagues developed a self-report measure of cognitive load specific to computer science [160] and explored subgoal labels to reduce cognitive load [161]. On the metacognitive side, Bergin et al. [16] discovered that students who perform well in programming used more metacognitive and resource management strategies than lower performing students, accounting for 45% of the variance in programming performance results. Additionally, a multinational study by Cutts et al. [51] indicate that students who have a strategic/algorithmic style of articulation carry on to be successful programmers.

Factors related to students' belief systems and prior experience have also been found to lead to success. Bergin et al. [15] found that programming performance was strongly correlated with intrinsic motivation, self-efficacy for learning and performance, and students' perception of their understanding. In addition, Wilson et al. [250] revealed comfort level in the course (i.e. willingness to ask and answer questions, anxiety level while working on assign-

ments, etc) and attribution to luck for success/failure to be predictive of course performance. As for prior experience, Bunderson et al. [31] attributed the high rate of female attrition in CS to the lack of previous experience with computers prior to entering the program. Further, virtually all prior experiences were beneficial for females, while only certain prior experiences correlated with success for males [228]. Hagan et al. [92] also discovered that students with experience in at least one programming language at the start of an introductory programming course perform significantly better, and that performance increases with the number of languages. Tying students' belief systems and prior experience together, Ramalingan et al. [189] and Wiedenbeck et al. [248] showed that self-efficacy for programming is influenced by prior experience and increases throughout an introductory programming course. Their results also revealed that the student's mental model of programming influences self-efficacy and that both the mental model and self-efficacy affect course performance.

By comparison, factors leading to success at the primary and secondary level are less explored. Studies that have been done at the middle-school level (ages 12-14), however, have shown that English and math ability, prior computing experience, and extracurricular technology activities contribute to success in CS learning [89, 187]. Lewis et al also found that 5th grade (ages 10-11) student performance on Scratch programming quizzes in a summer camp were highly correlated with their scores on a standardized math test [135]. Seufert presented a conceptual paper, bridging research across self-regulated learning and cognitive load [218]. Sands also argued for applying cognitive science and attending to cognitive load and working memory in the K-12 computer science classroom [211]. While not a full study, Sands' article recommended instructional strategies, such as modeling, worked examples, and peer collaborations, as scaffolds [211]. More recently in 2019, Mutlu-Bayraktar and colleagues published a systematic review of studies that explored cognitive load in multimedia learning [162]. By examining factors at the primary and middle school level, we can better understand the best ways to optimize instructional design to maximize opportunities for

developing complex thinking and problem solving skills [173].

2.2 Types of Comprehension

In this section, we discuss the ways previous scholars have assessed comprehension at the elementary school level, along with their benefits and drawbacks. Our studies build upon this body of work by (1) investigating the relationship between the types of comprehension exhibited through different forms of assessment, and (2) exploring how integrating student code into written assessments affect the types of comprehension they demonstrate. We first describe prior work on automated assessments, then traditional written assessments, and finally, interviews.

2.2.1 Artifact Analysis

There is a wealth of literature on automated assessment, including Scrape [252], Hairball [20], and Dr. Scratch [157]. Automated assessments have gotten more sophisticated over time, moving from counting instances of particular blocks [9, 252], to identifying correct and incorrect uses of code constructs [20], to analyzing higher order understanding [157, 192].

However, any technique focused on artifact analysis assumes that students understand the code they use in their projects. This is not necessarily true, as identified by Brennan et al. [25]. Students can use code in their projects that they don't truly understand, by copying exact code they were taught, remixing from the Scratch community, or receiving help from peers or instructors. Scratch project development is rarely performed in a strict exam-like setting, where young students are prohibited from speaking to peers or receiving help from the instructor. One study went so far as to record the level of help given by instructors in order to "subtract" it from understanding demonstrated by the artifact [20]. In addition, a student may understand a concept even if they did not choose to use it in their open-ended artifact. Written assessments or interviews are necessary to find out whether students

understand the concepts both included and not included in their code.

2.2.2 Written Assessments

Traditional written assessments are frequently used to assess student learning in Scratch, both in the school [152] and the extracurricular setting [135]. Researchers are also innovating on the form of written assessments, going beyond the pen-and-paper format. For example, Marinus et al. developed an assessment around Cubetto, a simplified version of the turtle LOGO programming task developed by Seymour Papert [148, 174]. However, very few validated assessments exist at the K-12 level. The validated assessments that do exist are designed for older audiences, such as college-level CS1 students [229], and middle school students students [10].

2.2.3 Interviews

Interviews provide a more nuanced and personalized way of assessing student learning. Brennan and Resnick found that through artifact-based interviews, they were able to identify the depth of a student's understanding of a particular concept, as opposed to only identifying whether they understood a concept [25]. While interviews can provide a more complete picture of student learning, they are limited by what students can remember about their projects and the project(s) selected for discussion [25]. Interviews are also very time-consuming, making them unrealistic for teachers who are already very time-constrained.

2.3 Strategies Supporting Comprehension

We outline two bodies of related literature that our work on strategies draws from: strategies in elementary reading and math and in computing education. We build upon this burgeoning body of work by: (1) developing and investigating TIPP&SEE, a learning strategy that

scaffolds the previewing and navigating of Scratch projects, and (2) exploring diagramming, commonly used in elementary math education and university computing education, as a strategy for young learners to scaffold decomposition.

2.3.1 Strategies from Reading and Math Education

One source of inspiration for our strategies is reading and math education. TIPP&SEE draws from previewing and navigating strategies in reading comprehension, while our diagramming strategy draws from similar strategies in elementary math education.

There are several existing evidence-based reading comprehension strategies that may have connections to programming, in particular previewing and text structure. Previewing [118, 145] helps students set goals for reading and activates prior knowledge. When reading example code containing a new concept, students might scan the code to quickly identify familiar and unfamiliar concepts. They could think about their prior knowledge of the concepts, predict how the new concept might work, and inspect the syntax of the new concept. In contrast, text structure [81, 249] prepares students to recognize disciplinary-specific text structures and use this knowledge to plan for reading and guide comprehension. In CS, programming languages and environments have specific structures that students must be able to discover to comprehend code and must be able to differentiate as they learn new languages and environments.

Similarly, the design of our diagramming strategy is shaped by diagrams used in elementary math. While there is a plethora of diagrams used in university computing, most of them represent more complex topics and were designed for more mature audiences. Diagrams in elementary math thus provide guidance for a more age-appropriate design. In math, diagramming facilitates learning by elucidating connections between concrete representations and symbolic manipulations, leading to positive outcomes for at-risk students and students with disabilities [108]. Diagrams come in many forms, from concept maps and vee

diagrams [170] to pictorial representations of word problems [243]. One example of such a learning strategy is the Draw-It Problem Solving Cycle, a strategy to support students with learning disabilities in planning and solving word problems [239]. When diagramming a word problem, this strategy encourages students to engage in four metacognitive steps: orient, plan, execute, and check.

2.3.2 Existing Strategies in Computing

Our strategies are also inspired by existing strategies in computing. There is a wealth of research in strategies at the professional and university level, taking many forms and addressing various aspects of computing.

At the professional level, software developers employ different strategies to solve a variety of problems. LaToza et al. defined several explicit programming strategies, human-executable procedures for accomplishing programming tasks, such as merging using Git or refactoring variables. They found that when using these strategies, developers were more systematic, organized, and more successful at a design task and a debugging task [126]. These strategies are most similar to our TIPP&SEE strategy, which describes a procedure for previewing and exploring a Scratch project. There are also software engineering strategies more analogous to our diagramming strategy. Perhaps the most (in)famous is the Unified Modeling Language (UML) diagram, frequently taught as the common language of software engineering [224, 237]. Through interviews with professionals, Marian Petre [182] found that most respondents did not use UML in practice, citing lack of context, the overhead of understanding notation, and issues of consistency across different diagrams as reasons. While this study was conducted with professionals, we can draw lessons from non-use in designing our diagrams and associated curriculum. Other visual representations include variants of flowcharts [164] and more recent developments like the "thinging machine", a diagrammatic language that encodes an abstract machine for creating, processing and exchanging

things [11], which have been designed to support the planning process in program writing.

At the university level, Lister et al. found that students who had better skills at reading, tracing, and explaining code tended to be better at writing code [138]. While Falkner et al. identified self-regulated learning strategies, such as diagramming, to be effective [64], Loksa et al. found that self-regulation during programming fluctuates greatly between students and that the instruction of self-regulation may require differentiation [141]. In addition, a multinational study [137] and its replication [49] demonstrated that students who sketched a diagram were more successful on code reading problems involving loops, arrays, and conditionals. Algorithmic visualizations have been used to support the instruction of dynamic processes in computing [72, 163, 181]. Turner et al. even developed tools to scaffold the use of UML in introductory computing classes [237]. With respect to *decomposition*, the concept for which our diagram will be designed, scholars have explored various approaches, such as problem-based learning [167] and explicit guided inquiry-based instruction [178] which led to learning gains in problem decomposition. Others have studied decomposition instruction that started very explicit and become progressively less explicit [116]. Another technique investigated was code templates [67], but researchers found that students decomposed problems around the templates, not the other way around, suggesting that students should be explicitly taught schemas for decomposition [39]. Nevertheless, strategies for university-age students (and their associated findings) are unlikely to directly translate to younger learners, who may not be able to regulate their own learning. Further, the context in which students learn computing in K-8 schools differ greatly from the university setting, making direct translation difficult, if not impossible.

Due to the relatively recent push for K-12 CS/CT instruction, prior work in strategies for students in that age range (ages 6-18) is thin by comparison. For students aged 15-18, strategies for debugging and code reuse were tested with moderate success; while students found the strategies valuable, many had difficulty regulating their choice of strategy [128,

119]. Some researchers have developed tools to support self-regulation, such as the PlanIt! tool from Miliken et al. [154]. A pilot study of PlanIt! with high school students showed that students learned to make more specific and actionable plans and that students enjoyed the guidance and affordances PlanIt! provided for more efficient program writing.

As for ages 6-14, Lee et al. developed the three-stage progression called Use→Modify→Create based on the idea that scaffolding increasingly deep interactions will promote the acquisition and development of CT [127]. Their approach provides more scaffolded, guided instruction for each concept, followed by a more open-ended project to engage students' interest and creativity. Students using Use→Modify→Create found Use→Modify tasks easier than open-ended tasks and felt more ownership over larger projects [142]. Another strategy is called PRIMM, which stands for Predict-Run-Investigate-Modify-Make [217]. PRIMM guides teachers in creating scaffolded activities in text-based programming languages to encourage learning. Finally, several researchers have provided guidance and research results on using Universal Design for Learning (UDL) instructional techniques in elementary computer science instruction, which posits that learning strategies specifically designed for some students often help many more and provide guidance and different ways to accommodate students with learning differences [96, 105].

CHAPTER 3

THEORETICAL FRAMEWORK

In this chapter, we delineate the theories that ground our approach to investigating program comprehension in young learners, as well as provide guidance for interpreting our results. We first discuss the pedagogical frameworks that frame the design and implementation of the curricula in this study. This is followed by a discussion of the Neo-Piagetian theories of cognitive development, which ground our understanding of the different factors that contribute to success in an introductory computing curriculum. We next describe the theories that frame both the different types and the depth of program comprehension, which shape the collection, analysis, and interpretation of our data. Lastly, we draw from theories of meta-cognition for the design of our strategies.

3.1 Pedagogical Frameworks

As with other subjects, including literacy [37, 236], computer science education researchers disagree on the appropriate level of structure to balance different learning goals. Papert [98], in his work on constructionism, posited that individuals learn best when they are constructing an artifact for public consumption, emphasizing self-directed learning. Constructionist curricula are more focused on, and have been shown to be successful at, increasing awareness and engagement, changing perceptions of computing, and building self-efficacy, especially for students from underrepresented communities in computing in informal contexts [111, 184, 144, 168, 238, 60, 32]. Such curricula were designed for the development of creative practices [25] and artifacts for public consumption [75, 88, 192], rather than the conceptual understanding we are seeking. Thus, such a self-directed approach may not lead to immediate understanding of the concepts underlying their artifacts [18].

More structured approaches, on the other hand, aim to develop mental models — an

understanding of specific CS concepts and how the code works [221]. While open-ended exploration may lead to the ability to create programs, prior work has highlighted difficulties in evaluating its success in teaching mental models because students do not always achieve conceptual understanding of their own code [18], especially compared to a more direct instruction approach [129]. On the other hand, an overly-structured approach can dissuade some students (especially females) from continuing in programming courses [245].

A more moderate approach is informed by seeking the Zone of Proximal Flow [12], a combination of Vygotsky's Zone of Proximal Development theory [242] with Csikszentmihalyi's ideas about Flow [48]. The Zone of Proximal Development describes what a student can learn with external support. In contrast, Flow is internally based; a student is in Flow when a task is not so challenging that they are overwhelmed, but not too easy for their skills that they are bored. The Zone of Proximal Flow refers to inquiry-based scaffolding that guides students through the Zone of Proximal Development so that they reach a state of Flow. Zone of Proximal Flow forms the basis of our instructional and curriculum design. Further, in our strategy development, we explore Use→Modify→Create [127], which facilitates the Zone of Proximal Flow. In this approach, students first engage with a concept in a structured project with provided code, and then make changes as they explore how the concept is applied.

3.2 Neo-Piagetian Theories of Cognitive Development

Piaget's theory posited that a child's cognition developed over time based on biological maturation and interaction with the environment [183]. Piaget's theory has been especially applicable to education in two important ways. First, it spurred the development of new teaching methods that capitalized on the exploratory activities of children themselves. Second, it strengthened the teaching of certain subjects, such as science and math, by cultivating and consolidating the basic thought structures of scientific and mathematical thinking [55].

Neo-Piagetian theories preserved the strengths of Piaget's theory while eliminating its

weaknesses [55]. They addressed the following weaknesses of Piaget's theory: (1) it did not sufficiently explain why development between each of the stages occurs, (2) it did not adequately account for the fact that some individuals move from stage to stage faster than other individuals, and (3) its proposed universal stages of cognitive development have been empirically disproven. We describe four Neo-Piagetian theories to provide some context to our studies: Case, Fischer, Commons, and Halford.

Case et al. [38] and Fischer et al. [68] proposed that development is not a straight progression through Piaget's main stages of development, but instead loops over all the stages, each involving their own executive control structures. However, Fischer et al. [68] argued that environmental and social factors drive development, not individual factors like Case et al. [38].

To account for environmental/social factors, we look into how gender and under-represented minority identity predict performance. Both identities can influence the learning opportunities available to students in their environments, which may be inequitably distributed.

To account for individual factors, we investigate the relationship between an individual student's reading comprehension, math proficiency, and cognitive skills and their performance on CT questions. Additionally, Commons et al. [44] proposed that developmental changes in more hierarchically complex tasks are attributed to the prior completion of simpler tasks. Simpler prerequisite skills needed to learn computing may include reading comprehension and math proficiency, as discussed in Chapter 2.

Furthermore, Halford [93] argued that people understand problems by mapping its parameters to mental models they already have. Existing mental models will differ between individuals based on their environment and prior experiences, supporting the need to analyze student-level factors.

3.3 Theories of Program Comprehension

Our approach to program comprehension stems from three broad sets of theories. The concept of a notional machine shapes our definition of comprehension and what it means to understand a program. Work from Storey and Schulte ground the different types of comprehension demonstrated by students, while Bloom's taxonomy and its computing variants frame our analysis for the depth of comprehension.

3.3.1 Notional Machines

The purpose of a notional machine is to explain program execution. It is a characterization of the computer in its role as an executor of programs in a particular language or a set of languages [222]. A notional machine:

- is an idealized abstraction of computer hardware and other aspects of the run-time environment of programs,
- serves the purpose of understanding what happens during program execution,
- is associated with one or more programming paradigms or languages, and possibly with a particular programming environment,
- enables the semantics of program code written in those paradigms or languages to be described,
- gives a particular perspective to the execution of programs, and
- correctly reflects what programs do when executed.

A notional machine is NOT:

- a mental representation that a student has of the computer (this is a mental model),

- a description or visualization of the computer, or
- a general, language- and paradigm-independent abstraction of the computer.

Based on these ideas, we broadly define "comprehension" as having a robust mental model of a notional machine. As such, the goal of our curricula was for students to form an accurate mental model of a program, as expressed through Scratch. More specifically in our studies, we define "comprehension" as being able to predict the outcome of a certain script run by the computer and to postulate which script produced a certain outcome.

3.3.2 Models for Different Types of Comprehension

Storey et al. [223] synthesized four models of program comprehension. These models include top-down [27], bottom-up [180, 220], systematic [140], and opportunistic comprehension [133]. She also further contextualized these models by differentiating them based on human characteristics, program characteristics, and the context for various comprehension tasks.

Schulte et al. [215] extended Storey's work through the Block model (Figure 3.1). The Block model introduces a duality between "structure" and "function" across three dimensions and four levels of specificity. Two dimensions fall under "structure"—text surface and program execution (data and control flow)—and function (goals of the program) is its own dimension. The dimensions and levels form a table, where each cell highlights one aspect of the understanding process. The cells are designed to be movable, thus allowing for the development of different learning paths. In the Block model, the ultimate goal is for students to build an abstract and general mental model automatically (i.e. unconsciously, so that cognitive resources are freed). The Block model generalizes program comprehension models by enabling students to build their own strategies to progress through the different cells.

Macro structure	(Understanding the) overall structure of the program text	Understanding the „algorithm“ of the program	Understanding the goal/ the purpose of the program (in its context)
Relations	References between blocks, e.g.: method calls, object creation, accessing data...	sequence of method calls „object sequence diagrams“	Understanding how subgoals are related to goals, how function is achieved by subfunctions
Blocks	'Regions of Interests' (ROI) that syntactically or semantically build a unit	Operation of a block, a method, or a ROI (as sequence of statements)	Function of a block, maybe seen as sub-goal
Atoms	Language elements	Operation of a statement	Function of a statement. Goal only understandable in context
	Text surface	**Program execution (data flow and control flow)**	**Functions (as means or as purpose), goals of the program**
Duality	***“Structure”***		***“Function”***

Figure 3.1: The Block Model

3.3.3 Bloom's Taxonomy and its Variants in Computing

The original Bloom's taxonomy defined six major cognitive categories: *Knowledge, Comprehension, Application, Analysis, Synthesis, and Evaluation* [19] (Figure 3.2). These categories were ordered from simple to complex and from concrete to abstract. Further, the original taxonomy represented a strict, cumulative hierarchy. Bloom's taxonomy was later revised to have a second dimension: the knowledge dimension [122]. The knowledge dimension consisted of the following categories: factual, conceptual, procedural, and meta-cognitive. With two dimensions, the revised Bloom's taxonomy was no longer a strict hierarchy, and instead

had multiple mastery paths.

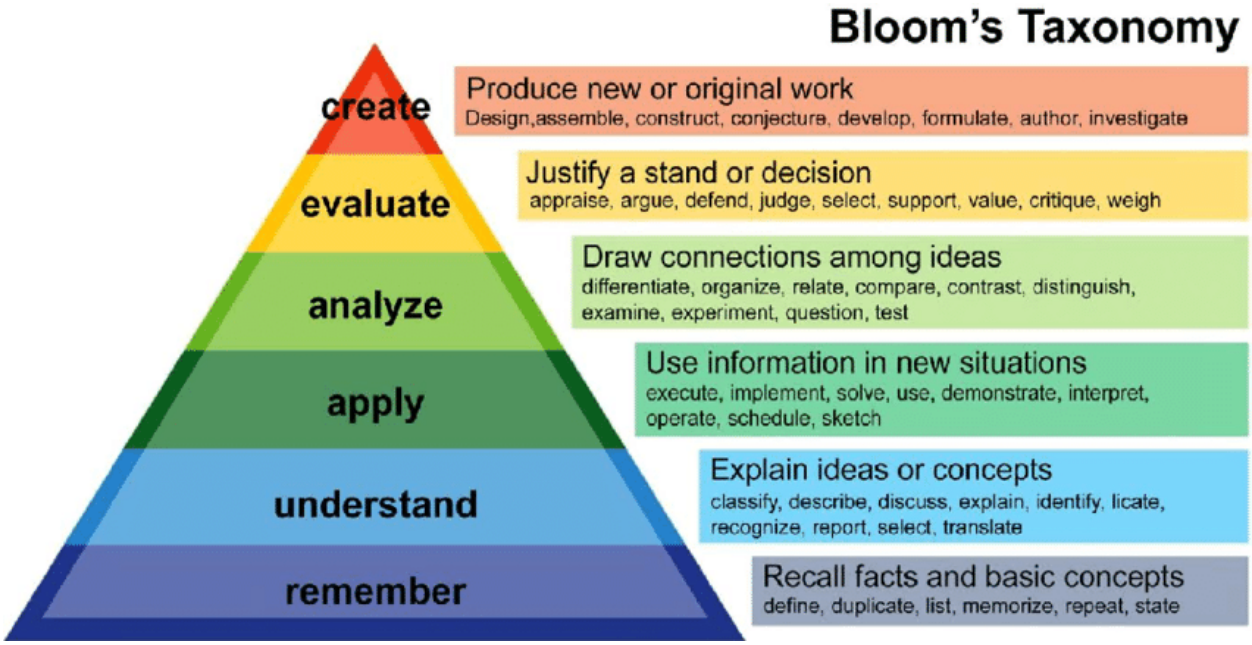


Figure 3.2: Bloom's Taxonomy

With its prominent use in computing, several scholars have proposed modifications to the Bloom's taxonomy to adapt it to have two aspects specific to computing: the ability to develop artifacts being a principal learning objective, and the centrality of studying process and problem solutions [80]. Johnson et al. [109] proposed that Bloom's taxonomy may need a "higher application" level, application informed by a critical approach to the subject (Figure 3.3).

Fuller et al. expanded upon Johnson's work and proposed the Matrix Taxonomy: a two-dimensional adaptation of Bloom's taxonomy (Figure 3.4). The two dimensions of the matrix encompass two different competencies: the ability to understand and interpret an existing product, known as the 'Interpreting' dimension, and the ability to design and build a product, known as the 'Producing' dimension. The levels in the 'Interpreting' dimension are *Remember, Understand, Analyze,* and *Evaluate*, while the levels in the 'Producing' dimension are *Apply* and *Create*. In our studies, the Use→Modify and Create tasks enable students to demonstrate their ability to produce code artifacts, with the Use→Modify task at the *Apply*

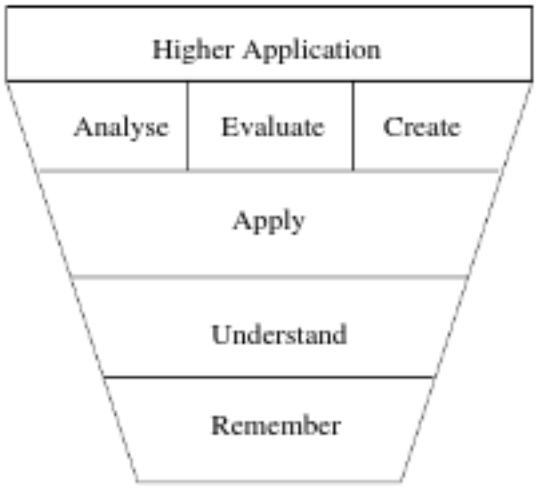


Figure 3.3: Bloom's Taxonomy with Higher Application

level and the Create task at the highest 'Producing' level. Worksheets and end-of-module assessments allow students to hone and demonstrate their interpretation abilities, both with whole projects and individual code snippets respectively.

3.3.4 SOLO Taxonomy

The structure of observed learning outcomes (SOLO) taxonomy arranges learning outcomes by structural complexity [17]. The SOLO taxonomy is comprised of five hierarchical levels of understanding:

1. Prestructural: Nothing is known about the subject or task.
2. Unistructural: One relevant aspect is known.
3. Multistructural: Several relevant independent aspects are known.
4. Relational: Aspects of knowledge are integrated into a structure.
5. Extended Abstract: Knowledge is generalized into a new domain.

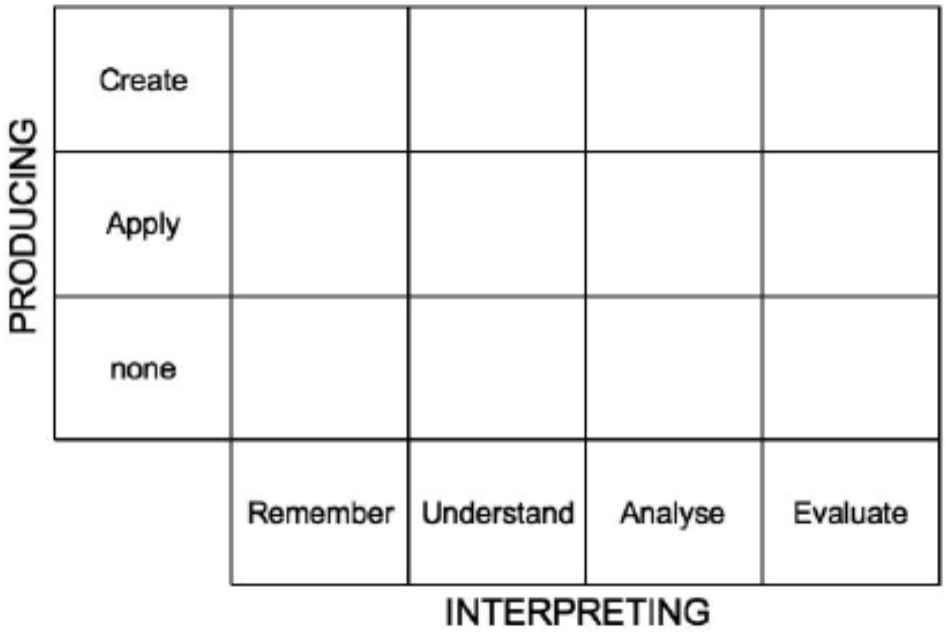


Figure 3.4: Matrix Taxonomy

SOLO is different from Bloom's taxonomy and its variants because it accounts for the context of the learner's response to what is being assessed. Its strength lies in encouraging a holistic approach that supports deep learning. Its weakness is that there is not much reported experience of using it for assessment in a range of subjects [80].

An example in the CS context is that expert programmers form abstract representations based upon the purpose of the code whereas novices form concrete representations based on how the code functions, inspiring a study by Lister et al [139]. They adapted the SOLO taxonomy to classify "explain in plain English questions" from students and educators:

1. Prestructural SOLO response: significant misconception of programming or is using a preconception irrelevant to programming

2. Unistructural SOLO response: "educated guess", a correct grasp of some but not all aspects of the problem

3. Multistructural SOLO response: an understanding of all parts of the problem, but

does not manifest an awareness of the relationships between these parts

4. Relational SOLO response: an integration of the parts of the problem into a coherent structure and uses that structure to solve the task

5. Extended abstract: an application beyond the immediate problem to be solved and links the problem to a broader context

They found that most educators actively sought to abstract beyond the concrete code while students did not, and that approximately one half of students in the top two quartiles demonstrated relational responses, while most students in the lower two quartiles demonstrated multistructural responses. Similarly, we used the SOLO taxonomy to categorize teacher interview responses in our study detailed in Chapter 10.

3.4 Theories of Meta-Cognition

TIPP&SEE, a mnemonic for students to remember the steps in exploring a new Scratch project, and diagramming are meta-cognitive strategies. Meta-cognition involves both self-regulation in learning and motivational aspects of learning. People who are meta-cognitive are reflective and constructive in the learning process, thinking about their own thinking and using this knowledge to guide both thinking and behavior [59]. These expert learners are strategic and purposeful: establishing goals, planning, self-monitoring, self-evaluating, giving self-feedback and correction, and motivating themselves toward the desired end [186]. In short, expert learners are meta-cognitive and strategic about their own learning.

However, strategic learning is an internal monitoring system, and is covert. To a less strategic learner, the "how" of learning is not obvious, and denies access to both process and content. To provide equitable learning opportunities, researchers developed and explored the explicit teaching of meta-cognitive strategies, a process for teaching students how to learn

within a content area, historically reading [118], writing [87], math [156], and content such as social studies [54] and science [54].

Learning strategies prompt and scaffold meta-cognitive thinking. Learning strategies are techniques, principles, or rules that enable a student to learn, solve problems, and to complete tasks independently [56]. The foundational idea of learning strategies is to support all learners in becoming independent by directly teaching them the processes that expert learners use. Meta-cognitive learning strategies make the covert activities of expert learners overt, enabling struggling learners to engage in, practice, and eventually internalize ways to guide their own thinking, motivation, and behaviors to meet learning goals. Strategy instruction promotes self-regulation in ways that manage information to optimize short-term memory and long-term storage and retrieval, thus automating procedural knowledge [218, 204]. Metacognitive strategies can facilitate problem solving, helping students to not only grasp the foundational knowledge and procedures, but to understand the conditions under which their knowledge will be useful for problem solving and innovation [66, 149, 47].

Mnemonic devices are one such scaffold. One type of mnemonic uses an acronym to cue memory and coordinate strategic thinking [216]. The mnemonic, TIPP&SEE, cues students to engage purposefully in a series of strategic steps and procedures that are foundational to higher order thinking skills [186] for computer science learning and problem solving

3.5 TPACK Model

TPACK is a framework for teacher knowledge for technology integration [120]. It describes the interactions of three critical bodies of knowledge: content, pedagogy, and technology. Pedagogical content knowledge (PCK) is the knowledge that teachers have about their content and the best ways to teach it [90]. Technological pedagogical knowledge (TPK) is the knowledge teachers have of how teaching and learning can change when technology is used in particular ways [120]. Technological content knowledge (TCK) is the knowledge teach-

ers have of how technology and content influence and constrain each other [120]. Lastly, TPACK is the knowledge teachers have of the representation of concepts using technology, the pedagogical techniques that use technology in constructive ways to teach content, and the difficulties students face when learning and how technology can help redress them [120].

The TPACK model has been used by scholars to analyze the different funds of knowledge of K-12 CS teachers. For example, Giannakos et al. used TPACK to examine the abilities and needs of a national sample of teachers in upper secondary education [82]. Vivian and Falkner utilize TPACK to study teachers' contributions to an online teacher professional development [241]. In the first study of Chapter 10, we also leveraged the TPACK model to characterize the knowledge expressed by teachers in their interviews.

CHAPTER 4

INSTRUCTIONAL CONTEXT

The research presented in this thesis took place in three different school districts: San Francisco Unified School District (SFUSD), Austin Independent School District (AISD), and Chicago Public School District (CPS). The work described in Chapters 5 and 8 were in SFUSD, the work described in Chapters 6, 7, and 9 were in AISD, and the studies described in Chapters 10 were in CPS. In this chapter, we outline the curriculum, as well as the design of the summative assessments for each school district.

4.1 Curriculum

4.1.1 San Francisco Unified School District

Students completed three modules in an introductory computational thinking (CT) curriculum in Scratch over the course of a year—the first module was an introduction to Scratch, the second covered events & sequence, and the third covered countable loops (see Table 4.1). The events & sequence module included a lesson on parallelism. Each module could take up to 5 class periods and consisted of an unplugged activity followed by projects that students had to *create* from scratch, i.e. without a starting project.

The Constructionist-inspired [98] curriculum was a modification of the Creative Computing Curriculum [45], where students were encouraged to develop artifacts for public consumption and to draw from the Scratch community. For all students in the study, this curriculum was their first formal in-school computing experience, though they may have had informal out-of-school exposure.

All teachers in the study underwent the same professional development held by SFUSD computer science school district leaders to teach this curriculum. Classroom size ranged from 13 to 31 students. Each lesson took 60-90 minutes and took place once every 1-2

Module	Project	Objective
Intro to Scratch	Scratch Surprise	Explore different Scratch blocks
Events & Sequence	10-Block Challenge	Create different combinations of the 10 listed blocks
	About Me	Create an interactive collage about yourself using events
Loops	Build a Band	Create a project about the music you like using loops

Table 4.1: Modules in the Curriculum

weeks, depending on the classroom. The primary language of instruction was English. Our study was conducted in the school district's second year of implementing this curriculum to minimize any logistical inconsistencies.

4.1.2 Austin Independent School District

Within a semester (approximately 5 months), students completed Scratch Act 1 [6], an introductory computational thinking (CT) curriculum modified from the Creative Computing curriculum [45] consisting of three modules: Sequence, Events, and Loops. Each module begins with Use/Modify project(s) and culminates in a Create project (see Table 4.2). All curriculum materials and assessments were available in English and Spanish.

All teachers underwent the same professional development designed by Prof. Diana Franklin and University of Chicago's CANON Lab school specialists, Donna Eatinger and Susan Krause, to teach the Scratch Act 1 curriculum to 4th grade students (ages 9-10). There were 7 to 18 consenting students in each classroom. Each lesson took 60-90 minutes and took place once every 1-2 weeks, depending on the classroom. Students were either taught in only English or a combination of English and Spanish.

Module	Project	Use-Modify-Create
Sequence	Name Poem	Use/Modify
	Ladybug Scramble	Use/Modify
	5-Block Challenge	Create
Events	Events Ofrenda	Use/Modify
	About Me	Create
Loops	Build a Band	Use/Modify
	Interactive Story	Create

Table 4.2: Scratch Act 1 Modules

4.1.3 Chicago Public School District

Over the course of a school year (approximately 9 months), students completed Scratch Encore [78], an intermediate CT curriculum that goes beyond sequence, loops, and basic conditionals. Like Scratch Act 1, modules in Scratch Encore start with a Use/Modify project, followed by a Create project. Modules also come in three different strands, Gaming, Multicultural, and Youth Culture, to facilitate culturally-responsive pedagogy.

All teachers took the same professional development, whether in-person or virtual in 2020, to teach the Scratch Encore curriculum to 5th-8th grade students (ages 11-15). Jennifer Palmer (Scratch Encore curriculum developer), Donna Eatinger (CANON lab school specialist), Merijke Coenraad (University of Maryland PhD student), and I conducted the professional development. There were 7 to 25 consenting students in each classroom. Each lesson took 45 to 60 minutes and took place once or twice a week, depending on the classroom. Students were taught only in English. From September 2020 to February 2021, all instruction in CPS was virtual, while from March to June 2021, instruction for elementary and middle schools was hybrid.

4.2 Assessment Design

Studies in SFUSD and AISD used assessments covering introductory CT concepts, namely events, sequence, and loops, while studies in CPS used an assessment covering an intermediate CT concept, decomposition by sequence.

4.2.1 Interpretation & Use Argument

In addition to statistical measures for validity, we employ an argument-based approach to assessment validation. Argumentation posits that the claims based on test scores be outlined as an argument that specifies the inferences and supporting assumptions needed to get from test responses to score-based interpretations and uses [113, 114]. Validation is therefore an evaluation of the coherence and completeness of those interpretations and uses and of the plausibility of its inferences and assumptions. A key distinction between this approach and statistical measures of validity is that in argumentation, it is the proposed score interpretation and uses that are validated, not the test or test scores.

The assessments designed for the studies in this dissertation are meant to be used within computational thinking curricula based in Scratch programming. More specifically, the introductory curriculum assessments are only suitable for introductory CT curricula where the concepts of events, sequence, and repetition are introduced through Scratch, such as Scratch Act 1 [6] and Creative Computing [24]. The Decomposition by Sequence assessment is even more constrained. It is only suitable for the Decomposition by Sequence module in Scratch Encore [78] because of curriculum-specific vocabulary used to introduce different CT concepts. For both assessments, students with higher scores can better perform the code reading and interpretation tasks in the assessments; we do not claim broader generalizability. For these assessments to be generalizable beyond these curricula, there will need to be other question types, such as non-programming or Bebras-style [52] questions. The development of validated assessments for elementary students is ongoing [175].

4.2.2 Introductory Curriculum Assessments

Upon completion of the Events & Sequence and Loops modules, students took a pen-and-paper assessment, consisting of multiple-choice, fill-in-the-blank and open-ended questions. Following the Evidence-Centered Design framework [155], assessments were designed based on domain analysis informed by the CS K-12 framework and K-8 learning trajectories [196]. Overarching learning goals were narrowed in domain modeling to identify specific knowledge and skills desired.

Prof. Diana Franklin, Bryan Twarek, William Marsland, and I designed the assessment questions. For face validity, questions were then reviewed by a larger group of practitioners and reading comprehension experts. Cronbach's alpha (α) was also calculated for internal reliability between questions on the same concept. Assessments were first administered in the 2017-2018 school year and improved upon for the 2018-2019 school year.

For the 2017-2018 school year, we designed 3 questions on events, 5 questions on sequence, 6 questions on loops, and 1 question on parallelism. After inspecting student responses, we excluded 4 questions from our analysis because their formatting resulted in spurious markings that made student responses unclear. We excluded 3 Explain in Plain English (EiPE) questions because some students *drew* the stage, instead of describing the code in words, which would result in ambiguous analysis. As a result, our analysis included a question on events, a question on parallelism, 2 questions on sequence (α=.78), as well as 5 questions on loops. One of the loops questions has 3 sub-questions, asking about the code in, before, and after a loop. Thus, there were a total of 7 items for loops (α=.82).

For the 2018-2019 school year, we revised the assessments from the previous year. Between the questions and sub-questions on both assessments, there were 5 items whose scores indicate a single underlying construct we call "events" (α=.72), 4 items that reflect a construct we call "sequence" (α=.7), and 9 items that indicate a construct we call "loops" (α=.85). A question with parallel loops was excluded in the reliability calculation because

its inclusion lowered the the reliability of the loops questions (α=.82), suggesting that it was not testing the same concepts as the other questions. An understanding of the concept of parallelism, instead of loops, was likely more crucial to answering this question correctly.

Additionally, for a more fine-grained picture, an exploratory factor analysis was conducted on student scores to characterize the underlying structure of our questions, i.e. which questions tested the same concept and the same level of Bloom's Taxonomy, a framework for classifying learning objectives [19]. Questions with multiple parts were treated as separate items. We excluded two questions from this analysis: a question on parallelism because of the Cronbach's alpha results, and an extra credit question on nested loops because that concept was not explicitly covered in the curriculum. A maximum likelihood factor analysis was conducted with six factors, the minimum number of factors that was deemed sufficient, and with the varimax rotation, which rotates the orthogonal basis so that the factors are not correlated. The minimum number of factors was determined using both the Kaiser's eigenvalue-greater-than-one criterion [112] and the scree plot elbow [40]. Based on the factor loadings from this analysis, we drafted a test blueprint (Table 4.3), with E&S and L short for Events & Sequence and Loops, respectively. We only included five of the six factors, as the last factor only accounted for one question. The remaining five factors accounted for 12 of the 18 questions included in the factor analysis.

4.2.3 Decomposition by Sequence Assessment

Prof. Diana Franklin, Jennifer Palmer, and I designed the Decomposition by Sequence module in Scratch Encore based on the Decomposition learning trajectory [193] with Scratch programming language-specific considerations. The end-of-module summative assessment targeted the first three learning goals of the module, namely that students should be able to:

1. decompose a sequence of events,

	Remember	**Understand**
Scratch Basics	E&S Q2, Q3 (Loading=1.07)	—
Events	—	E&S Q4a, Q4b (Loading=1.90)
Sequence	—	E&S Q6, Q7 (Loading=2.08); L: Q5a,b,c (Loading=1.90)
Loops	—	L: Q1, Q2, Q4 (Loading=1.90); L: Q5a,b,c (Loading=1.90)

Table 4.3: Test Blueprint with Concept & Bloom's Level

2. create scripts that will trigger the action of one sprite dependent on the action of another sprite, and

3. use sensing blocks to stop and start actions.

Questions were designed to address the first two levels of Bloom's taxonomy [19]: *remember* and *understand*. The assessment was administered on Qualtrics, with a combination of multiple-choice and drag-and-drop questions. For face validity, the questions were reviewed by a larger group of practitioners and researchers. A summary of the questions is shown in Table 4.4 and a more detailed description are in the Appendix.

To evaluate the internal reliability of questions covering the same learning goal, we conducted a Cronbach's alpha test. We did not find high reliability within questions for each learning goal (LG1: $\alpha = .42$, LG2: $\alpha = .45$, LG: $\alpha = .31$), likely due in part to our sample size [30]. With this low reliability and the limitations of Cronbach's alpha [227], we also conducted an exploratory factor analysis to uncover the underlying structure of our questions. A maximum likelihood factor analysis was conducted with two factors, the minimum number of factors that was deemed sufficient, and the varimax rotation, which rotates the

Learning Goal	**Remember**	**Understand**
1. Decompose a Sequence of Events	What are events?	Decompose a golf club swing
	What are actions?	
	What is an end condition?	
2. Create scripts with conditional interactions	When do you use `repeat until`?	Build the script for a golf ball
	When do you use `wait until`?	
3. Sensing blocks to stop/start actions		What event caused the needle to start moving?
		What event caused the balloon to pop?
		Build the script for a golf ball

Table 4.4: Decomposition by Sequence Assessment Questions Summary

orthogonal basis so that the factors are not correlated. The minimum number of factors was determined using both the Kaiser's eigenvalue-greater-than-one criterion [112] and the scree plot elbow [40]. The two factors corresponded to the two levels of Bloom's taxonomy, regardless of learning goals. The first factor (Loading = 1.30) accounted for all except one of the "Remember" questions, which asked students to identify two examples of "actions" in Scratch programming. The second factor (Loading = 1.09) accounted for all except one of the "Understand" questions, which asked students to identify the event that caused a balloon sprite to pop given two example scripts. This indicates that the two underlying constructs covered by this assessment are the abilities to recall and interpret concepts related to the decomposition of conditional interactions between sprites, not necessarily defined by specific learning goals.

CHAPTER 5

ANALYZING ELEMENTARY-AGE STUDENTS' COMPUTATIONAL THINKING PERFORMANCE THROUGH AN EQUITY LENS

With many countries worldwide integrating CS/CT instruction at the elementary/primary school level, it is paramount that we understand the interplay between program comprehension and developmental factors in young learners. To ensure that such instruction is equitable, it is even more important that we consider individual factors, such as skills developed at that age, and societal factors, such as the school environment.

In this study, we examine 4th grade (ages 9-10) learning outcomes from an introductory computational thinking curriculum across school performance, reading comprehension, math proficiency, race/ethnicity, and gender. School performance has been shown to be a proxy for the race, income, and parental involvement of their students [102, 190] and related to resources and teacher turnover rates [29]. Together with the lack of women and under-represented minorities in computing, disparities in reading comprehension and math proficiency are also fairly well documented [58, 1].

We pursue the following research questions in this study:

- How does school performance influence performance in the introductory CT topics—events, sequence, & loops?
- How do per-student factors (reading, math, gender, race/ethnicity) predict performance in CT topics? How do their relative effects change based on the learning goal within each topic?

In this study, the curriculum had already been used in San Francisco Unified School District (SFUSD) for two years before we conducted this study so I was not involved in the

curriculum development or the teacher professional development. I was involved in deciding what kind of schools to recruit for the study, but participant recruitment was led by two former SFUSD computer science leaders, Bryan Twarek and Bill Marsland. I led the study design, assessment design, and data analysis for this study.

5.1 Methods

5.1.1 Participants

The participants in our study were 296 4th grade (ages 9-10) students from a large urban school district, distributed between a high-performing school, 2 mid-performing schools, and a low-performing school. School performance levels were designated by the school district based on characteristics of both students (e.g. percentage of minority students, English language learners, students with special needs, students in poverty, etc) and teachers (e.g. years of experiences, turnover rates, etc).

Student gender was split almost evenly between male and female[1]. The participant ethnic breakdown was 32.9% Asian, 28.8% Hispanic/Latinx, 9.5 % White, 8.3% Pacific Islander, and 6.3% Black. The remaining students did not report.

5.1.2 Data Analysis

We performed two sets of data analyses on two summative assessments administered in an introductory CT curriculum. Question descriptions and scoring scheme are depicted in Table 5.1. For a bird's-eye view, we first analyze performance across school levels. For a more detailed view, we compare across per-student factors—reading comprehension, math proficiency, gender, and under-represented minority (URM) status.

1. Students in our study only identified as either male or female. No students identified as any other gender.

Question	Scoring Scheme	Max Score
Q1 Events Starting 1 Script	2pts/correct; -1pt/incorrect	4pts
Q2 Events Starting Parallel Scripts	2pts/correct; -1pt/incorrect	4pts
Q3 Repeat Iteration Count	1pt/correct	1pt
Q4 Loop Unrolling	1pt/correct	1pt
Q5 Repeat Blocks vs Iterations	2pts/correct; -1pt/incorrect	4pts
Q6a Code in Loop	2pts/correct; -1pt/incorrect	4pts
Q6b Code Before Loop	2pts/correct; -1pt/incorrect	2pts
Q6c Code After Loop	2pts/correct; -1pt/incorrect	2pts
EC Nested Loops	1pt/correct	1pt

Table 5.1: Question Description and Scoring Scheme

Comparison across School Performance

Since there were 2 mid-performing schools in this study, we selected the one with three classrooms taught by the same teacher for a better comparison with the high- and the low-performing schools, for a total of 204 students. With this large sample size, the power of all tests was at least 80%.

This quasi-experimental analysis followed the hierarchical *CRH-pq(A)* model. The linear model is as follows:

$$Y_{ijk} = \mu + \alpha_j + \beta_{k(j)} + \epsilon_{i(jk)} \tag{5.1}$$

where:

- Y_{ijk} is the question score for the i^{th} student in classroom k within school j,
- μ is the grand mean of the question score,
- α_j is the effect of school j,
- $\beta_{k(j)}$ is the effect of classroom k within school j,

- and $\epsilon_{i(jk)}$ is the error effect associated with Y_{ijk}.

The independent variable in this analysis was the school performance level, with classrooms nested within them. Both the school performance level and individual classrooms were fixed factors. The classrooms in our study were of different sizes, so we randomly sampled classrooms of 18 students (the smallest classroom size in our study) and ran the linear model based on the sampled classrooms. This process was repeated 1000 times, and the average of the linear model outputs over all iterations was calculated; the mean of the outputs was used.

Because there are three schools, our analysis was performed in two steps to find statistical significance. First, an ANOVA F-test was used to find whether there are any statistically-significant differences between schools. Then, a Fisher-Hayter Post Hoc test was performed pairwise on the three pair choices to determine which pairs' result differences were statistically significant. Both tests provide p values — $p < 0.05$ is statistically significant.

The eta squared (η^2) effect size was also calculated. η^2 measures the proportion of the total variance in a dependent variable (DV) that is associated with the membership of different groups defined by an independent variable (IV) [43]. For example, if an IV has a η^2 of .25, that means that 25% of a DV's variance is associated with that IV.

Comparison Across Per-Student Factors

Complementing our analysis across school performance, we also compare four per-student factors: reading proficiency, math proficiency, gender and under-represented minority (URM) status. Multiple regression analysis was used (1) to see if any of these factors were predictive of performance on the different questions, and (2) to compare the effects of the predictive factors. To be included in this analysis, students needed to have provided information on *all* factors, resulting in a total of 189 students.

Logistic regression was used for questions where the answers followed a binomial distri-

bution, i.e. there was only 1 correct answer (Q3, Q4, Q6b, Q6c, EC). We estimate the model as follows:

$$log\left(\frac{score_s}{1-score_s}\right) = \beta_0 + \beta_1 reading_s + \beta_2 math_s + \beta_3 gender_s + \beta_4 URM_s \tag{5.2}$$

Similarly, log-linear regression was used for questions where the answers followed a Poisson distribution, i.e. there were multiple correct options so the regression was done on the number of correct options chosen (Q1, Q2, Q5, Q6a). We estimate the model as follows:

$$log\left(score_s\right) = \beta_0 + \beta_1 reading_s + \beta_2 math_s + \beta_3 gender_s + \beta_4 URM_s \tag{5.3}$$

Reading and math proficiency scores were both normalized and dummy variables were assigned for gender (1 for female, 0 for male) and URM status (1 for URM, 0 for non-URM). We initially ran regression models with interaction terms between reading scores, math scores, and URM status, none of which were statistically significant and were therefore dropped from the model.

Reading Proficiency Analysis

Out of the 296 participants, 231 of them had Scholastic Reading Inventory (SRI) assessment scores. The SRI assessment measures reading skills and longitudinal progress on the Lexile Framework for Reading [131]. The SRI Technical Guide defines lexile score ranges for four proficiency levels; the ranges for 4th-grade are shown in Table 5.2 [2]. To identify inequities across the different proficiency levels, the ANOVA F-test was used.

To account for the imbalance across the different proficiency levels, Type III Sum of Squares was used. If the overall F-test was statistically significant, the Tukey-Kramer Post Hoc test was performed on each pair of reading proficiency levels to determine which pairs' result differences were statistically significant.

Proficiency Level	SRI Lexile Score
Below Basic (Sig. Below Grade Level)	< 540
Basic (Below Grade Level)	540-739
Proficient (At Grade Level)	740-940
Advanced (Above Grade Level)	> 940

Table 5.2: 4th Grade Reading Proficiency Levels

Math Proficiency Analysis

Out of the 291 participants, 285 of them had Smarter Balanced Assessment Consortium (SBAC) math scale scores. Designed based on the US Common Core State State Standards [4], the SBAC math assessment assesses students' knowledge of important mathematical facts and procedures and their ability to apply that knowledge in the problem-solving [3]. SBAC defines 4 proficient levels based on different score ranges. Table 5.3 shows the ranges for 4th grade [5]. To identify inequities across the different proficiency levels, we used the same analysis procedure as the reading score analysis.

Proficiency Level	SBAC Math Scale Score
Novice (Sig. Below Grade Level)	< 2411
Developing (Below Grade Level)	2411-2484
Proficient (At Grade Level)	2485-2548
Advanced (Above Grade Level)	> 2548

Table 5.3: 4th Grade Math Proficiency Levels

Non-Academic Factors

Gender was fairly evenly split among the students, with 144 students who identify as male and 152 students who identify as female. As for race/ethnicity, 100 of the students in our study identified as Asian, 81 identified as Hispanic/Latinx, 30 identified as White, 25

identified as Pacific Islanders, and 19 identified as Black. 41 students declined to state or provided no information. For our analysis, we categorized students based on whether they identified as a race/ethnicity that was under-represented (Hispanic/Latinx, Pacific Islanders, Black) or well-represented (Asian, White) in computing. To compare across genders and URM status, we used the ANOVA F-test with Type III Sum of Squares to account for the imbalance.

5.2 Results

We present three sets of results to better understand the influences of student performance in computer science. First, we present results across schools of different academic performance to understand whether current instruction serves the goal of equity. We then explore student-level factors, academic and non-academic, to explore potential sources of inequity.

In each section, we present overall results across assessment questions. We then provide detailed results for questions that illustrate the types of questions on which students performed similarly or differently across that attribute. Finally, we discuss potential causes or implications of the results.

5.2.1 Inequities Across Schools: The Big Picture

We begin by comparing overall performance across high-, mid-, and low-performing schools. Broadly, students in high-performing schools showed a good understanding of events and loops. 99% of them knew the number of iterations a repeat loop performs (Q3), 70% could see the relationship between the loop and equivalent sequential code (Q4), and more than 80% of them understood the order of blocks in a loop compared to blocks before and after the loop (Q6). Only two concepts, parallelism (Q2) and nested loops (EC) were beyond their grasp.

However, our results revealed that students at mid- and low-performing schools exhibited

a much shallower understanding of loops. While most could specify how many times a repeat loop will iterate, fewer than half could identify the unrolled equivalent of a repeat loop and identify both constructs that repeat actions (repeat loop and sequential code). Comparing between school levels, there were statistically-significant differences between the high- and mid-performing schools on questions which asked about advanced loop concepts (Q5, Q6, Q7; see Figure 5.1). Students in the mid- and low-performing schools performed differently on questions on events, parallelism and advanced questions on loops (Q1, Q2, Q7, EC). Finally, students in the high-performing school outperformed students in the low-performing school on all questions with statistically-significant differences.

In the next two subsections, we present two sets of questions that highlight the staggering performance gaps between the different levels of schools (events and loops).

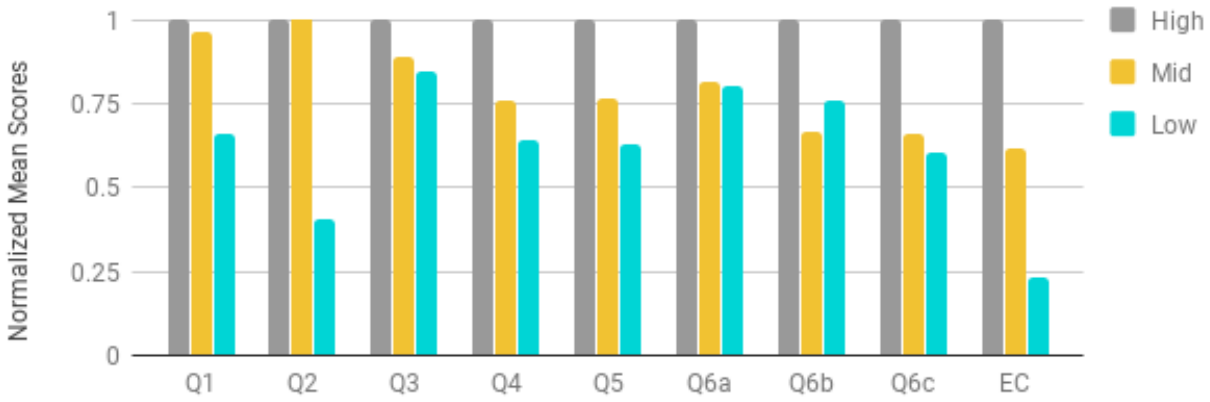


Figure 5.1: Overall Comparison Across Schools with Scores Normalized relative to the High-Performing School

Events starting Single & Parallel Scripts

There were two questions on events — Q1 covered events starting a single script (Figure 5.2), while Q2 covered events starting parallel scripts (Figure 5.3).

In Q1, students received two points for every correct script circled and lost one for any incorrect script circled, for 0-4 points.

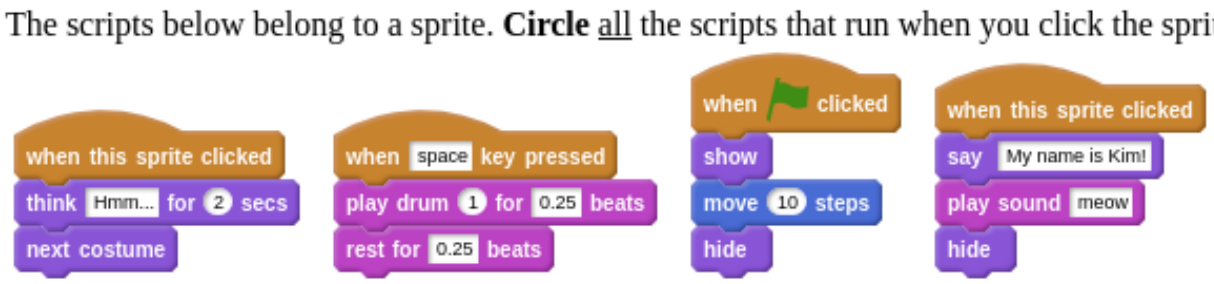


Figure 5.2: Q1 Events Starting a Single Script

The overall average score on Q1 was 2.49 (Figure 5.4a). Across all three schools, there was a statistically-significant difference ($F(2, 144) = 7.43, p < 0.001, \eta^2 = 0.0792$). Between pairs of schools, there were significant differences between the low-performing school and both the high- and mid-performing schools with a Fisher-Hayter Post Hoc ($p < 0.05$).

To better understand how students answered, student responses are categorized as: (1) NO correct - students who circled none of the correct answers, (2) BOTH correct & wrong - students who circled some some correct and some incorrect answers, (3) ONLY correct - students who circled correct (subset/all) but not wrong answers, and (4) ALL correct & NO wrong - students who circled all the correct answers and none of the incorrect ones. As shown in Table 8.3, students in the high-performing school circled correct options most frequently and provided the most complete answers, followed by the mid- and low-performing schools. Conversely, students in the low-performing school circled incorrect options (No Correct, Both Correct/Wrong) most frequently and were most likely to miss correct options, followed by the mid- and high-performing school.

Q2 assessed students' understanding of events across multiple scripts versus sequential events in one script (Figure 5.3). Students were asked to circle the true statements from the following:

a) Pico plays the drum 7 times THEN changes costumes 4 times.

b) Giga plays the drum 7 times THEN changes costumes 4 times.

Sch	Category			
	No Correct	Both Correct/Wrong	ONLY Correct	ALL Correct/NO Wrong
H	19.3%	7.9%	13.6%	59.1%
M	24.5%	15.9%	18.4%	41.1%
L	33.3%	16.7%	23.3%	26.7%
All	24.8	13.8%	18.0%	43.4%

Table 5.4: Q1 Qualitative Results

c) Pico plays the drum AND changes costumes at the same time.

d) Giga plays the drum AND changes costumes at the same time.

e) Pico and Giga both play the drum 7 times THEN change costumes 4 times.

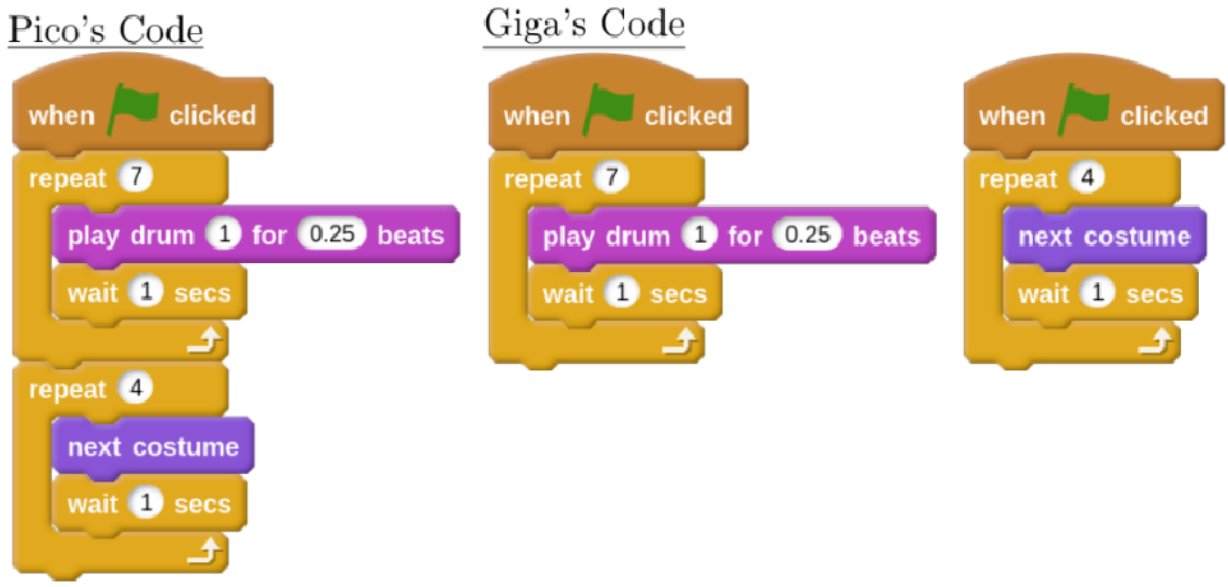


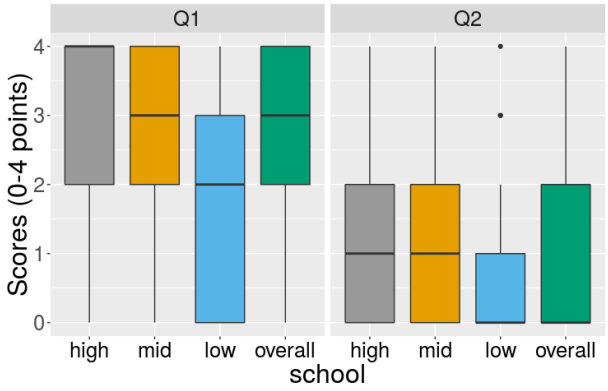
Figure 5.3: Q2 Events Starting Parallel Scripts

The correct answers were (a) and (d). Students earned 2 points for each correct answer circled and lost 1 point for each incorrect answer circled, for 0-4 points. Most students struggled with Q2, with an overall average score of 1.11 out of 4 points (Figure 5.4a). When broken down by school, the average scores were 1.31, 1.4 and 0.53 points for high-, mid-,

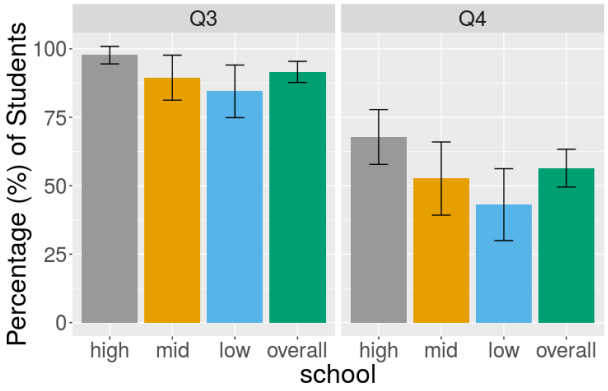
and low-performing schools, respectively. Across all three schools, there is a statistically-significant difference ($F(2, 144) = 7.82, p < 0.001, \eta^2 = 0.0845$). Between pairs of schools, there are significant differences between the low-performing school and both the high- and mid-performing schools with a Fisher-Hayter Post Hoc ($p < 0.05$).

64.4%, 70.1%, and 46.6% of students in high-, mid-, and low-performing schools, respectively, correctly identified Pico's sequential behavior. However, only 41.4%, 36.9%, and 35.8% of students in high-, mid-, and low-performing schools, respectively, circled Giga's parallel behavior.

Some very common errors include: 44.8% circled Giga having sequential behavior, 22.1% circled Pico having parallel behavior, and 53.9% circled the last option (both sprites have sequential behavior). Taking Q1 and Q2 in perspective, the higher frequency of answers with sequential behavior suggest that students may not understand parallelism as deeply as sequential execution in Scratch, with students in the high-performing school significantly outperforming students in the mid- and low-performing schools.



(a) Q1 Events Starting Single vs Q2 Parallel Scripts

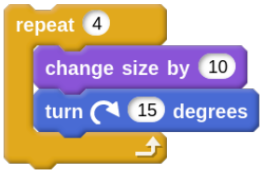


(b) Q3 Iteration Count vs Q4 Loop Unrolling Results

Figure 5.4: Questions highlighting Inequities across School Performance

Loop Functionality

Q3 and Q4 ask about basic loop functionality in two different ways. In Q3, students were asked how the number of times the loop would iterate, while in Q4, students were asked to correctly identify the unrolled version (Figure 5.5.



2. **Circle** the script that makes the sprite do the same thing as the loop above:

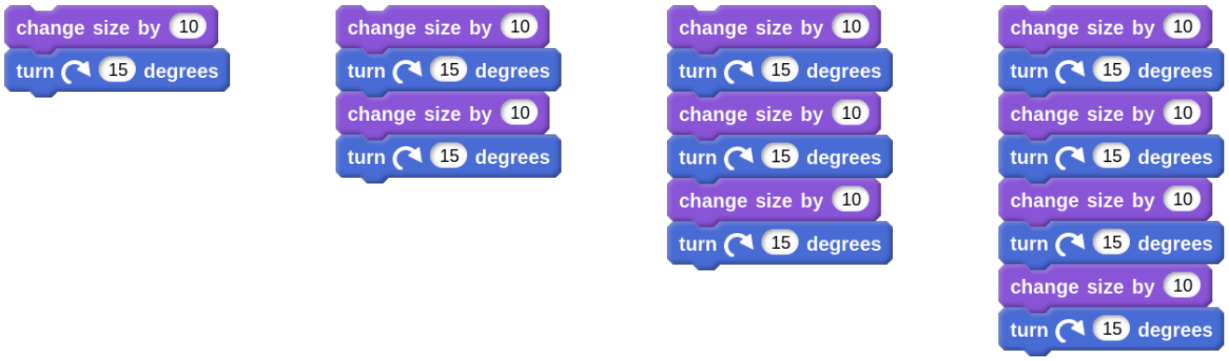


Figure 5.5: Q4 Loop Unrolling

Students performed very well on Q3. Almost all of the students from each school were able to answer correctly, with 98.9%, 88.3%, and 84.5% of the students in the high-, mid-, and low-performing schools, respectively getting the answer correct (Figure 5.4b). Comparing the differences in the number of students who answered correctly, we found a statistically-significant difference ($F(2, 144) = 5.05, p < 0.01, \eta^2 = 0.0431$) among the schools. A Fisher-Hayter Post Hoc pairwise analysis revealed a significant difference between the high- and low-performing schools ($p < 0.05$).

In contrast, students struggled on Q4, with only 56.4% overall answering it correctly. Within individual schools, 70.1%, 53.1%, and 44.8% of the students in the high-, mid-, and low-performing schools, respectively, answered correctly (Figure 5.4b). There is a

statistically-significant difference among schools for Q2 ($F(2, 144) = 5.25, p < 0.01, \eta^2 = 0.0539$), with only a significant difference between high- and low-performing schools from a Fisher-Hayter Post Hoc ($p < 0.05$).

When we put Q3 and Q4 performance in perspective, we see that while students are able to identify how many times a repeat loop is run, many students do not truly understand what that means. This implies a limited understanding of loop functionality, especially in the low-performing school.

School Performance Discussion

These results show that at all schools, many of the students are learning basics of core computer science concepts. However, the overall goal of equity is not yet being achieved. Students from low-performing schools are more likely to display a surface-level understanding of the concepts. This is shown especially with loop iteration count vs loop unrolling, in which students at all schools can answer how many times the loop iterates but have difficulty answering exactly how many times, and in what order, that loop causes them to run.

In order to address such inequity, curricular updates, teaching strategies, or learning strategies could be developed. To do so, however, we need to better understand why some students display so much more understanding than others.

5.2.2 Per-Student Factors

In order to gain more insight into differences in student performance, we looked at two categories of student factors — academic and non-academic factors. Academic factors included reading and math proficiencies, and non-academic factors included gender and URM[2] status, i.e. whether or not a student identified as a race/ethnicity under-represented in computing.

2. Under-represented racial/ethnic minorities include Black/African-American, Hispanic/Latinx, Native Hawaiian & Pacific Islanders. There were no students who identified as Native American in our study.

A summary of the regressions is presented in Table 5.5—the larger the absolute value of the coefficient, the more predictive that factor was of that question score.

In the following section, we present a deeper dive into the academic factors, comparing the influences of both reading and math proficiencies.

Question	Reading	Math	Gender	URM
Q1 Events Starting 1 Script	0.19*	—	—	—
Q2 Events Starting Parallel Scripts	—	.24*	—	—
Q3 Repeat Iteration Count	0.87*	—	—	—
Q4 Loop Unrolling	0.57*	—	—	-1.01*
Q5 Repeat Blocks vs Iterations	—	—	—	—
Q6a Code in Loop	—	—	—	—
Q6b Code Before Loop	0.75**	—	—	—
Q6c Code After Loop	0.57*	0.55*	—	—
EC Nested Loops	—	0.67*	-0.81*	-1.1*

* $p < .05$, ** $p < .01$

Table 5.5: Regressions to Identify Factors Predictive of Question Scores

Academic Factors

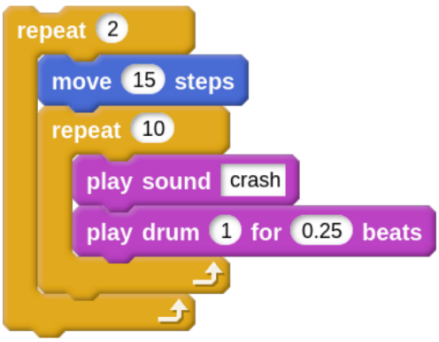
We highlight two questions where reading proficiencies were predictive: Q3 Repeat Iteration Count and Q4 Loop Unrolling. For Q3, reading proficiency was the only per-student factor that was predictive of performance, while both reading proficiency and URM status were predictive of performance in Q4 (see Table 5.5).

Comparing relative grade level performance, the differences between reading proficiency levels on both questions were statistically-significant (Q3: $F(3, 227) = 4.57, p < .01, \eta_p^2 = .056$; Q4: $F(3, 227) = 15.39, p < .01, \eta_p^2 = .17$). For Q3, there were statistically-significant differences between the advanced group and both the basic and below basic groups, and

between the proficient and the below basic group. For Q4, there were statistically-significant differences between all pairs except between the proficient group and both the advanced and basic group.

For math proficiency, we focus on two questions: Q2 Events Starting Parallel Scripts (Figure 5.3), which asked students to distinguish between sequential and parallel execution, and EC Nested Loops (Figure 5.6). Math proficiency was the only per-student factor predictive of performance in in Q2, while other non-academic factors were also predictive of performance in EC (see Table 5.5).

There were statistically-significant differences between math proficiency levels on both questions (Q2: $F(3, 281) = 12.27, p < .01, \eta_p^2 = .12$, EC: $F(3, 281) = 25.29, p < .01, \eta_p^2 = .22$). For Q2, there were statistically-significant differences between the novice group and both the proficient and advanced groups. As for EC, there were statistically-significant differences between all groups except for the proficient and advanced groups.

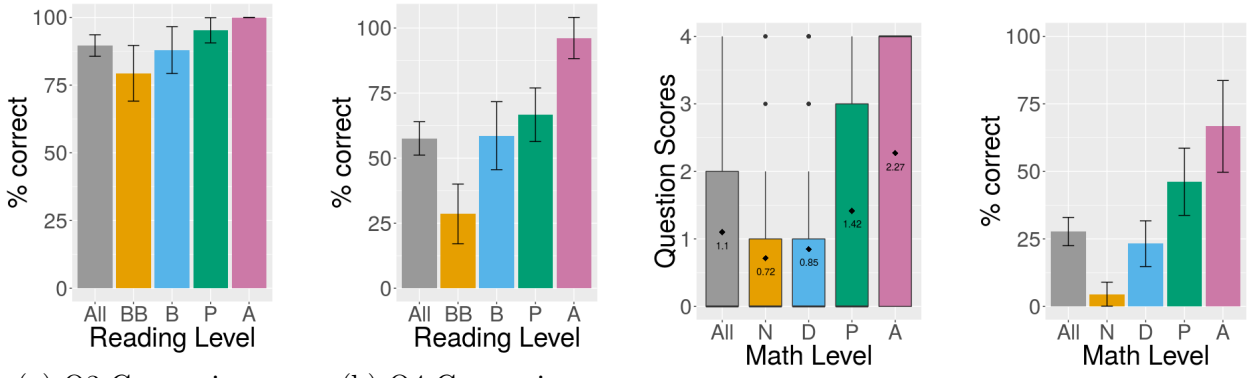


Extra Challenge: How many times will the "crash" sound play?

Figure 5.6: Extra Challenge Question on Nested Loops

Academic Factors Discussion

Reading proficiency was predictive of performance in 5 questions. All 5 questions asked about basic functionality and emphasized a text surface understanding over a program execution



(a) Q3 Comparison Across Reading Levels

(b) Q4 Comparison Across Reading Levels

(c) Q1 Comparison Across Math Levels

(d) EC Comparison Across Math Levels

Figure 5.7: Questions with Predictive Academic Factors

understanding—the two dimensions of structural understanding in the Block model (see Table 5.5). To answer Q1 correctly, students must be able to read and comprehend the event both in the question and in the answer scripts. Similarly, to answer Q6b and Q6c correctly, students must understand how to read the top-to-bottom order of Scratch scripts. Our results further support the idea that learning to program may depend on reading comprehension at several stages in the learning process. Just as in reading [185], it is not enough to decode the letters into words; to succeed, the student needs to make meaning of the sequences of words into instructions and the sequences of instructions into functions or programs.

In contrast, the questions where math proficiency was predictive emphasized a program execution understanding. For Q2, students needed to understand parallel execution, which most students struggled with. This result is not entirely surprising – the difficulties that students face while learning parallelism and a related concept, concurrency, are very well-documented [21, 22, 121, 134, 188]. This merits future work into the mental models younger learners have about parallelism, as well as the skills associated with building appropriate mental models. For the extra challenge question on nested loops, a understanding of loops may be related to an understanding of multiplication, i.e. multiplication can be conceptualized as repetitive addition. If students do not have a firm grasp on multiplication, they may

struggle to understand repetitive program execution.

Taking both academic factors into consideration, our results suggest that students must first be able to read and comprehend (1) the words in the blocks themselves and (2) the structure of scripts in Scratch before they are able to tackle a higher-order understanding of program execution, at which math proficiency becomes more predictive.

In terms of equity across different proficiency levels, our results reveal that the closest proficiency levels performed similarly, except in certain loop questions (Table 5.6 & 5.7). The performances of the significantly-below- and below-grade-level groups were significantly different on the loop unrolling question, and most of the advanced loop questions. The number of significant performance gaps only grows the further the proficiency levels are from each other, culminating in significant gaps on all questions between the significantly-below- and the above-grade-level groups. Significant performance gaps on loop questions, even between the closest groups, reinforce the need for improvement in its instruction.

While reading and math proficiencies were predictive of student performance on most questions (see Table 5.5), our results revealed non-academic factors—gender and URM status—to also be predictive, which will be explored in the following section.

Non-Academic Factors

There were only two questions where the non-academic factors were predictive — Q4 Loop Unrolling and EC nested loops.

On Q4 Loop Unrolling, URM status was the most predictive factor. Further comparing all students who had information on race/ethnicity, not just those who also had information on gender, reading and math, we found that students who identified as a well-represented race/ethnicity outperformed those who identified as an under-represented race/ethnicity ($F(1, 253) = 7.29, p < .01, \eta^2 = .11$; see Figure 5.8a)

The EC question on nested loops (Figure 5.6), which was not explicitly covered in the

Q	Reading Proficiency					
	SB*B	B*At	At*Ab	SB*At	B*Ab	SB*Ab
Q1				*		*
Q2	—	—	—	—	—	—
Q3				*	*	*
Q4	*		*	*	*	*
Q5	*			*	*	*
Q6a	*	*		*	*	*
Q6b	*			*	*	*
Q6c	*			*	*	*
EC		*	*	*	*	*

Table 5.6: Summary of statistically-significant differences between reading proficiency levels

curriculum, was the only question in which both gender and URM status were predictive (see Figure 5.5). Comparing all students with information on gender and race/ethnicity, regardless of whether or not they had information on reading and math proficiencies, an ANOVA F-test revealed that students who identified as male statistically-significantly outperformed students who identified as female ($F(1, 294) = 5.73, p < .05, \eta^2 = .019$; see Figure 5.8b). Similarly, students who identified as a member of a well-represented race/ethnicity statistically-significantly outperformed students who identified as a member of an under-represented race/ethnicity ($F(1, 53) = 38.77, p < .01, \eta^2 = .13$; see Figure 5.8c).

Non-Academic Factors Discussion

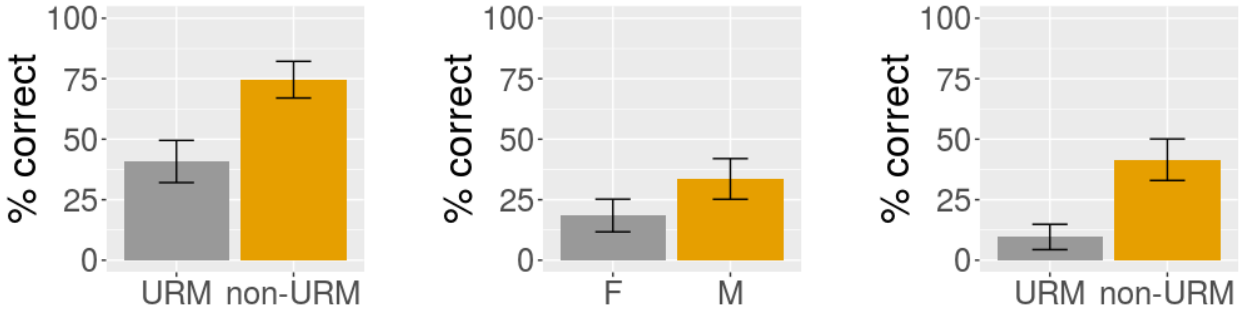
URM status was the most predictive of performance on two questions, Q4 Loop Unrolling and EC Nested Loops. Further, the interactions between URM status and both reading and math proficiencies were not significant. This suggests that there may be other factors associated with being part of an under-represented group that could be impacting their

Q	Math Proficiency					
	SB*B	B*At	At*Ab	SB*At	B*Ab	SB*Ab
Q1					*	*
Q2				*	*	*
Q3					*	*
Q4	*			*		*
Q5	*	*		*	*	*
Q6a	*			*	*	*
Q6b	*			*		*
Q6c	*	*		*	*	*
EC	*	*		*	*	*

Table 5.7: Summary of statistically-significant differences between math proficiency levels

performance in a computing curriculum. Among others, potential factors include spatial skills, which have been found to be tied to STEM achievement [147]. Males have been measured as having higher spatial ability, potentially due to the toys given to them in early childhood [136]. Socioeconomic status may also have an impact, as it is linked to academic skills developed in early childhood [97]. Investigation into other such factors associated with being part of an under-represented race/ethnic group and how they influence performance in a computing curriculum merits further study.

We can gain insight into potential factors involved in URM performance by focusing on the question where URM status was most predictive—the extra challenge question on nested loops. This question is unique because it is the only question that (1) covered material which was clearly beyond the curriculum and (2) can be predicted by gender and URM status. Prior research would hypothesize that this could be due to informal computing experience that well-represented students may have, a variable we did not control for [57, 107, 244]. Under-represented students may face barriers to accessing informal opportunities such as



(a) Q4 Comparison across URM Status

(b) EC Comparison across Genders

(c) EC Comparison across URM Status

Figure 5.8: Questions with Predictive Non-Academic Factors

cost, transportation, and the lack of parental technical familiarity [57, 61]. This further supports the need for in-school, formal computing instruction from a young age, before the disparities between well-represented and under-represented students grow too large.

5.3 Implications

This chapter presents an investigation into the factors associated with learning outcomes from a school district's implementation of an introductory CT curriculum for elementary-age students. Our analysis revealed worrisome differences across school performance levels, with students in the high-performing school significantly outperforming those in the low-performing school in all CT questions. School performance has been associated with the race, income, and parental involvement of their students, as well as resources and teacher turnover rates [29, 102]. Thus, the relatively poor learning outcomes from students in the mid- and low-performing schools indicates that disadvantaged students, the very students that such a wide implementation is aiming to reach, are getting the short end of the stick.

A deeper dive into per-student factors reveal how academic and non-academic factors can put some students at a disadvantage. Analysis at the student-level revealed reading comprehension to be predictive of most questions that emphasized a text surface understanding,

while math proficiency was predictive of questions which emphasized a more program execution understanding. Comparing relative grade-level performance, we found stark performance differences between students who perform below grade-level and those who perform at or above grade-level. Students who are not becoming proficient in reading and math are disproportionately students of color, ELLs, and students who live in poverty—students from communities traditionally marginalized from computing [1, 97]. If the associations between reading and math proficiencies and CS/CT learning remain unaddressed, efforts to increase access to formal computing experiences will fall short of addressing issues of equity.

In contrast, identifying as male and/or a member of a race/ethnicity well-represented in computing was predictive of performance in a question not explicitly taught in the curriculum. This indicates that there are still other factors that advantages such students, or conversely, factors that disadvantage students have identities under-represented in computing. Investigation into identifying these other factors merit further study.

These results have broad implications for researchers, instructors, curriculum developers, policymakers and other stakeholders in the movement to integrate CS/CT instruction at the pre-university level. Simply providing CS/CT instruction is not sufficient. This study has shown that it is not enough to provide equitable *access* to CS/CT instruction; we must also work towards equitable outcomes if we are to truly include students that have been historically marginalized in computing.

CHAPTER 6

INVESTIGATING THE ROLE OF COGNITIVE ABILITIES IN COMPUTATIONAL THINKING FOR YOUNG LEARNERS

All over the world, children are being exposed to the ideas of computer science (CS) and computational thinking (CT) at younger and younger ages as a result of nationwide movements to promote CS education [103]. Between the ages of 6 to 12, children develop the basic cognitive skills needed for learning [69]. Research into cognitive skills have a long tradition in related discipline-based education research fields, such as math [79, 158], science [158, 255], and reading [158, 169]. Few such studies have occurred in computer science. Further, the cognitive science research that does exist in computer science education is mostly with university-age or adult learners [202]. An understanding of the relationship between cognitive abilities and learning outcomes can help a new field such as elementary computer science create an appropriate developmental trajectory to guide standards [196, 193, 195, 194], inform development of curriculum and assessment, and set the instructional pace for optimal learning outcomes [117].

In this study, we investigated the cognitive abilities of fourth-grade students (age 9-10) who were introduced to CT either through a Use→Modify→Create (UMC) curriculum or the same curriculum with additional scaffolding from the TIPP&SEE learning strategy in a large urban school district in the United States. Use → Modify → Create is a learn-by-example approach, where students first observe and make small changes to sample code that demonstrates a new concept before incorporating the concept into a program they write from scratch [127]. TIPP&SEE is a metacognitive strategy that scaffolds the student exploration process in the Use → Modify step [208].

In this study, we explore the following research questions:

a) How are working memory, pattern recognition, and long-term retrieval associated with

performance on the CS/CT concepts: events, sequence, and loops?

b) How much does the TIPP&SEE learning strategy support students with differing cognitive abilities?

c) For which computational thinking concepts does TIPP&SEE support students with differing cognitive abilities?

In this study, I was involved in the design of TIPP&SEE and curricular materials. I was also involved in participant recruitment, which was led by our collaborator from Texas State University, Prof. Cathy Thomas. I also provided computing education support to the undergraduate students who were helping the teachers. I led the study design, assessment development, and data analysis.

6.1 Methods

6.1.1 Study Design

Teachers were randomly assigned to either the treatment or the control condition, resulting in five English-only and three bilingual English and Spanish classrooms in each condition. The eight teachers in the treatment condition were taught the TIPP&SEE learning strategy, which scaffolds student exploration of example programs for Use → Modify activities. Classrooms in the control condition were taught Scratch Act 1 and completed the same Use→Modify tasks but without the TIPP&SEE worksheets guiding them through familiarization with and exploration of the example code.

There were a total of 92 and 101 students in the TIPP&SEE (TS) and control (C) conditions, respectively. Due to the student population in this study, we tried our best to ensure that both conditions had as similar proportions as possible of students with: economic challenges (76.1% TS vs 89.2% C), multilinguality (designated "Limited English Proficiency"

by the school district; 27.2% TS vs 51.5% C), disabilities (17.4% TS vs 14.9% C), and below-grade level proficiencies in reading (58.7% TS vs 45.5% C) and math (59.8% TS vs 57.8% C). These factors were chosen based on research on what factors influence cognitive function in general and computer science performance in particular. More specifically, factors of economic challenge and poverty such as nutrition [191], stress [63], trauma [42], neighborhood-wide poverty [91], rural and urban poor environments [234], refugee status [42], family factors [91], food insufficiency, housing, and employment [53] all impact cognitive development and function. Similarly, economic challenges [65, 73, 146, 210], disabilities [106], English proficiency [187], and reading and math skills [89, 135, 209] all influence computer science performance.

6.1.2 Woodcock-Johnson IV Tests of Cognitive Abilities

The Woodcock-Johnson IV Tests of Cognitive Abilities (WJ IV) [213, 212] is a standardized, norm-referenced test of cognitive abilities that was developed based upon the Catell-Horn-Carroll theory of intelligence [70], and is appropriate for measuring cognitive abilities in persons from age two to 80+ years of age. These cognitive tests are not malleable to instruction, but are malleable to child development, maturity, and age. The purpose of these assessments is to gather information that allows comparison of an individual to others of similar age on important cognitive abilities. Together with other sources of information, these types of assessments contribute to the identification of exceptionalities, including "giftedness" and disability. When used ethically and properly, the WJ IV cognitive tests are less flawed, more theoretically grounded, and more fair than other methods of diagnoses [14].

For the purposes of this study, graduate students in school psychology who had been trained to administer this test with fidelity conducted individual assessments of participants in their school settings. Students in our sample were tested before or early in the computer science instructional instruction. All assessments were audio-recorded and were dual

scored. Disagreements in scoring were resolved through discussion, with resulting inter-rater agreement of 100%. Inter-rater agreement was supervised by the second author, a licensed diagnostician. Four subtests were administered to each participant.

Numbers Reversed

Numbers Reversed is a 34 item subtest of auditory, short-term working memory. For Grade 4, administrators begin with Sample Item A, and items are presented by audiotape. Participants listen to an increasing series of numbers that do not follow a predictable sequence. For example, Sample Item A includes 2 unrelated numbers, and item 34 includes 8. They are asked to repeat the numbers in reverse order. This test assesses auditory memory that requires both attention and manipulation (recoding) of new information and is a complex span task. This test was selected because research in math and reading has demonstrated that short-term memory is highly predictive of performance [46, 158]

Verbal Attention

Verbal Attention is a 36 item subtest of auditory, short-term working memory. For standardization, each item is presented using an audiotape included in the test kit. Participants listen to an increasing series of words that include animal names and numbers, and are then asked to answer a question. For example, item 9 includes 1 word, while item 36 includes a combination of 5 animal names and numbers. This subtest assesses the capacity of auditory memory, with a focus on attention. In this test, participants are asked to hold information in working memory, and use their executive search skills to identify the correct information to answer a question, assessing abilities in directing attention to needed information that is present in working memory. This subtest was selected because research in math and reading has demonstrated that short-term memory one of the strongest predictors of performance [46, 158].

Pair Cancellation

Pair Cancellation is a 49 item, 3-minute, timed subtest of accuracy in pattern recognition and scanning abilities. In this test, participants scan lines of pictures to identify specific patterns, for example, a picture of a dog, followed by a picture of a ball, and directed to circle each instance. This subtest measures aspects of visual/spatial perception, information processing speed, attention and concentration. This subtest was selected because of its relationship to basic reading skills such as rate and fluency [179] and math calculation [46].

Visual-Auditory Learning

Visual-Auditory Learning is a 7 item subtest of paired associates memory, one aspect of long-term storage and retrieval. In this subtest, participants are shown black and white rebuses and asked to associate each with a word/name. Initially, they are asked to name single rebuses, but as the assessment proceeds, are asked to "read" sentences of sequences of rebuses. This task represents learning, in that it requires short-term working memory to hold and organize the novel information, testing abilities in encoding, storage and retrieval of the new learning. This test requires students to organize, story and retrieve information during learning. The child must remember the word they are taught for each rebus in order to read the sentence. This subtest was chosen because of the importance of encoding to math and reading [179], and because research has demonstrated contributions of long-term memory in math and reading learning and outcomes [219], including for problem-solving [149]. Maximizing long-term memory is a goal of cognitive load theorists, so understanding children's abilities is important [172]. This test is less common in research, but is commonly used in assessment of children with and at risk for a learning disability.

6.1.3 Data Analysis

To understand how different cognitive abilities relate to CS/CT performance (our first research question), we first separated our data by groups of TIPP&SEE and control students because of previous work showing that TIPP&SEE was associated with better CT performance [208]. We then ran Spearman correlations between the cognitive abilities subtest scores and scores on their end-of-module assessments. We chose the non-parametric Spearman correlation because not all of the assessment scores met assumptions of normality and linearity. We provide ρ values for correlation strength and p values for statistical significance, with $p < .05$ as our threshold. We also interpreted ρ values based on guidelines from Hinkle et al [100], where $\rho = 0 - 0.3$ is very weak, $\rho = 0.3 - 0.5$ is weak, $\rho = 0.5 - 0.7$ is moderate, $\rho = 0.7 - 0.9$ is strong, and $\rho = 0.9 - 1$ is very strong.

To examine how much TIPP&SEE supports students with various levels of cognitive ability and in which CT concepts (our second and third research questions), we first ranked student scores according to classifications from the Woodcock-Johnson IV test manual (Table 6.1). The distribution of all student scores followed a normal distribution, with the number of students in each classification decreasing the farther the scores were from the mean. For some subtests, classifications on either tails of the distribution only had one student (see Table 6.2). As a result, we combined ranks of students with scores in the "Very Superior" and "Superior" into one "Superior" classification and students with scores in the "Very Low" and "Low" into one "Low" classification, in order to have cell sizes large enough for analysis. For the pair cancellation subtest, there was only one student in the "Superior" classification and was therefore excluded from analysis. While the correlations in the previous analysis allow for a more fine-grained picture, this classification allowed us to better describe students' relative standing among same aged peers and identify students most at-risk.

Scores 40 and below were considered to be outliers and were removed from analysis as they may not represent a fair example of their cognitive abilities. Possible reasons for outliers

include students reaching their ceiling before maintaining the minimum score required for each test and students being unable to pay attention to the task for an adequate amount of time (3 minutes). Outliers could also be due to the test environment. For example, if a student was not able to hear the audio the first time it was presented, it could result in them having a lower score because they could not hear the item, nor can the examiner repeat the items. Therefore, these extremely low scores may not be due to limitations in cognitive ability, but instead reflective of test administration issues, including audio difficulty, noise interference, or hearing issues. Further, significant under-performance may be a result of student disengagement with the test conditions. Low student motivation and interest or lack of rapport with the test administrator can influence test results. Additionally, the WJ IV Tests are culturally and linguistically loaded, meaning that children who have limited English proficiency may struggle with the instructions of the test and may score lower than their actual ability due to an inability to understand the test directions and task. Table 6.2 shows the total number of students, as well as students with disabilities, English Language Learners (as designated by the school district), and students with economic disadvantages in each WJ IV classification.

Comparing across conditions (TIPP&SEE and Control) and cognitive ability classifications (Low, Low Average, Average, High Average, and Superior), we transformed both aggregate and individual question assessment scores with the Aligned Rank Transform (ART), which enables non-parametric factorial analyses, before running an ANOVA F-test [99, 251]. A non-parametric transformation was chosen because of small cell sizes in the WJ IV classifications. Type III sum of squares was employed to account for unequal cell sizes and estimated marginal means were used for post-hoc comparisons. For statistical significance, we provide F and p values for both condition and WJ IV classification. For practical significance [125, 198], we also provide the partial eta squared (η_p^2) effect size. The effect size specifies the magnitude of the observed effect or relationship between variables [143]. η_p^2

measures the proportion of the total variance in a dependent variable (DV) that is associated with the membership of different groups defined by an independent variable (IV) [43]. For example, if an IV has a η_p^2 of 0.25, that means that 25% of a DV's variance is associated with that IV.

WJ IV	**Standard Score**	**Percentile Rank**
Very Superior	131 & above	98 to 99.9
Superior	121 to 130	92 to 97
High Average	111 to 120	76 to 91
Average	90 to 110	25 to 75
Low Average	80 to 89	9 to 24
Low	70 to 79	3 to 8
Very Low	41 to 69	0.1 to 2
Extremely Low	40 & below	Outliers

Table 6.1: Woodcock Johnson IV (WJ IV) Classifications

6.2 Results

To address our first research question, we first outline the results from our analysis of the correlations between cognitive ability scores and performance on question sets covering the same CT concepts. We next delineate the outcomes from comparing the computational thinking performance of students in different Woodcock-Johnson IV classifications, addressing our second and third research questions.

6.2.1 Correlations between Cognitive Abilities & CT Performance

We detail the correlations found between cognitive abilities and performance based on the test blueprint of questions and CT concepts developed through the exploratory factor analysis

(EFA) described in the methods section. EFA enables us to discuss questions covering the same CT concept as a collective; the following results are organized based on CT concept. A summary of the correlations is shown in Table 6.3.

Finding 1: Pair Cancellation, a measure of pattern recognition, was not correlated with better performance on any CT concept.

There were almost no correlations between scores on the Pair Cancellation subtest, which measures pattern recognition and scanning abilities, and scores on CT assessment questions. There was only a very weak correlation between Pair Cancellation scores and scores on one question on Events (Q4b) ($\rho = .232, p = .03$) for the TIPP&SEE students. However, given that none of the other questions were correlated and that classification over Pair Cancellation subtest scores were not statistically-significant (see Section 6.2.2), this very weak correlation on the single question likely does not imply a relationship between the skills measured by the Pair Cancellation subtest and learning the CT concepts covered in this curriculum.

Finding 2: Measures of working memory and long-term retrieval were weakly correlated with better performance on CT questions, with the correlations increasing with more complex CT concepts.

Scratch Basics

There were two questions on the Events & Sequence assessment that covered the basics of Scratch (Table 4.3). Q2 asked students to identify the last block in script, while Q3 asked students to identify all the scripts that ran when the sprite was clicked. There was a weak correlation between the Numbers Reversed subtest (a measure of working memory) and scores on Q2 for TIPP&SEE students ($\rho = .323, p = .0022$). For control students, there were very weak correlations between Q3 scores and both measures of working memory, Numbers Reversed ($\rho = .270, p = .0077$) and Verbal Attention subtests ($\rho = .277, p = .0063$). There was a greater correlation between Q3 scores and scores on the Visual-Auditory Learning

subtest, which measures long-term retrieval ($\rho = .431, p = 1.18 \times 10^{-5}$).

Events

Q4a and Q4b in the Events & Sequence assessment covered an understanding of events (Table 4.3). Looking at a Scratch stage with two sprites that resulted from a green flag click, students were asked to identify the script that ran for each sprite. For TIPP&SEE students, performance on both events questions were very weakly correlated with one of the working memory measures, Numbers Reversed (Q4a:$\rho = .218, p = .043$; Q4b:$\rho = .237, p = .027$). They were more correlated with the other working memory measure, Verbal Attention (Q4a:$\rho = .335, p = .0015$; Q4b:$\rho = .391, p = .00018$), and the long-term retrieval measure, Visual-Auditory Learning (Q4a:$\rho = .420, p = 5.09 \times 10^{-5}$; Q4b:$\rho = .416, p = 6.14 \times 10^{-5}$). For control students, only the long-term retrieval measure was very weakly correlated with scores on Events questions (Q4a:$\rho = .219, p = .031$; Q4b:$\rho = .235, p = .021$).

Sequence

The two questions on sequence from the Events & Sequence assessment (Q6 and Q7) asked students to describe the order in which the blocks in an example script would run. For TIPP&SEE students, there was a very weak correlation between one of the working memory measures, Numbers Reversed, and performance on Q6 ($\rho = .263, p = .014$). Control students showed a very weak correlation between the other working memory measure, Verbal Attention, and performance on Q7 ($\rho = .235, p = .021$). Similar to the questions on Events, they were more correlated with the Visual-Auditory Learning subtest in both TIPP&SEE (Q6:$\rho = .222, p = .039$; Q7:$\rho = .294, p = .0057$) and control conditions (Q6:$\rho = .223, p = .022$; Q7:$\rho = .361, p = .0003$).

Loops

Q1 from the Loops assessment asked students to identify the number of times an example loop would repeat. Q2 and Q4 from the same assessment asked students to unroll a loop, but with different answer choices. Q2 asked about a single-block loop repeating 4 times and had the answer choices of the block in the loop repeated 1, 2, 3, or 4 times. Q4 asked about a double-block loop repeating 3 times and had the answer choices of the two blocks alternating 3 times (the correct execution) and a script with the first block repeated 3 times followed by the second block repeated 3 times (a common misconception) [89]. Q5a, b, and c from the Loops assessment covered both sequence and loops, asking students to identify code that ran before, in, and after a loop.

For questions that only covered loops (Q1, Q2, and Q4), there was only one very weak correlation between Q4 scores and scores on one of the working memory measure, Verbal Attention, for TIPP&SEE students ($\rho = .240, p = .027$). In contrast, for control students, performance on Q2 and Q4 were weakly correlated with both working memory measures, Numbers Reversed (Q2:$\rho = .306, p = .0024$; Q4:$\rho = .238, p = .019$) and Verbal Attention (Q2:$\rho = .399, p = 5.75 \times 10^{-5}$; Q4:$\rho = .317, p = .0017$). Visual-Auditory Learning, a measure of long-term retrieval, was weakly correlated with all loops questions for control students (Q1:$\rho = .258, p = .011$; Q2:$\rho = .372, p = .00019$; Q4:$\rho = .381, p = .00013$). For questions covering both sequence and loops (Q5a-c), there were weak correlations between scores on these questions and measures of both working memory and long-term retrieval in both TIPP&SEE (Q5a:$\rho = .347, p = .0011$; Q5b:$\rho = .342, p = .0013$; Q5c:$\rho = .365, p = .00059$) and control conditions (Q5a:$\rho = .358, p = .00034$; Q5b:$\rho = .468, p = 1.52 \times 10^{-6}$; Q5c:$\rho = .360, p = .00032$).

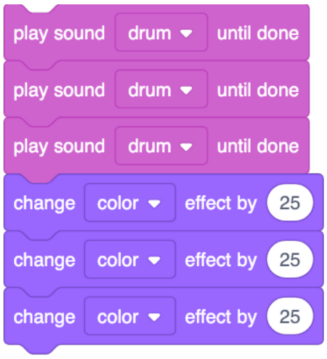


Figure 6.1: Common Misconception of Multi-Step Loop Execution

Discussion

With the exception of the Pair Cancellation subtest measuring pattern recognition, the correlations between cognitive skill measures and CT performance grew with the complexity of CT concepts, with more correlations of very weak magnitude ($\rho < .3$) for questions covering Scratch Basics, Events, and Sequence to majority of weak correlations ($\rho = .3 - .5$) for questions with loops.

It is worth noting that for the questions on events, TIPP&SEE student scores were correlated with both measures of working memory and the measure of long-term retrieval, unlike the control students, whose scores were only correlated with the measure of long-term retrieval. The questions on events were the only ones that used the vocabulary word "Stage" to refer to the area in the Scratch interface where students see the output of their code and showed students an image of the Scratch stage. Recalling domain-specific vocabulary may have loaded on students' working memory and long-term retrieval, independent of the computational thinking concept covered by that question. Acquiring disciplinary vocabulary is often a challenge that impedes learning for diverse learners in STEM content [233]. Unlike their typically developing peers, diverse learners may benefit from pre-teaching, explicit instruction, and increased exposure to learn new words well enough for them to be useful. Further, while related images may be paired with key information to enhance learning, visual information that requires interpretation can be more challenging for many learners [233].

Related to Scratch, students need opportunities to see and practice with the graphical representations in the platform, and to pair those with their meaning. In Scratch, the opportunities to use these for their own purposes should also enhance their vocabulary. For questions 4a and 4b, it is possible that neither the word "Stage" nor its image conveyed meaning to students, given limited exposure to this concept in their instruction [201].

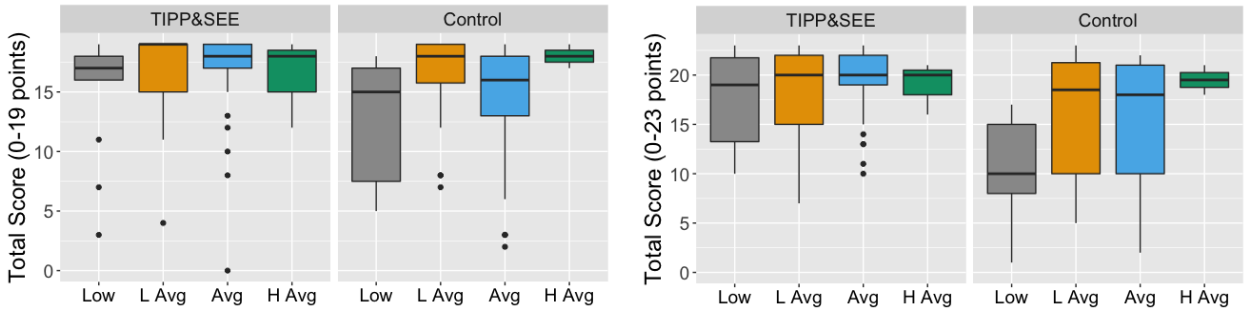
It is also noteworthy that for the questions on loops, there was only one very weak correlation between one working memory measure and one question in the TIPP&SEE condition, while in the control condition, both working memory and long-term retrieval measures were correlated with all but one question, which was still correlated with long-term retrieval. Further, for the most advanced questions that required knowledge of both sequence and loops, there were weak, with some bordering on moderate, correlations with both working and long-term retrieval measures in both conditions. While TIPP&SEE may have provided enough additional scaffolding to Use→Modify→Create for loops and easier CT concepts, more support may be needed for more complex CT concepts.

6.2.2 TIPP&SEE Support across Cognitive Abilities

We first report the results from comparing total scores from two end-of-module assessments across WJ IV classifications for each of the cognitive subtests. We follow with results from analyzing different sets of questions that cover different CT concepts to understand which concepts TIPP&SEE provides support for. Results from different questions are discussed collectively based on the CT concepts they cover (Table 4.3).

Finding 3: For both TIPP&SEE and control conditions, there was no statistically-significant effect of the Pair Cancellation subtest, a measure of pattern recognition, on CT performance.

Figures 6.2a and 6.2b illustrate the distribution of scores in each Pair Cancellation subtest classification within condition, in ascending order of classification from "Low" to "High Average". The "Superior" classification was omitted in the analysis as there was only



(a) Events & Sequence Assessment

(b) Loops Assessment

Figure 6.2: Performance across Pair Cancellation Classification

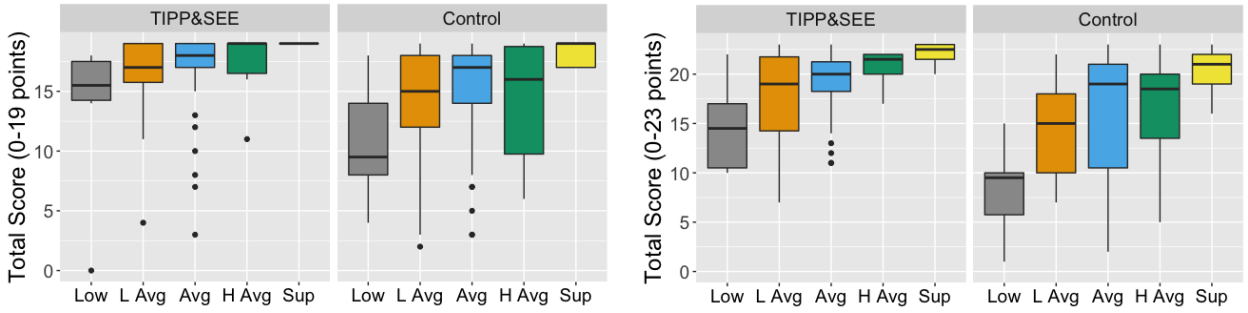
one student in that category.

Analysis across classifications for the Pair Cancellation subtest revealed no statistically-significant effect on total scores on either assessment (E&S: $F(1, 161) = 2.19, p = .0918$, L: $F(1, 159) = 2.36, p = .0739$). There was only a statistically-significant effect of condition on aggregate scores on both the Events & Sequence and Loops assessments (E&S: $F(1, 161) = 8.63, p = .0038, \eta_p^2 = .0509$, L: $F(1, 159) = 8.08, p = .0051, \eta_p^2 = .048$). Because of this, further analysis comparing Pair Cancellation classifications and CT questions was not conducted.

Finding 4: When using TIPP&SEE, students classified as having low scores on measures of working memory and long-term retrieval performed equal or better than control students classified as having average scores.

Numbers Reversed

Comparing across classifications based on the Numbers Reversed subtest, there were statistically-significant effects of both condition and classification on the scores from both the Events & Sequence (Condition: $F(1, 163) = 8.32, p = .0045, \eta_p^2 = .0486$; Classification: $F(4, 163) = 6.20, p = .00011, \eta_p^2 = .132$) and Loops assessments (Condition: $F(1, 161) = 14.97, p = .00016, \eta_p^2 = .0851$; Classification: $F(4, 161) = 8.99, p = 1.39 \times 10^{-6}, \eta_p^2 = .183$). Post-hoc analyses revealed no statistically-significant differences in performance between TIPP&SEE



(a) Events & Sequence Assessment

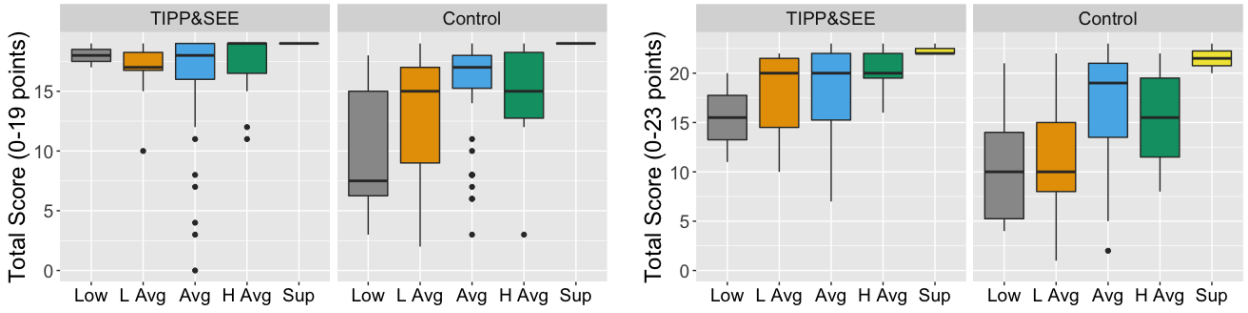
(b) Loops Assessment

Figure 6.3: Performance across Numbers Reversed Classification

students with low Number Reversed scores and control students with low average (E&S: $t = -.924, p = .357$, L: $t = -.70, p = .485$), average (E&S: $t = -.836, p = .405$, L: $t = -.345, p = .731$), high average (E&S: $t = -.997, p = .320$, L: $t = -.479, p = .633$), and superior (E&S: $t = -1.38; p = .170$, L: $t = -.253, p = .80$) scores on the Numbers Reversed subtest. Figures 6.3a and 6.3b depict the distribution of scores for each Numbers Reversed subtest classification nested within each condition, in increasing order from "Low" to "Superior".

Verbal Attention

Our analysis across Verbal Attention subtest classification showed statistically-significant effects of both condition and classification on aggregate scores on Events & Sequence (Condition: $F(1, 162) = 4.14, p = .043, \eta_p^2 = .0249$; Classification: $F(4, 162) = 6.59, p = 6.05 \times 10^{-5}, \eta_p^2 = .140$) and Loops assessments (Condition: $F(1, 160) = 9.49, p = .0024, \eta_p^2 = .0559$; Classification: $F(4, 160) = 7.67, p = 1.12 \times 10^{-5}, \eta_p^2 = .161$). Unlike in the previous working memory measure Numbers Reversed, TIPP&SEE students with low Verbal Attention scores did not perform as well as control students with average scores on the Events & Sequence assessment. Instead, they *out-performed* them, performing better than control students with low average ($t = -2.57; p = .011$), average ($t = -2.073; p = .039$), and high average ($t = -2.39; p = .018$) scores on the Verbal Attention subtest. Results were



(a) Events & Sequence Assessment

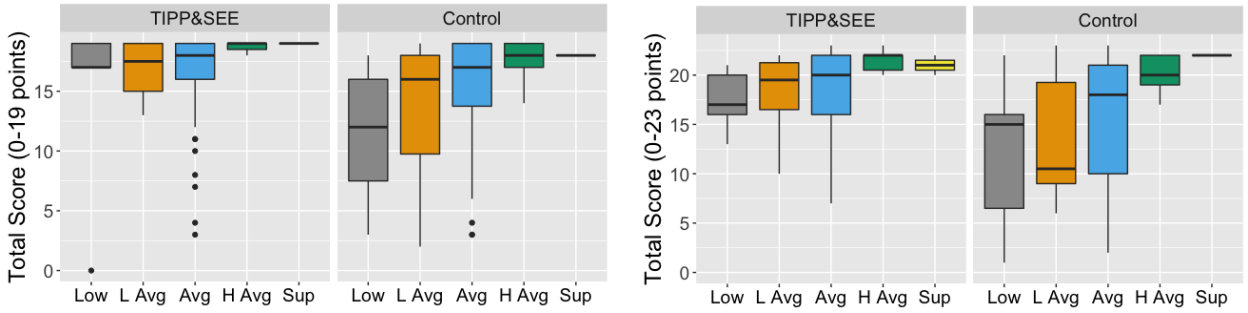
(b) Loops Assessment

Figure 6.4: Performance across Verbal Attention Classification

more similar to Numbers Reversed in the Loops assessment, with TIPP&SEE students who had low Verbal Attention scores performing as well as control students with low average ($t = -.867, p = .387$), average ($t = -.560, p = .576$), and high average ($t = -.640, p = .523$) Verbal Attention scores. Figures 6.4a and 6.4b show the distribution of scores for each Verbal Attention subtest classification for each condition.

Visual-Auditory Learning

In our analysis across Visual-Auditory Learning subtest classifications, there were statistically-significant effects of both condition and classification on performance on both Events & Sequence (Condition: $F(1, 164) = 5.21, p = .0237, \eta_p^2 = .0308$; Classification: $F(4, 164) = 5.41, p = .00041, \eta_p^2 = .117$) and Loops assessments (Condition: $F(1, 162) = 5.97, p = .016, \eta_p^2 = .0355$; Classification: $F(4, 162) = 5.77, p = .00023, \eta_p^2 = .12$). Similar to the working memory measures, post-hoc comparisons indicated that TIPP&SEE students with low Visual-Auditory scores performed as well as control students with low average (E&S: $t = -1.18, p = .239$, L: $t = -.976, p = .330$), average (E&S: $t = -1.01, p = .316$, L: $t = -.632, p = .528$), and high average (E&S: $t = -1.79, p = .0755$, L: $t = -.488, p = .626$) scores. Figures 6.5a and 6.5b portray the distribution of scores for each Visual Auditory-Learning classification in each condition.



(a) Events & Sequence Assessment

(b) Loops Assessment

Figure 6.5: Performance across Visual-Auditory Learning Classification

Finding 5: For questions on events, there was a statistically-significant effect of Visual-Auditory Learning classification, not condition, on CT performance.

We now turn our attention from aggregate performance to performance on specific concepts. There was only a statistically-significant effect of visual-auditory learning, a measure of long-term retrieval, on performance on both questions covering events (Tables 6.4, 6.5, & 6.6). There was no effect of condition on performance on either question (Tables 6.4, 6.5, & 6.6). This may be due to the use of the vocabulary word "Stage" to describe the graphical output of Scratch code, which could have relied on students' long-term retrieval. In contrast, comparisons across the two measures of working memory were ambiguous, where the two Events questions had divergent outcomes.

Finding 6: There were statistically-significant effects of both condition and Verbal Attention classification on performance on questions covering loops.

When contrasting across one of the working memory measures, Verbal Attention, we found statistically-significant effects of both condition and classification on performance on Loops questions (Tables 6.4, 6.5, & 6.6). This may be early evidence for TIPP&SEE support for the concept of loops and for the role of working memory in learning this concept. On the other hand, comparisons across the other working memory measure, Numbers Reversed, and the long-term retrieval measure, Visual-Auditory Learning, were mixed, with varying outcomes for each question.

Finding 7: Results for the rest of the CT concepts were inconclusive.

Results were mixed for the questions covering the other CT concepts, with disparate outcomes for questions on the same concept. Of the questions covering the basic syntax and semantics of Scratch, the effects of both conditions and cognitive score classifications were not consistent (Tables 6.4, 6.5, & 6.6). As for questions on sequence, there were no effects of both condition and classification on question scores in comparisons across Verbal Attention classification, while results were mixed for Numbers Reversed and Visual-Auditory Learning (Tables 6.4, 6.5, & 6.6). Lastly, outcomes were ambiguous for the questions combining sequence and loops for comparisons across subtests of both working memory and long-term retrieval (Tables 6.4, 6.5, & 6.6).

Discussion

Our analysis of TIPP&SEE support for students with differing cognitive abilities indicated that when using TIPP&SEE, students with low scores on cognitive tests perform similarly on summative, end-of-module assessments as students with average scores who underwent a less scaffolded curriculum. In some assessments, TIPP&SEE students with low cognitive scores even out-performed control students with average scores or performed as well as control students with superior scores, as was the case in our comparison across both the working memory measures, Verbal Attention and Numbers Reversed, on the Events & Sequence assessment.

Results were less definitive when questions were broken down by CT concept. There was a statistically-significant effect of long-term retrieval (Visual-Auditory Learning subtest), not condition, on performance on questions covering events only. In contrast, there were statistically-significant effects of condition and one measure of working memory (Verbal Attention subtest) on performance on questions covering loops. The rest of the CT concepts had mixed outcomes, with questions on the same concept having mismatched outcomes.

6.3 Discussion

We now return to our overarching research questions:

How are working memory, pattern recognition, and long-term retrieval associated with performance on the CS/CT concepts: events, sequence, and loops?

The correlations found overall were weaker than would be expected based on prior work which showed that cognitive abilities affect learning opportunities in math, science, and reading [79, 158, 169, 255]. This may be because the scaffolding in the curriculum, either through Use→Modify→Create alone or Use→Modify→Create with TIPP&SEE, supported students in learning CS/CT. While the correlations were smaller than expected, this study presents a critical first step in exploring cognitive capacity in the learning of CS/CT in young students.

As the CT concepts grew more advanced, the correlations increased between measures of cognitive skill and performance on CT questions. For the basic CT concepts of events and sequence, there were various correlations of very weak to weak magnitude between measures of working memory and long-term retrieval and performance on questions covering these concepts. With respect to a more complex CT concept covered in this curriculum, loops, TIPP&SEE student performance was only very weakly associated with one working memory measure for one question. With only one very weak correlation with only one question, it is unlikely that a relationship between working memory and TIPP&SEE students learning of loops. In contrast, the performance of control students were correlated with both measures of working memory, as well as long-term retrieval in all but the simplest loops question. Lastly, for both groups, measures of working memory and long-term retrieval were the most correlated with performance on the most complex set of questions requiring a combined knowledge of both sequence and loops. Taken together, this may be early evidence that this curriculum was manageable for all students with scaffolding from Use→Modify→Create for simpler CT topics such as events and sequence. This also implies that with increased

complexity, the burden on cognitive abilities might require additional scaffolding, such as TIPP&SEE, or adapted curriculum to remain accessible.

How much does the TIPP&SEE learning strategy support students with differing cognitive abilities?

There were more correlations between CS/CT performance and the Numbers Reversed subtest for students in the TIPP&SEE condition. Numbers Reversed can be considered a measure that focuses on working memory capacity. It is a complex span task in that the operation of reversing the sequence remembered requires active engagement to hold and manipulate the information, and reflects ability to control attention during tasks, an executive function. Weaknesses in these skills would impact an individuals' ability to follow multi-step and complex directions and the quantity of materials that could be managed at a time. Teaching strategies such as verbal rehearsal and visualization can support student learning. Further accommodations would include short, simple directions, visual cues, chunking of information to be learned, and monitoring performance.

For students in the Control condition, there were more correlations between CS/CT scores and the Verbal Attention and Visual-Auditory Learning subtests. The skills assessed by the Verbal Attention subtest, similar to the Numbers Reversed subtest, measure working memory capacity. Uniquely, it also examines skills in holding and finding information for needed purposes, and skills in focusing attention to sort distractors. The ability to update information and find it in a timely fashion is important. Performance on this subtest is predictive of academic performance, and students with weaknesses in these capacities might require additional repetition, limiting distractions, simplification of directions, and also strategy instruction, including mnemonics for learning rules, patterns, and lists of words [212]. This subtest was selected for its role in working memory, which is predictive of performance in other academic tasks. Further, learning, remembering, and retrieving, and using new verbal information may be important to success in learning CS/CT.

The Visual-Auditory Learning subtest may also be related to learning CS/CT. This skill requires associated memory, in which more than one type of information is learned. For example, in Scratch programming, new vocabulary and images are organized and stored together. The term "sprite" and the images of various sprites provide mental models of the concept of sprites. In this subtest, the novel information is both encoded and retrieved. For weaknesses in this skill set, rehearsal, overlearning, shorter and more frequent sessions, and the use of visual images may strengthen learning. Given the results across concepts, while correlations were weak, TIPP&SEE students appeared to be more impacted by short-term memory capacity and attention, while students in the Control group more often were impacted by skills in holding and "looking up" needed information for use, both in short-term working memory (Verbal Attention) and in the associative memory functions in the encoding process that facilitate both storage and retrieval. While this research is exploratory, it is possible that TIPP&SEE is providing the recommended rehearsal and scaffolded practice needed to perform the CS/CT tasks and supporting students.

We also found that on many tasks, students in the TIPP&SEE group with low scores on cognitive markers of short-term memory and long-term retrieval performed as well as their average peers on CS/CT tasks. Students who experience poverty and deprivation tend to have low scores on theses cognitive markers, which are often linked with academic underperformance [42, 53, 63, 91, 191]. Students who demonstrate a weakness in these areas may experience difficulty with developing strategies independently while studying, difficulty with vocabulary development, and difficulty simultaneously remembering a comprehension question and integrating previously learned information. TIPP&SEE can provide the strategic scaffold that "levels the playing field". This "leveling of the playing field" has been demonstrated with explicit teaching of meta-cognitive strategies in academic areas including reading, math, and science [59]. In this study, we see the potential for strategy instruction as an effective scaffold for young learners in CT/CS.

For which computational thinking concepts does TIPP&SEE support students with differing cognitive abilities?

Results were less clear when we took a more detailed look into specific CT concepts. Most questions had inconclusive results, with questions covering the same CT concept having inconsistent outcomes. While students with low cognitive scores were better served with TIPP&SEE in aggregate, we cannot yet tell in which concepts TIPP&SEE was more useful for these students. A larger suite of questions covering a wider variety of learning goals within each CT concept would be necessary to get a more definitive picture.

Across all our research questions, it was surprising that the Pair Cancellation subtest, which measures scanning and pattern recognition, was not associated with CS/CT performance and only weakly demonstrated a correlation with one Events question. This leads us to hypothesize that this cognitive skill is not related to the tasks in this curriculum. Perhaps, these skill is less necessary at the elementary level but would be more important for more advanced CS/CT concepts. It may also be the case that this skill does not apply for younger learners, the CT concepts were not complex enough to engage this skill, or something else entirely. Future research will be needed to further investigate this relationship.

WJ IV	**Total**				**Disability**			
	NR	VA	PC	VAL	NR	VA	PC	VAL
Very Superior	1	1	0	0	0	0	0	0
Superior	8	7	1	3	0	1	0	0
High Average	27	20	6	12	3	2	1	2
Average	96	107	107	110	11	10	12	16
Low Average	33	29	33	33	6	10	4	5
Very Low	7	5	7	0	3	5	2	0
Extreme Low	2	2	1	0	1	0	1	0
	ELLs				**Econ Disadvantage**			
	NR	VA	PC	VAL	NR	VA	PC	VAL
Very Superior	0	0	0	0	1	0	0	0
Superior	2	2	1	1	7	5	0	3
High Average	8	2	1	1	20	14	6	10
Average	33	40	46	41	83	93	92	92
Low Average	15	15	14	14	30	29	32	30
Very Low	3	4	1	0	7	5	4	0
Extreme Low	1	1	1	0	2	2	1	0

Table 6.2: Students in Each WJ IV Classification for all 4 cognitive subtests (Numbers Reversed (NR), Verbal Attention (VA), Pair Cancellation (PC), & Visual-Auditory Learning (VAL))

Concept	**Q**	**Numb Reversed**		**Verbal Attn**		**Visual-Auditory**	
		TS	C	TS	C	TS	C
Scratch	E&S Q2	.323**	—	—	—	—	—
Basics	E&S Q3	—	.270**	—	.277**	—	.431**
Events	E&S Q4a	.218*	—	.335**	—	.420**	.219*
	E&S Q4b	.237*	—	.391**	—	.416**	.235*
Sequence	E&S Q6	.263*	—	—	—	.222*	.223*
	E&S Q7	—	—	—	.235*	.294**	.361**
Loops	L Q1	—	—	—	—	—	.258*
	L Q2	—	.306**	—	.399**	—	.372**
	L Q4	—	.238*	.240*	.317**	—	.381**
Sequence & Loops	L Q5a	.442**	.321**	.410**	.258*	.347**	.358**
	L Q5b	.432**	.334**	.268*	.340**	.342**	.468**
	L Q5c	.285**	.285**	.276*	.331**	.365**	.360**

$^{*}p < .05; ^{**}p < .01$

Table 6.3: Correlations between Cognitive Skills & CT Performance on Questions from Events & Sequence (E&S) and Loops (L) Assessments

		Condition			Classification		
Numbers Reversed		$F(1, 163)$	p	η_p^2	$F(4, 163)$	p	η_p^2
Scratch	E&S: Q2	1.57	.213	—	4.13**	.0032	.0921
Basics	E&S: Q3	9.20**	.0028	.0534	6.67**	5.36×10^{-5}	.0141
Events	E&S: Q4a	2.14	.145	—	3.19	.0149	.0725
	E&S: Q4b	1.42	.235	—	1.11	.352	—
Sequence	E&S: Q6	1.46	.228	—	3.382**	.00541	.0856
	E&S: Q7	11.42**	.000911	.0655	3.28*	.0128	.0746
	L: Q1	15.62**	.000116	.0884	2.38	.0543	—
Loops	L: Q2	.556	.453	—	3.08*	.0177	.0717
	L: Q4	5.54*	.0198	.0333	2.68*	.0338	.0624
Sequence	L: Q5a	3.64	.0580	—	6.28**	.000101	.135
&	L: Q5b	28.39**	3.28×10^{-7}	.149	8.25*	4.42×10^{-6}	.170
Loops	L: Q5c	7.79**	.00590	.0461	5.54**	.000333	.121

*$p < .05$ **$p < .01$

Table 6.4: Test Statistics from Concept-Level Analysis of Events & Sequence (E&S) and Loops (L) Assessments for Numbers Reversed

		Condition			Classification		
Verbal Attention		$F(1, 162)$	p	η_p^2	$F(4, 162)$	p	η_p^2
Scratch	E&S: Q2	4.96*	.027	.0297	.658	.622	—
Basics	E&S: Q3	.909	.342	—	2.34	.058	—
Events	E&S: Q4a	.332	.566	—	1.54	.192	—
	E&S: Q4b	2.47	.118	—	2.24	.0670	—
Sequence	E&S: Q6	.320	.572	—	2.29	.0620	—
	E&S: Q7	.516	.473	—	2.18	.0737	—
	L: Q1	4.29*	.0398	.0261	2.48*	.0464	.0583
Loops	L: Q2	38.27**	4.94×10^{-9}	.193	5.93**	.000178	.129
	L: Q4	23.76**	2.61×10^{-6}	.129	3.51**	.00892	.0807
Sequence	L: Q5a	2.49	.117	—	4.38**	.00218	.0987
&	L: Q5b	11.83**	.000745	.0688	5.49**	.000361	.121
Loops	L: Q5c	2.23	.137	—	6.04**	.000149	.131

*$p < .05$ **$p < .01$

Table 6.5: Test Statistics from Concept-Level Analysis of Events & Sequence (E&S) and Loops (L) Assessments for Verbal Attention

		Condition			Classification		
Visual-Auditory		$F(1, 164)$	p	η_p^2	$F(4, 164)$	p	η_p^2
Scratch	E&S: Q2	20.09**	1.38×10^{-5}	.109	1.73	.146	—
Basics	E&S: Q3	.386	.535	—	2.69*	.033	.0617
Events	E&S: Q4a	.0398	.842	—	2.61*	.0375	.0598
	E&S: Q4b	.408	.524	—	2.69*	.0328	.0616
Sequence	E&S: Q6	2.81	.0951	—	2.32	.0589	—
	E&S: Q7	.552	.458	—	2.69*	.0327	.0617
	L: Q1	4.48*	.0358	.0269	1.46	.216	—
Loops	L: Q2	2.49	.116	—	2.36	.0552	—
	L: Q4	6.89**	.00950	.0408	4.21**	.00288	.0941
Sequence	L: Q5a	4.78*	.0301	.0286	3.10**	.0172	.0711
&	L: Q5b	3.64	.0582	—	5.31**	.000483	.115
Loops	L: Q5c	2.68	.104	—	5.19**	.000579	.114

*$p < .05$ **$p < .01$

Table 6.6: Test Statistics from Concept-Level Analysis of Events & Sequence (E&S) and Loops (L) Assessments for Visual-Auditory Learning

CHAPTER 7

UNDERSTANDING THE LINK BETWEEN COMPUTER SCIENCE INSTRUCTION AND READING & MATH PERFORMANCE

Many school districts all over the world have introduced computing curricula in their primary schools, expanding access to computing instruction. However, such expansion is not necessarily reaching all schools. In the United States, an early study of New York City's CS for All implementation found that schools with CS courses and activities served fewer Black and Latinx students and more White and Asian students, compared with schools without CS courses [65].

Schools can be so focused on core academic progress, particularly under-performing schools, that equitable educational opportunities for participation in content such as computer science/computational thinking (CS/CT) may be missed [41, 124]. The ramifications of these missed opportunities impact long-term outcomes for college entrance, employment, and quality of life. This amplifies the need for researchers and educators to provide equitable opportunities across schools. One way to increase the likelihood for inclusion of CS/CT curricula at the primary level would be to reliably demonstrate its impact on the core academic outcomes, such as reading and math, that are most valued by schools.

The purpose of this study was to explore the relationships between computer science learning and reading and math outcomes, with the overarching goal of examining whether exposure to critical thinking and problem-solving skills through computer science learning may generalize to other content. This study was guided by the following research questions:

- How is CS/CT instruction associated with reading and math performance on standardized tests?

- How do any associations between CS/CT instruction and reading and math performance apply to learners who face academic challenges?

In this study, I spearheaded the design of the study and assessment, as well as the statistical analysis. I participated in the development of student-facing materials and in student recruitment, which was led by Prof. Cathy Thomas from Texas State University. I also provided computing education support to the computer science undergraduate students assisting in classrooms.

7.1 Methods

7.1.1 Study Design

Teachers were randomly assigned to either the treatment or the comparison condition, resulting in five English-only and three bilingual English and Spanish classrooms in each condition. The eight teachers in the treatment condition were taught the TIPP&SEE learning strategy, which scaffolds student exploration of example programs for Use → Modify activities [208]. Classrooms in the comparison condition were taught Scratch Act 1 without the TIPP&SEE worksheets guiding them through the Use/Modify projects. There were a total of 92 and 101 students in the TIPP&SEE and comparison condition respectively.

An additional 162 students who did not receive CS instruction in any form were included as a de-identified control group. These students were randomly selected from the total district population of fourth graders such that the control group had a similar profile to the students who received CS instruction in terms of race/ethnicity, free and reduced lunch status (as a proxy for economic disadvantage), and special education status.

7.1.2 Reading & Math Scores

Statewide reading and math scores for 2018 and 2019 were provided by the district for each consenting student. Annually, to determine grade level readiness in reading and mathematics, Texas students complete the State of Texas Assessment of Academic Readiness (STAAR). Each STAAR question is aligned to the state curriculum standards, the Texas Essential Knowledge and Skills (TEKS), which teachers are mandated to follow as a guide to structure their lesson plans and teaching goals [230]. An external evaluation of the STAAR tests reported that development is consistent with best practices and that the Texas Education Agency (TEA) has provided evidence, including test blueprints and TEKS documentation, that support content validity [104]. This evaluation developed a predictive model to examine internal consistency reliability using Item Response Theory parameter estimates. Grade 4 projected statistics for reliability is high (0.913, Reading; 0.916, Math) and expected standard error of measurement is reasonable (2.71, Reading; 2.80, Math). At fourth-grade, the Texas Education Code [231] mandates that in mathematics, students should have the skills to use problem-solving models to support their planning, self-monitoring, and completion of work. Reading standards require development of comprehension of sequencing to carry out procedures. Standardized reading and math tests such as the STAAR assess skills that may be foundational to and associated with CS/CT.

7.1.3 Data Analysis

To evaluate the relationship between CS/CT instruction and reading and math performance, we first compared across conditions: students who received instruction with the TIPP&SEE learning strategy ("TIPP&SEE students"), instruction without TIPP&SEE ("Comparison students"), and no CS/CT instruction ("No CS students"). If there were no differences between the TIPP&SEE and Comparison students, we aggregated them into one group ("CS students") to compare students who did and did not receive CS/CT instruction in any

form. The different cohort configurations are summarized in Table 7.1. Since we tested two hypotheses on each dependent variable (reading or math score), we applied the Bonferroni correction so that our threshold for statistical significance is $p < .025$, half the standard threshold of $p < .05$.

Cohort	**Description**
Comparison	CS instruction without TIPP&SEE
TIPP&SEE	CS instruction with TIPP&SEE
CS	Comparison and TIPP&SEE cohorts
No CS	No CS instruction

Table 7.1: Different Cohort Configurations Studied

To study all participants as a whole, an ANCOVA was conducted on their reading and math scores to see if CS/CT instruction was associated with reading and math gains for all students. In our analysis, we controlled for 2018 scores as a covariate due to pre-test differences between the comparison and TIPP&SEE groups. We used Type III Sum of Squares for the imbalance between groups. We report F and p values from the ANCOVA. We also report the partial eta squared (η_p^2) effect size. The effect size indicates the magnitude of the observed effect or relationship between variables [143]. η_p^2 measures the proportion of the total variance in a dependent variable (DV) that is associated with the membership of different groups defined by an independent variable (IV) [43]. For example, if an IV has a η_p^2 of 0.25, that means that 25% of a DV's variance is associated with that IV. When comparing more than two groups, the Tukey post-hoc test was used to determine which groups were statistically-significantly different from one another, from which we report a p value.

Turning our attention to students groups who typically face academic challenges, we analyzed each group differently depending on the sample size. 86.5% of the students in our sample faced economic disadvantages, so they were analyzed the same way as the general

student body. Students with limited English proficiency and students with disabilities were analyzed using a non-parametric ANCOVA because of their smaller sample size (Table 7.2). When comparing more than two groups, we used the Dunn post-hoc test, a non-parametric post-hoc test. We report p values from both non-parametric tests. Unlike the parametric tests, we do not report effect sizes for non-parametric tests. The few existing non-parametric effect size estimates are not well-known or fully validated, and parametric effect size estimates are not appropriate to use on non-parametric data that violate assumptions of normality and heterogeneity of variances [130, 253]. The number of students in each group is shown in Table 7.2.

	TIPP&SEE	**Comp**	**No CS**
Overall	92	101	162
Economic Disadvantage	72	95	132
Limited English Proficiency	25	52	76
Special Ed/Disability	16	15	28

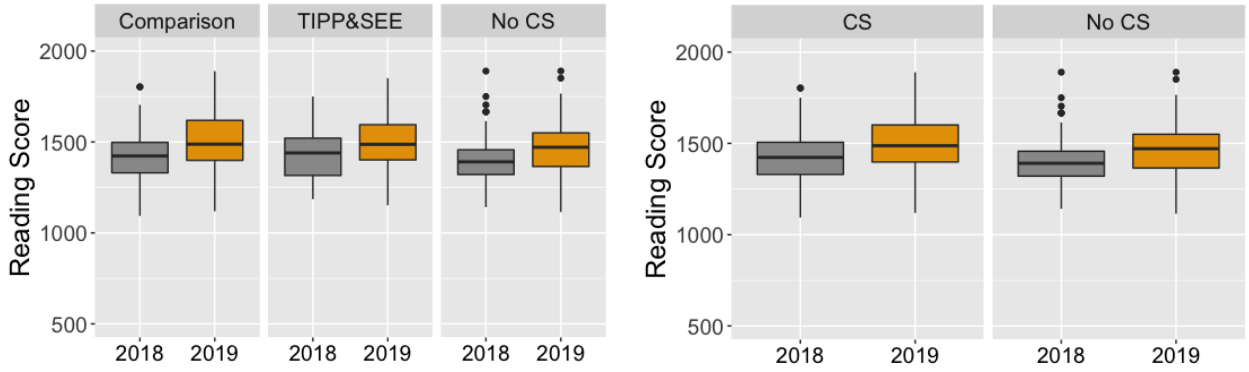
Table 7.2: Number of Students in Each Group

7.2 Results

To address our research questions, we first discuss results from analyzing all students in our study, followed by results from focusing on students who face academic challenges that impact reading and math performance.

7.2.1 Overall Student Body

CS/CT instruction was not associated with reading performance, both across conditions ($F(2, 331) = .384, p = .681$) and between CS/CT and control students ($F(1, 332) = .722, p = .396$). Figure 7.1a depicts the distribution of reading scores across conditions, while Figure



(a) Across Condition

(b) With & Without CS/CT Instruction

Figure 7.1: Reading Performance of Students

7.1b compares the reading scores of students who received any CS/CT instruction (combining both TIPP&SEE and Comparison groups) and students who did not. In both analyses, their 2019 reading scores were only associated with their 2018 reading scores, most likely a demonstration of academic growth across school years ($F(1, 331) = 478.94, p < .01, \eta_p^2 = .591; F(1, 332) = 480.40, p < .01, \eta_p^2 = .591$). This suggests that reading is not a skill that can necessarily be improved through programming.

Unlike reading, CS/CT instruction was related to math performance across conditions ($F(2, 332) = 11.08, p < .01, \eta_p^2 = .0625$). Just like reading, their 2019 math scores were also associated with their 2018 math scores, demonstrating growth across school years ($F(1, 332) = 437.71, p < .01, \eta_p^2 = .569$). A Tukey post-hoc test revealed statistically-significant differences between the Comparison groups and both the TIPP&SEE and no CS groups ($p < .01$). Figure 7.2 depicts the distribution of math scores across conditions. It is interesting to note that increased gains in math performance were only observed in the Comparison group, not the TIPP&SEE group. Potential reasons for this phenomenon are further explored in §8.1.3. The fact that the comparison group showed increased math gains suggests that there may be skills learned through CS/CT, such as critical thinking and problem-solving, that are generalizing to math in response to exposure to more open-ended CS/CT instruction.

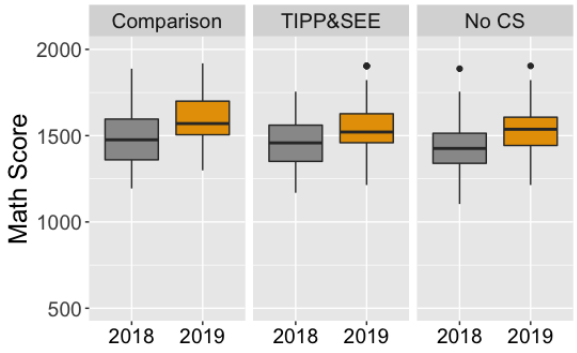


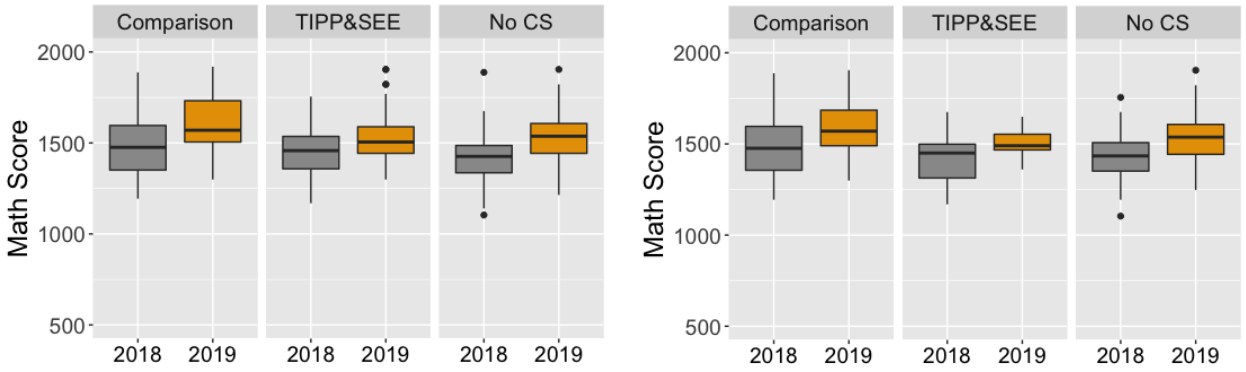
Figure 7.2: Math Performance of Students across Condition

7.2.2 Students facing Academic Challenges

We now turn our attention to students in situations that are correlated with academic challenges affecting reading and math performance: students with economic disadvantages, students with limited English proficiency, and students with disabilities [8, 83, 153].

For students facing economic challenges, we found no association between CS/CT instruction and reading performance gains, both comparing across conditions ($F(2, 278) = .103, p = .90$) and comparing those who did and did not receive CS/CT instruction in any form ($F(2, 279) = .044, p = .83$). In contrast, there was an association between CS/CT instruction and math performance gains when comparing the TIPP&SEE, Comparison, and the No CS groups ($F(2, 278) = 10.67, p < .01, \eta_p^2 = .071$). A Tukey post-hoc analysis demonstrated statistically-significant differences between the Comparison group and both the TIPP&SEE and No CS group ($p < .01$). Figure 7.3a depicts the distribution of the math scores of students with economic disadvantages in each condition.

Students with limited English proficiency also exhibited the same pattern. There was no association between reading performance gains and CS/CT instruction across conditions ($p = .73$) and across the presence or absence of CS/CT instruction ($p = .54$). CS/CT instruction was associated with math performance gains ($p < .01$), with a Dunn post-hoc test showing the Comparison students outperforming the TIPP&SEE group ($p < .025$). Figure 7.3b illustrates the spread of the math scores of students with limited English proficiency in



(a) Students with Economic Disadvantages (b) Students with Limited English Proficiency

Figure 7.3: Math Performance

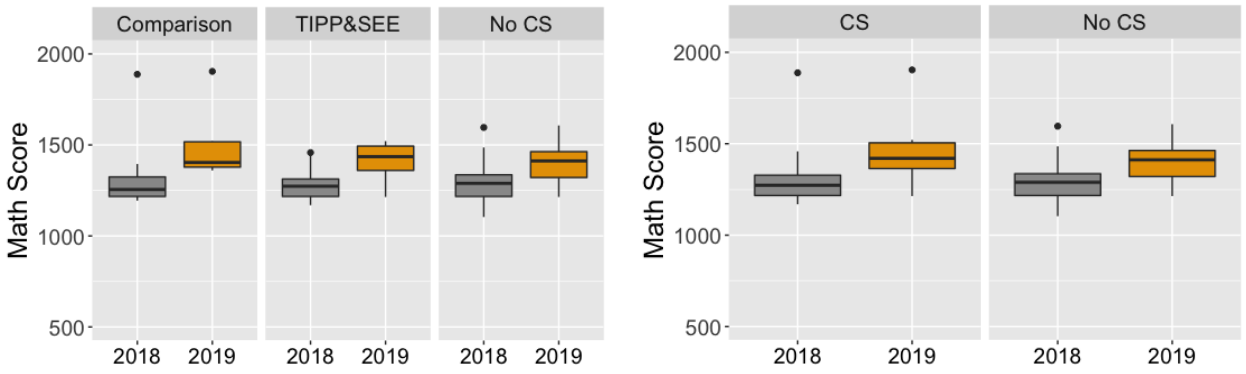
each condition.

Students with disabilities trended similarly with the overall student sample and the other students who face academic challenges in terms of reading performance: no association with CS/CT instruction across both conditions ($p = .75$) and the presence or absence of CS/CT instruction ($p = .45$). While other groups of students saw math performance gains with more open-ended CS/CT instruction, students with disabilities did not, in neither conditions ($p = .69$) nor presence/absence of CS/CT instruction ($p = .24$). Figures 7.4a and 7.4b present the math scores of students with disabilities across conditions and across the presence or absence of CS/CT instruction, respectively. It is important to note that the math scores of students with disabilities also trend lower than that of the other student categories. In 2019, they had a median score of 1412, compared with 1553 for the overall student sample, 1537 for students with economic disadvantage, and 1553 for students with limited English proficiency.

7.3 Discussion

We now return to our overarching research questions:

- How is CS/CT instruction associated with reading and math performance on stan-



(a) Across Condition (b) With & without CS/CT Instruction

Figure 7.4: Math Performance of Students with Disabilities

dardized tests?

- How do any associations between CS/CT instruction and reading and math performance apply to learners who face academic challenges?

For our first research question, this exploratory research provides preliminary evidence of the link between CS/CT and core academic of progress in reading and math. While students who struggle to read also struggle in CS/CT instruction [209], opportunities to participate in CS/CT instruction were not associated with reading improvements. Changes in reading may be associated with time, maturation, exposure to reading instruction, and potentially other factors.

However, in this study, students engaged in a less scaffolded computer science learning opportunity did demonstrate improved outcomes in math. This finding is encouraging given the importance of problem-solving and critical thinking skills for all learning. It is possible that this association is a result of exposure to and experience with the higher level math concepts embedded in the Scratch curriculum. It is also possible that the association is in response to opportunities in CS/CT curriculum to engage in higher level thinking skills with support. While more research is needed to replicate and extend this finding, generalizing of CS/CT skills to math is a very positive and desirable outcome.

It is important to note that improved outcomes in math did not result from TIPP&SEE, the more scaffolded learning opportunity. While TIPP&SEE led to better computing learning outcomes [209], it did not improve math performance. There are several possible explanations for this. First, it may be the case that the more scaffolded, less open-ended instruction resulted in less exposure to the skills that generalize to math. Second, some blocks in Scratch may expose students to more advanced math concepts, such as angles. When advanced concepts came up in Scratch, teachers in our study either explicitly taught those math concepts or worked around them [232]. It may be the case that a more structured curriculum dissuaded teachers from diverging from the curriculum to cover more advanced math concepts. Further research would be necessary to investigate this trend.

For our second research question, the association between math performance gains and more open-ended, less scaffolded CS/CT instruction applied to students with economic disadvantages and with limited English proficiency, but not to students with disabilities.

A majority of students in our study (86.5%) faced some form of economic disadvantage. The selection of such a student sample was intentional, as we wanted to expand access to CS/CT instruction to students who may not have been exposed otherwise. The fact that students with economic disadvantages made up such a large proportion of our sample is a potential reason why the overall student trend applied to them. A future study with a more representative sample would be necessary to identify broader trends.

While it would be reasonable to expect that limited English proficiency would be an additional barrier to CS/CT instruction in the US, students with limited English proficiency exhibited similar patterns as the larger student body. This is likely attributable to the sophisticated and well-established bilingual English/Spanish instruction in the schools in our study and the availability of bilingual materials [203].

The reading and math performance of students with disabilities were not associated with CS/CT instruction at all. The lack of improvement in the comparison condition is

predictable. Previous research in science learning has demonstrated that students with disabilities require scaffolding in order to access the curriculum [200]. Inquiry alone, as was available in the more open-ended comparison condition, is not sufficient for them. More specifically, students with learning disabilities do not succeed in open inquiry in either math or science and trends with their under-performance relative to even their lowest performing peers in reading and math [123, 150]. Additionally, students with disabilities tend to have slower growth rates in their reading and math scores [83, 214, 246]. It may also be the case that a one-year observation of reading and math scores was not enough time to notice any effect, that one semester of CS/CT instruction was insufficient, or that the forms of CS/CT instruction offered in our study have no link to reading and math for students with disabilities. Discerning the reason for this trend difference would require further investigation. This disparity for students with disabilities suggests that while we have preliminary evidence that more open-ended CS/CT instruction is a medium for learning cognitive skills generalizable to math, it is by far *not* a substitute for equitable math instruction and most definitely *not* a panacea for addressing inequities in math, as some have claimed [177]. Further research is necessary to better understand these relationships for students with disabilities, especially since we could not disaggregate the different types of disabilities due to privacy concerns.

CHAPTER 8

TYPES OF COMPREHENSION

In addition to understanding the factors that contribute to program comprehension and performance of young learners, it is also crucial to investigate the different types of comprehension achieved under the instructional practices in an introductory curriculum. Artifact creation, such as programs and scripts, are a fundamental tenet of CS instruction. Tinkering with someone else's code, perhaps from an instructor or from open source code, is a common step to developing such artifacts. However, little is known about how much and what type of comprehension is achieved through tinkering, especially at the K-8 level.

In this chapter, we describe two studies that address that research gap. The first study explores the relationship between characteristics of student code and their performance on code comprehension question [205]. The second study investigates the different responses students give, and consequently, the types of comprehension students demonstrate, when asked about their own code compared with generic code [206].

The first study took place in the 2017-2018 school year (approximately 9 months), while the second study took place over 2 school years 2017-2019. In 2017-2018, our study consisted of 316 4th grade students. In the 2018-2019, our study consisted of 329 3rd, 4th, and 5th grade students.

8.1 Exploring Relationship between Features of Student Code and Performance on Code Comprehension Questions

Many research studies of elementary students have used artifact analysis to draw conclusions about program comprehension [20, 75, 77, 157, 252]. However, artifact analysis can be misleading, stemming from both false negatives and false positives. Students can include code in their programs that they do not understand, leading to false positives [25]. Conversely,

students may understand concepts that they do not choose to include in their code, leading to false negatives.

This study seeks to better understand the relationship between student artifacts and student understanding. In particular, it answers the following research question: *What is the relationship between the presence of specific code constructs in a student's computational artifact and that student's performance on a question asking about those code constructs?*

In this study, we focus on structural understanding from the Block Model [215]. More specifically, we define understanding as being able to predict the outcome of a certain script run by the computer or if students could postulate which script produced a certain outcome, similar to Sorva et al [221]. This section presents key findings from this study, based on summative assessments given at the conclusion of two modules in a curriculum taught in a large urban school district in the United States.

For this study, I led the study design, assessment design and data analysis. It took place in an existing curriculum in San Francisco Unified School District (SFUSD), so I was not involved in curriculum design. I was involved in decisions regarding the participant recruitment strategy, but recruitment was led by two former SFUSD computer science leaders, Bryan Twarek and Bill Marsland.

8.1.1 Methods

In this section, we describe the context in which students created their artifacts, as well as the artifact analysis utilized in this study.

Artifact Creation Context

In addition to their summative assessments, we also collected students' culminating projects for each module. Students created a project over 1-3 class periods (approximately 1 hour each) based on an open-ended prompt meant to encourage the use of code constructs related

to the CT concept covered in the module. For example, to motivate students to use loops, they were prompted to build a band in Scratch, where sound clips would need to repeat in order to continuously play. 287 students submitted projects for Module 2, and 275 students submitted projects for Module 3.

In this study, the artifacts students created are projects that students develop from scratch, not remixed from the Scratch community. However, they were not prohibited from tinkering with code from the community. The knowledge they should possess is structural content related to control flow and individual block actions. More specifically, they should know what event causes a script to run, the order in which blocks run, and the result of those actions on the stage.

Artifact Analysis

Student projects were analyzed for attributes that were either indicative of overall complexity or were related to the concept tested in a question (Table 8.1). Overall complexity can be gleaned from total number of blocks, scripts, sprites, unique blocks, unique categories, and average script length (Scratch analog for lines of code in text-based programming). These attributes were analyzed for correlations with any of the assessment questions. Additionally, Module 2 taught sequence and events and Module 3 taught loops, so our analysis looked for both the total and the unique count of event and loop blocks. These attributes were analyzed for correlations only with their respective assessment questions.

Depending on whether the variables in question were dichotomous or continuous, either the point-biserial or the Pearson correlation coefficient was calculated. If either the attribute or the question score was dichotomous, the point-biserial correlation coefficient was used (shown in Q1, Q3-5, Q7). Otherwise, if both were continuous, the Pearson correlation coefficient was used.

Both methods result in a correlation coefficient r and a p-value. Absolute values of r

between .9 and 1 , between .7 and .9, between .5 and .7, between .3 and .5, and between 0 and .3 are considered very strong, strong, medium, weak, and very weak correlations, respectively [101]. A p-value less than .05 is statistically significant, meaning that we can reject the null hypothesis that the correlation is equal to 0.

Construct	Measures		
Block	total count	unique count	categories
Scripts	total count	avg length	
Sprites	total count		
Loops/Events	total count	unique count	

Table 8.1: Attributes from Artifact Analysis

8.1.2 Results

Our results aim to answer the core question — *does a student's use of certain code constructs mean that they understand the concepts associated with those code constructs?* For each question, we identify a specific code construct that would make sense as a proxy. We then present the data to give two sets of intuition. First, we present the statistical correlation between code attributes and assessment question results. Second, we present data to give a sense of the rates of the two circumstances we want to avoid - false positives (students with relevant code in their artifacts but do not understand the concept) and false negatives (students without relevant code in their artifacts but who do understand the concept).

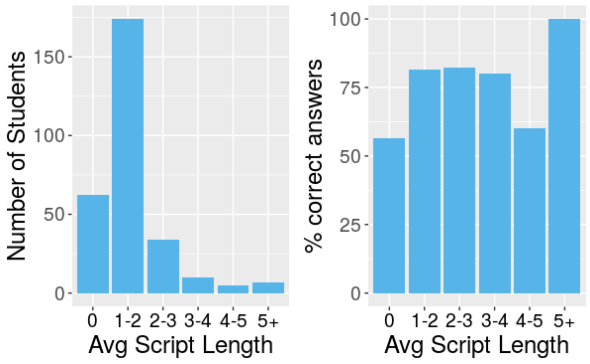
Q1: Sequence

Q1 asked students to circle the `say` block that ran last in a script with alternating `say` and `wait` blocks. A reasonable code construct for this question would be script length, assuming that students who implement scripts of sufficient length are more likely to understand

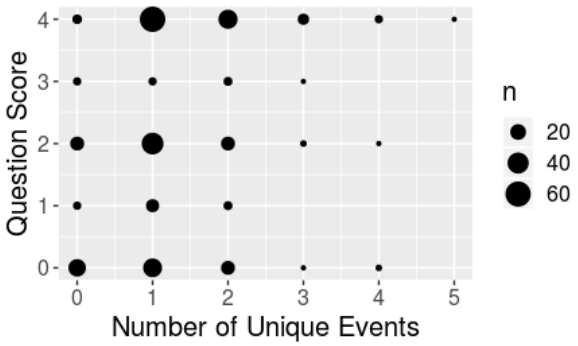
sequence. The distribution of the average script length across student projects is shown in Figure 8.1a. This shows that about 80% of students have a script length of less than 2, which means students predominantly have scripts without sequence—consisting of only an event block and a single action block.

Looking at Figure 8.1a, we see that students with script length 1-2 perform similarly to students with greater script length. In addition, there is no script length that can serve as a cut off – that prior to that length, students are incredibly unlikely to correctly identify the last instruction (leading to few false negatives) and afterwards are incredibly unlikely to incorrectly identify the last instruction (leading to few false positives). Not surprisingly, there was only a very weak correlation between question response and average script length ($r = .15, p < .01$).

In addition to the very weak correlation found with the average script length, there were also very weak correlations found with the number of categories from which blocks were used and the number of unique events. The rest of the attributes were found to have no correlation with performance on this question.



(a) Q1 Student Count (left) & Correct Responses (right) for Average Script Length



(b) Q2 Events for 1 Script vs Unique Events

Figure 8.1: Questions and Code Constructs for Sequence and Events

Q2: Events Starting 1 Script

Q2 showed four scripts and asked students to circle which script(s) would run if they click on the sprite. Two scripts started with `when sprite clicked`, one with `when green flag clicked`, and one with `when space key pressed`. Students received two points for every correct script circled and lost one for any incorrect script circled, for 0-4 points. Circling no or all four scripts earned 0 points, as neither gives any insight into understanding.

We explored two code constructs — the total number of events and the number of unique events a student used in their project. For the total number of events, it would be a sensible expectation that the more students implement and practice events, the better they understand their functionality. As for the number of unique events, it would be reasonable to expect that students who implement scripts with multiple events understand the relationship between the event the user performs and the resulting script that gets run.

There were very weak correlations between student performance on this question and the rest of the attributes of their projects, except for number of sprites, which had no correlation. Similarly, there were only very weak correlations between question score and (1) the number of events in code ($r = .21, p < .01$) and (2) the number of unique events in code ($r = .26, p < .01$). While there was a slight increasing trend in the average score depending on the number of unique events, there was a vast difference in individual performance on the assessment question for each number of unique events (Figure 8.1b).

Q3: Events Starting Parallel Scripts

Question 3 consists of two actions (playing drum and changing costume) in three scripts across two sprites (Pico & Giga), all started by `when green flag clicked`. Pico's single script performs the actions sequentially, whereas Giga's two scripts run in parallel (Figure 8.2a). To assess students' understanding of multiple events in multiple scripts versus sequential events in one script, students were asked to identify the true statements from the

following:

a) Pico plays the drum 7 times THEN changes costumes 4 times.

b) Giga plays the drum 7 times THEN changes costumes 4 times.

c) Pico plays the drum AND changes costumes at the same time.

d) Giga plays the drum AND changes costumes at the same time.

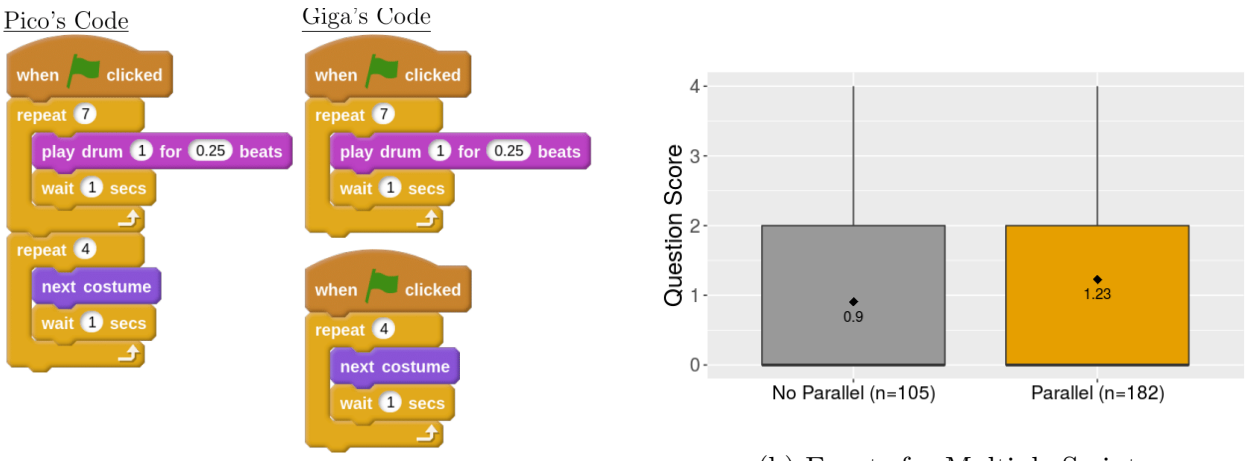
e) Pico and Giga both play the drum 7 times THEN change costumes 4 times.

The correct answers were (a) and (d). Students earned 2 points for each correct answer circled and lost 1 point for each incorrect answer circled, for 0-4 points. Circling no or all four scripts earned 0 points.

For this question, the artifact attribute used was using the same event for multiple scripts. There was no correlation between using the same event for multiple scripts and scores on this question (p=.076) (Figure 8.2b). In fact, students across the board struggled with this question, with an average score of 1.11. Performance on this question suggests a very high frequency of false positives for parallelism. Although students may use the same event for different sprites, they do not truly understand the resulting parallel execution. This result is not surprising, however, as even high school and university students struggle with parallelism [121, 134].

Q4: Repeat Iteration Count

Students were shown a repeat block and asked how many times the loop would repeat. For this question, the code attribute we focused on was the number of loops, assuming that students who have implemented and practiced more with loops were more likely to have a better understanding of repetition. The distribution of the number of students who used specific numbers of loops is shown in Figure 8.3a.

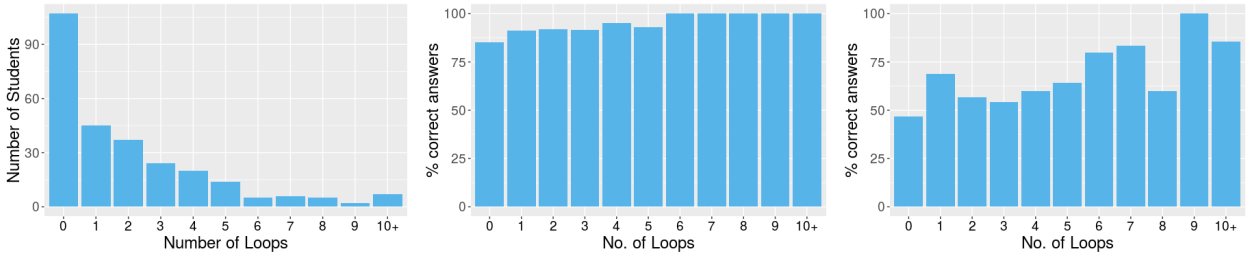


(a) Sequential (left) and Parallel (right) Scripts

(b) Events for Multiple Scripts

Figure 8.2: Q3 Question Materials and Results

Overall, students did very well on this problem, with 90.1% of students answering correctly. There was only a very weak correlation between answering this question correctly and the number of loops in their projects ($r = .12, p < .05$). There was no clear cut-off number of loops at which students who meet or exceed the cut-off are more likely to understand loops better compared with students below the cut-off (Figure 8.3b). Even students who did not use a single loop in their projects performed well on this problem, with 85% of them answering correctly. The other attributes were similarly very weakly correlated to performance on this question, except for the average script length and number of sprites, which had no correlation.



(a) Student Count for No. of Loops

(b) Q4 Correct Responses vs No. of Loops

(c) Q5 Correct Responses vs No. of Loops

Figure 8.3: Repetition Questions and Code Constructs

Q5: Unrolling a Loop

Students were shown a `repeat 4` loop consisting of two blocks. They were given choices of those two blocks repeated 1, 2, 3, and 4 times. Students were then asked to choose the unrolled code that had the same functionality as the loop. Similar to Q4, the code attribute chosen for this question was also the number of loops.

Compared with Q4, students struggled with this question, with only 57.3% answering correctly. Performance on this question was very weakly correlated with the number of loops used ($r = .17, p < .01$). The percentage of students who answer correctly fluctuates as the number of loops increases (Figure 8.3c). This suggests that there is no clear threshold number of loops above which students are more likely to understand loop functionality, compared with students who are under that threshold.

As for the other attributes, performance on this question was also very weakly correlated with the number of categories and scripts. There was no correlation with any of the other attributes.

Q6: Repeated Blocks vs Repeat Loops

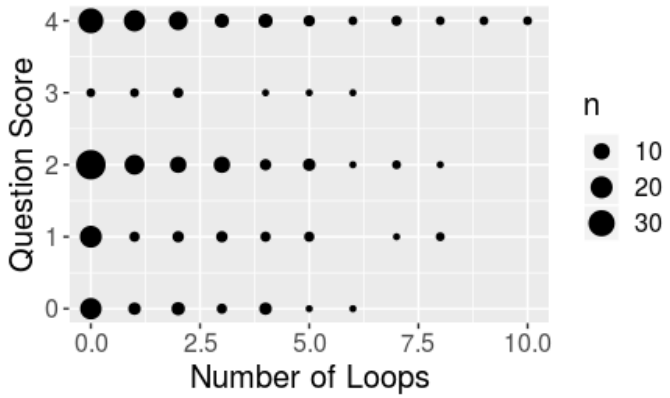
Students were asked to circle the scripts that would make a sprite perform some actions exactly three times. Students were provided one set of blocks (a) alone and (b) inside a `repeat 3` loop, and three sets of sequential blocks (c) alone and (d) within a repeat block (Figure 8.7). Q6 was designed based on a common misconception observed by teachers—not understanding the relationship between repeated code within a loop and repeated loop iterations. Choices were provided in random order on different assessments. Q6 had two correct answers (b and c described above); students received two points for each correct answer circled and lost one point for each incorrect answer circled, for 0-4 points.

As this question also asked about loop functionality like the previous two questions, the number of loops in their project was the code attribute of focus. There was a very weak

correlation between scores on this question and number of loops used ($r = .17, p < .01$). Student scores on this question varied regardless of the number of loops they used in their projects (Figure 8.4b).

As for the rest of the attributes, the total number of blocks, scripts, and unique loops were also very weakly correlated with scores on Q6. The others were not correlated with performance on this question.

(a) Answer Option (d) and inspiration for question.



(b) Repeat Blocks & Loops vs No. of Loops

Figure 8.4: Q6 Question Materials and Results

Q7: Loops Within Sequence

Question 7 consisted of a repeat loop sandwiched between two blocks and asked them three sub-questions: which blocks run (a) *in*, (b) *before*, and (c) *after* the loop. On each sub-question, students earned 2 points for each correct answer circled and lost 1 point for each incorrect answer circle, for 0-4 points (a) or 0-2 points (b, c).

For this question, the code construct of focus was whether or not they had a script that had at least one loop and one other action block. It would be reasonable to expect that students who used a loop within a sequence, whether the other block was before or after the

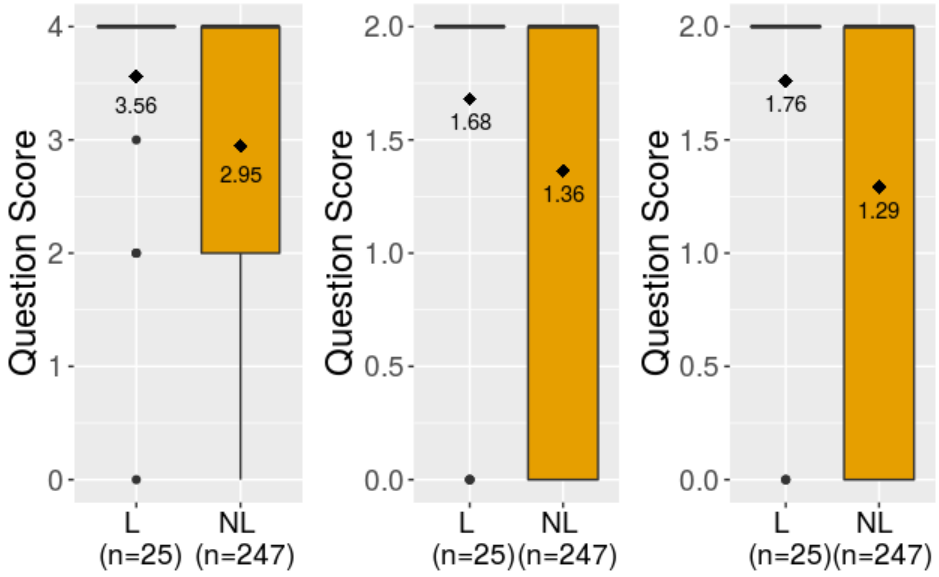


Figure 8.5: Q7a-c (left to right) vs Presence of Loop in Sequence (Loop/No Loop)

loop, would be more likely to perform well on this question. There were only 25 students who met this criteria.

There was only a very weak correlation between scores on this attribute and part (a) ($r = .12, p < .05$), no correlation with part (b) ($p = .1$), and a very weak correlation with part (c) ($r = .15, p < .05$). Students with the attribute code did well enough (Figure 8.5) that this code attribute could be considered a proxy for understanding. However, it is also clear from the figures that a majority of students lacking that particular code snippet also largely understand this concept.

Scores on part (a) were very weakly correlated with the number of categories, total number of scripts, total number of loops, and the number of unique loops. Scores on part (b) were very weakly correlated with the total number of blocks, categories, scripts, and the number of unique loops. Scores on part (c) were very weakly correlated to the rest of attributes, except for the average script length and the total number of sprites. There were no correlations between the other attributes and any of the question parts.

8.1.3 Discussion and Implications

We now revisit our original research question to see what this question-by-question analysis reveals. *What is the relationship between the presence of specific code constructs in a student's artifact and that student's performance on a question asking about those code constructs?*

	Blocks			Scripts		Sprites	Loops/Events	
Q	Total	Unique	Categories	Total	Length	Total	Total	Unique
Q1	-	-	VW	-	VW	-	-	VW
Q2	VW	VW	VW	VW	VW	-	VW	VW
Q3	**W**	VW	VW	VW	-	-	VW	VW
Q4	VW	VW	VW	VW	-	-	VW	VW
Q5	-	-	VW	VW	-	-	VW	-
Q6	VW	-	-	VW	-	-	VW	VW
Q7a	-	-	VW	VW	-	-	VW	VW
Q7b	VW	-	VW	VW	-	-	-	VW
Q7c	VW	VW	VW	VW	-	-	VW	VW

Table 8.2: Correlations between Question Score and Project Attributes.

In Table 8.2, dash ('-'), 'VW', & 'W' indicate no, very weak, and weak correlations, respectively. Correlation intervals are in Section 8.1.1. Concept-related code constructs had either very weak or no correlations with performance on the written assessments for every question (Table 8.2, except for Q3 which covered parallelism and had a more specific attribute of focus). In addition, there was only a single instance of an attribute having more than a very weak correlation – a weak correlation between the total number of blocks with identifying parallel vs serial code (in bold on Table 8.2). The *presence* of these correlations indicate that using the constructs meant that students were *more likely* to understand a concept – at the very minimum, a correlation would ideally indicate a structural understanding of the code constructs they used. However, the *magnitude* of these correlations fall far short

of demonstrating a proxy for understanding. This supports Brennan et. al.'s finding that students can use code in their projects that they do not truly understand [25].

This has important implications for computer science education research. As researchers, we need to be careful about the claims we make based solely on artifact analysis. Artifact analysis shows that a student *built* something - not that they *understood* something. A student does not use a loop in their code and immediately "understands loops" all at once. An understanding of loops is made up of individual learning goals, some dependent on each other (like a learning progression [13]) and some not (like a piece of knowledge approach [95]). We presented three assessment questions related to loops, all with very different performance by students. While a vast majority of students were able to tell how many times the loop would iterate, many fewer were able to identify equivalent sequential code or reason about the number of times something occurred within a loop and the number of times the loop iterated. Rich et al. presented a plethora of learning goals for this age group related to loops [196], some of which were tested in our assessment.

This study highlights the drawbacks of artifact analysis – an expedient but inaccurate choice – in order to spur work that will bridge this gap between written assessments, artifact analysis, and interviews. Avenues for future work include: (1) assessment techniques that balance the nuance and accuracy of interviews, the ease of written assessments, and the incorporation of student work fundamental to artifact analysis, (2) more careful applications of artifact analysis, and (3) support for teachers in designing projects that enable students to better demonstrate understanding.

As more elementary schools integrate computing into their curricula, with the equity goal contained in CS for All movements, they need tools that accurately assess the success of their curricula and teaching techniques. Such tools would enable schools to identify gaps and fill them with better curricula, refined software tools, teaching strategies, and learning strategies.

8.2 Response Patterns to Personalized and Generic Code Comprehension Questions

In elementary computing, three techniques are commonly used for assessing code comprehension: analyzing programs that students create (artifact analysis), giving written assessments, and interviewing students. Each has its major advantages and drawbacks. For example, artifact analysis has both false negatives and false positives. That is, students include code in their programs that they do not understand [25], leading to false positives. At the same time, students may understand concepts that they do not choose to include in their culminating project. On the other hand, interviews are the most accurate measure because of the ability to ask follow-up questions, but they are prohibitively time consuming.

In this study, we explore this code comprehension space with a novel approach—creating a personalized assessment that asks students questions involving their own code. With these customized assessments, this study seeks to answer the following question: *How does integrating a student's code from their artifacts affect the understanding they demonstrate on written assessment questions?*

For this study, I built the tool that developed personalized assessments for students, and directed the study design, assessment design and data analysis. This study occurred within a two-year-old curriculum in San Francisco Unified School District (SFUSD), so I was not involved in material design. I provided input into the participant recruitment strategy, but recruitment was led by two former SFUSD computer science leaders, Bryan Twarek and Bill Marsland.

8.2.1 Methods

In this section, we describe the *Personalized Assessment Worksheets for Scratch (PAWS)*, our tool for integrating student code into code comprehension questions, our data analysis

process.

Personalized Assessment Worksheets for Scratch (PAWS)

We developed PAWS, an assessment generator that searches Scratch projects for code snippets that are suitable for personalized questions, hereafter referred to as "candidate code". Candidate code is specified differently for each question. If there was candidate code in the student project, the generator randomly assigned the student either a personalized question using their candidate code or code from a generic question.

Data Analysis

Analysis was performed separately on each question, including *only* students with candidate code for that question. Due to absent students or blank responses, there was a slight ($< 10\%$) imbalance in the number of generic and personalized questions available for analysis. To account for the imbalance, the Type 3 Sum of Squares was used.

We first performed statistical tests to see if personalization had any influence on question scores; we compared students with candidate code who received (1) personalized questions and (2) generic questions using the ANOVA F-test. This test provides the F value in addition to the p value; in this study, we used $p < .05$ for statistical significance in all tests.

A subsequent hand analysis was performed on a subset of incorrect responses to detect patterns. This analysis came in several forms. With personalized questions, the answers were cross-referenced with student artifacts to find associations between incorrect responses and contents of student artifacts. For questions where student responses could be categorized without any overlap, the Chi-squared test was used to see if there was a dependency between the treatments (personalized vs generic) and the frequency of each response type. This test results in a Chi-squared value (χ^2) and like the ANOVA F-test, a p value.

Finally, open-response questions were also qualitatively coded for patterns in specific

attributes of their answers. First, a subset of free-response questions were open-coded to develop the qualitative coding scheme. The questions were then coded by at least two researchers with an inter-rater reliability (Fleiss' Kappa) of 80%. The proportion of personalized or generic responses with specific attributes were next compared using the Fisher's exact test, which is a non-parametric test for the equality of two proportions that also results in a p value. A non-parametric test was selected due to the small number of occurrences for some attributes.

8.2.2 Results

In this section, we present results divided by the different question types in which we integrated students' code. We first show results for a multiple-choice question in which students' individual code blocks are placed in a generic snippet of code. We then present multiple-choice questions involving full or truncated code snippets from students projects, which were not combined with generic code. Finally, we present results on open-ended questions on student scripts. We follow each set of results with a discussion of the insights gained and the kinds of understanding elicited from each form of code integration.

Blocks Integrated in Generic Code

This question (B1) asked students to circle the `say` block that ran last in a script with alternating `say` and `wait` blocks (Figure 8.6). Candidate code was any `say` block. Candidate `say` blocks replaced generic `say` blocks in the script.

87.04% of students who received the generic question circled the correct answer, compared with only 79.66% of students who received personalized questions. However, this difference was not statistically significant ($F(1,221)=1.83$, $p=.18$).

After inspecting the projects of students who did not answer the personalized question correctly, we found that some students circled the last `say` block in their *projects*, not the

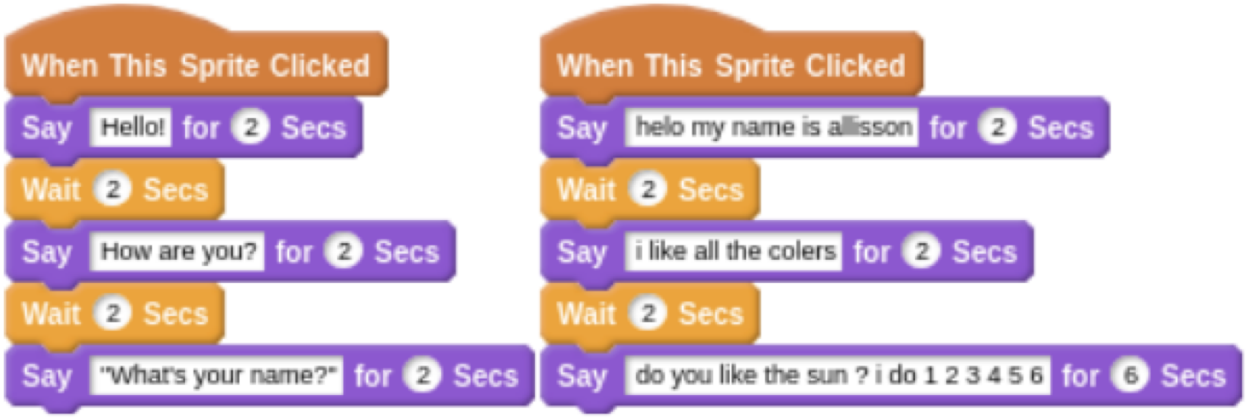


Figure 8.6: Generic (left) and Personalized (right) Scripts in B1

last `say` block in the *script* shown to them. Seeing their own code in the question may have caused students to think of characteristics of their own projects rather than the assessment code. Therefore, the use of student code in an assessment question should not differ too greatly from its use in their projects, lest it elicits a mismatch between a student's functional understanding of their own code and a structural understanding of the code snippet in the question.

Multiple-Choice/Fill-in-the-Blank with Code Snippets

We tested three questions using code snippets in multiple-choice questions: one on events and two on loops.

MF1: Multiple-Choice on Events

MF1 showed four scripts and asked students to circle which script(s) would run if they click on the sprite. In the generic assessment, there are two scripts with the `when sprite clicked` event block, one with `when green flag clicked`, and one with `when (space) key pressed`. Candidate code was any script from student projects; if their script had more than 3 blocks, only the first 3 were included.

In Year 1's personalized assessment, one of the generic `when sprite clicked` scripts was retained, and a candidate script could be swapped with any of the other three scripts.

However, this could lead to one, two, or three `when sprite clicked` events. To hold the number of each event type constant, in Year 2, candidate code only swapped out a script with same event block (though a different parameter was allowed for when key pressed).

Because there were multiple correct answers, we split student responses in several ways: (1) NO correct - students who circled none of the correct answers, (2) SOME correct & incorrect - students who circled some answers that were correct and some that were incorrect, (3) SOME correct & NO incorrect - students who only circled (some subset of the) correct answers and none of the incorrect ones, and (4) ALL correct & NO incorrect - students who circled all the correct answers and none of the incorrect ones - in other words, answered the question correctly. The distribution of student responses is displayed on Table 8.3.

Comparing the frequencies of each response type, we found a statistically-significant dependency between whether a student had a personalized or a generic question and how they responded on MF1 (Year 1: $\chi^2 = 19.59, p < .01$; Year 2: $\chi^2 = 27.34, p < .01$). As shown in Table 8.3, students who had a personalized question were more likely to circle both some (lower percentage of No Correct) or all (higher percentage of All Correct, No Incorrect) of the correct options than students who had a generic question.

From Year 1 to Year 2, there was also an improvement in the proportion of students circling all the correct answers, as opposed to only a subset. This may be attributed to the stricter candidate code selection criteria and/or more direct instruction of parallelism in Year 2.

MF2: Repeat Iteration Count

Students were shown a repeat block and asked how many times the loop would repeat. Candidate code was defined as a repeat block with fewer than 3 blocks inside it. If a student had a repeat block with more blocks, PAWS included only the first three blocks.

Overall, students performed well on this question, with at least 85% of students answering correctly in both years. There was no statistically-significant difference between control

Feature	Category			
	No Correct	Some Correct Some Incorrect	Some Correct No Incorrect	All Correct No Incorrect
Personalized (Y1)	15.6%	9.0%	20.7%	54.1%
Generic (Y1)	25.7%	13.3%	11.5%	49.6%
Personalized (Y2)	15.3%	9.0%	3.6%	72.1%
Generic (Y2)	23.7%	15.8%	0%	60.4%

Table 8.3: MF1 Qualitative Results

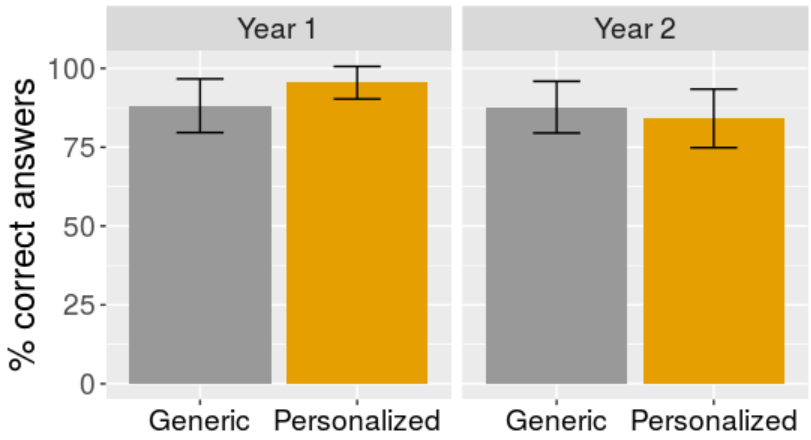


Figure 8.7: MF2 Repeat Iteration Count Results

and treatment (Year 1: F(1,123)=2.27, p=.13; Year 2: F(1,126)=.33, p=.56; see Figure 8.7).

MF3: Unrolling a Loop

In Year 1, students were shown two blocks inside a `repeat 4` loop and given choices of those blocks repeated 1, 2, 3, and 4 times. Students were asked to choose the unrolled code that did the same thing as the loop. Candidate code consisted of a repeat block with 2 blocks that was repeated at most 4 times. If a student had a repeat block with more than 2 blocks and/or the repeat block was repeated more than 4 times, PAWS included only the first 2

blocks and changed the number of times the repeat block ran to 4.

Overall, many more students struggled with MF3 than MF2; only 63.33% of students with a personalized question and 65.38% of students with a generic question answering correctly. These differences, however, were not statistically significant ($F(1,110)=.0018$, $p=.97$).

Hand inspection of assessments raised a potential issue with personalized code: certain personalized block combinations are visually similar (i.e. similar colors), which may have increased the difficulty of some personalized questions. This occurrence was too rare, however, to know whether it influenced the results.

Multiple-Choice/Fill-in-the-Blank Discussion

Unlike inserting individual *blocks* into a larger script, using complete *scripts* did not result in students bringing in project context that would impact their answers. However, further research is necessary to understand whether students are remembering how their code worked (functional understanding) or if it shows that they truly understand the mechanics of how the code works (structural understanding). In addition, care should be taken when defining candidate code, because students may be confused by idiosyncratic cases such as duplicate blocks.

Open-Response Questions

We tried three open-response questions. The first two asked students to explain code (one for sequence/events and one for loops), and the last asked them to describe when to use loops.

OR1: Explain a Sequence

In Year 2, OR1 was an open-response question in which students are provided a single script and asked to explain what that script does. OR1 was design to allow students to demonstrate (1) their ability to *articulate* an instruction's functionality and their understanding of the

order of instructions, and (2) their ability to identify the *event* that causes the action. A candidate code snippet was any script, excluding code constructs beyond the scope of their curriculum (e.g. forever loops, conditionals, and variables), limiting script length by allowing a maximum of 3 action blocks (the length of the generic script), and excluding blocks that had ambiguous sequential execution (e.g. the `play sound` block).

Our Year 2 revision transformed a fully open-ended question (Figure 8.8a) into more targeted questions that asked students to identify the event and sequence of the script (Figure 8.8b). The number of lines changed based on the number of action blocks in their script, as we allowed scripts with anywhere from 1-3 action blocks.

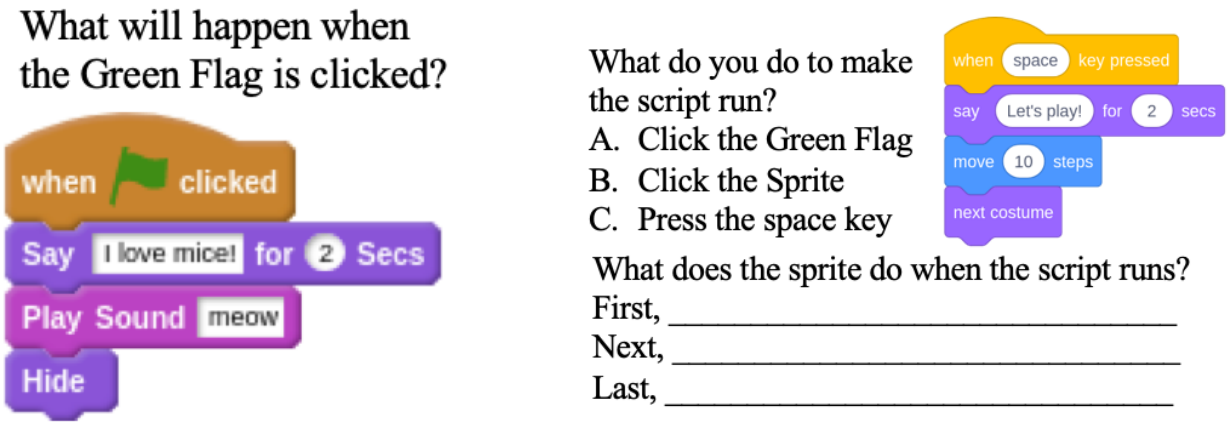
OR1 was worth 7 points—1 point was given for identifying the correct event that triggered the script, and 6 points were given for correctly describing the order and action of the blocks in the script. In order to receive credit for describing a block, the student needed to use the block name (e.g. `say`) and, if applicable, the major parameter (e.g. `say ‘‘hello"`). However, they were not expected to articulate minor details such as `for 3 seconds`. Students who received personalized and generic questions performed similarly in their score ($F(1, 192)=.17$, $p=.68$).

For this question, the score could obscure differences in student response patterns, so we performed a qualitative analysis of different details about the responses. This analysis revealed some patterns.

Students given personalized assessments were less thorough in their answer in two ways. First, 6.06% of them provided an answer with incomplete or missing parameters, compared with 3.52% of students with generic code ($p=.50$). Seeing their own code may have led them to provide answers that demonstrated their functional understanding (because they likely executed their own code), as opposed to their structural understanding. For example, when describing a script with several `move` blocks, a student wrote “it will go back and forth and stop”.

In addition, 5.05% of students with personalized assessments, in contrast with 2.11% of students with generic assessments, had an answer where they incorrectly described at least one block's functionality (p=0.45). The generic code snippet used only blocks students have seen before, but personalized code may have had blocks not covered by the second module (events & sequence). Although students may have used such blocks in their projects, they may not fully understand them.

Nonetheless, a greater percentage of students with personalized assessments *answered* the question (90.91% personalized vs 87.37% generic; p=.37). Students with the generic question may have been less familiar with or engaged by the generic code, causing them to just copy the numbers from the blocks, write nonsensical responses, or leave the question blank.



(a) Year 1 Fully Open-Ended

(b) Year 2 Scaffolded

Figure 8.8: Open-Response Question 1: Explain a Sequence

OR2: Explain a Loop

For OR2, students were shown a loop and asked to explain what the loop would do in their own words. In Year 1, candidate code was defined as either a countable or forever loop. Answers were given between 0-10 points depending on accuracy and completeness. As in OR1, the Year 2 question was revised to provide more scaffolding for student responses.

Instead of a blank empty box, students were given lines preceded by "First", "Next", and "Last", similar to OR1 (Figure 8.8b), and there was a separate blank for the number of times the loop would repeat. Year 2's OR2 was graded in the same way as Year 2's OR1, with an additional point for identifying the number of loop iterations. In addition, forever loops were excluded as they were not explicitly covered in the curriculum; thus, we could not expect students to understand them.

Most students performed well on this question, with an average score of 8.37 out of 10 possible points in Y1 and 5.97 out of 7 points in Y2. As shown in Figure 8.9a, there was no statistically-significant difference in performance between students who received a generic question and students who received a personalized question (Y1: $F(1,156)=3.30$, $p=.071$, Y2: $F(1,75)=.69$, $p=.50$).

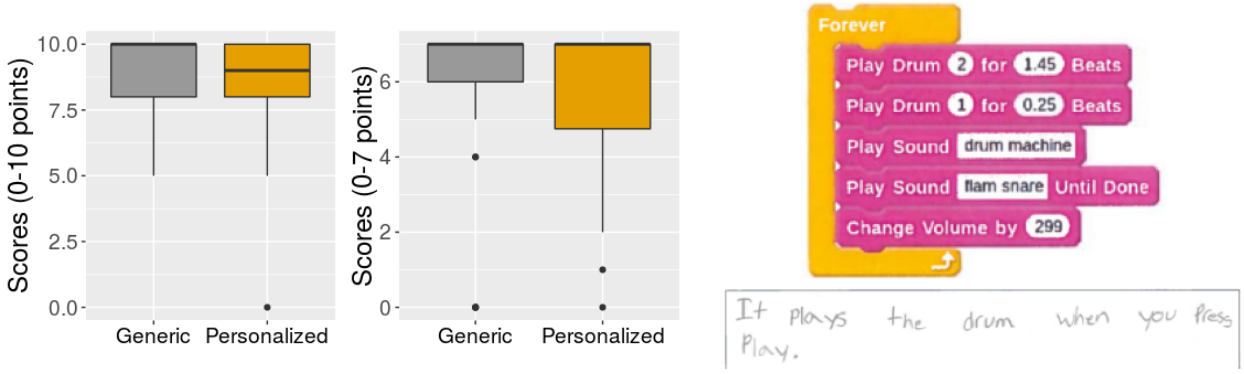
While there was no statistically-significant difference in performance between the two treatments, qualitative analysis of Year 2 responses again revealed distinct patterns in student responses to personalized and generic questions.

Students who received personalized questions were more likely to write responses that were not precise enough to assess their structural understanding (how the code works). A greater proportion of them left out parameters (8.33% personalized vs 7.69% generic). They were also more leave out block descriptions entirely (16.67% personalized vs 0% generic; $p < .05$). As in OR1, some students with personalized questions brought in context from their projects in their responses, demonstrating functional instead of structural understanding (Figure 8.9b).

Similar to Year 2's OR1, a higher percentage of students who received personalized code in Year 2 *answered* the question (97.22% personalized vs. 89.74% generic; $p=.36$). This could be attributed to students being more familiar or engaged by seeing their own code.

Nonetheless, there was one trend from OR2 that differed from OR1. In OR1, a greater proportion of students with *personalized* code described a block incorrectly. In contrast,

a greater proportion of students with *generic* code described a block incorrectly in OR2 (12.82% vs. 2.78%; p=.20). Closer inspection of responses revealed that for this question, more of the personalized code included blocks that were taught in the curriculum, whether in the events & sequence module or in the loops module. By this point in the curriculum, students have been exposed to more blocks. Students with personalized code actually used the blocks they were being asked about. At a minimum, these students could be expected to have a functional understanding of the loop shown in their question and thus, would be less likely to provide a description that was completely wrong. On the other hand, students may not have actually used the blocks in the generic script, even though they were covered in the curriculum.



(a) Free-Response on Loops Results

(b) Personalized Script described with Project Context

Figure 8.9: Open-Response Question 2: Explain a Loop

OR3: Reasons for Using a Loop

In Year 2, OR3 was added to simulate an interview question and thus, was not graded for correctness like the previous open response questions. Candidate code was any countable loop from their projects, which was truncated if it had more than 3 blocks. The generic version of OR3 asked students to explain when they should use a loop, while the personalized question showed them a loop from their project and asked them why they chose to use that loop and how they decided the number of iterations.

Students who received generic questions were nearly twice as likely to write the correct, general purpose use of a loop (63% vs 36%; $p < .05$). This included responses like: "when I want to do something more than once", "when I want to repeat something" and "when you put the same blocks over and over".

In contrast, a higher percentage of students who received personalized questions cited a specific use of a loop (17% vs 9%; p=.34). Responses that were too specific included: "to start movements", "when you play a sound" and "to change costume". Since students are shown a snippet of their own code in the personalized question, they may interpret the question as asking them to identify the use of a loop in that specific instance. This led students to explain the loop in terms of its use in the sample code rather than its general use.

A higher proportion of students with personalized questions also attributed the use of a loop to making their code shorter or to save time (19% vs 7%; p=.12). Since the students see their own code in the personalized question, they may have been prompted to think of the reason why they used the specific loop shown in the question. In this case, students cited benefits of using a loop, rather than its general purpose, which is to repeat code.

These situations where students do not provide the appropriate level of specificity or describe benefits of using a code construct without explaining their underlying reasons would be mitigated in an interview setting because an interviewer would be able to ask follow-up questions.

Open-Response Discussion

Overall, we found that when students explain code snippets, they are less precise when given their own code but are more likely to answer with correct statements. Students who received personalized code may remember their intentions in coding that script and the project in general, allowing them to better understand what the sprite would do when the script was run. However, this familiarity may put them more in a functional frame of mind, causing

them to skim over the details in their answers. In addition, thinking of their specific project may have thwarted the question which asks when, in general, students should use loops in projects.

It is not clear the reason for the difference in accuracy in responses. On one hand, students given code from their projects may be more likely to understand that code because they have used it. On the other hand, they may have used blocks in their project that they don't fully understand. A counter argument is that a block in generic code might have only been seen by the student in a lesson but never used in their own program, so they may not be as familiar with it. Interviews or think-alouds would be required to better understand if these trade-offs favor one side or the other.

8.2.3 Implications

We now revisit our original driving question to see what this analysis reveals: *How does integrating a student's code from their artifacts affect the understanding they demonstrate on written assessment questions?*

We identified the following patterns:

- When blocks are taken out of context from their project, they may answer based on how the block is used in their project rather than in the script on the assessment.
- When asked multiple-choice questions about their scripts or partial scripts in which the original meaning is retained, they answer similarly to or better than students receiving generic questions.
- When explaining their code, they are more likely to answer the question, but they often do not describe the individual blocks as thoroughly as students receiving generic questions.

When students with personalized and generic questions answered differently, they demonstrated a functional understanding of the code in the question, instead of the intended structural understanding. As they built the code snippets they were being asked about, they were likely to remember their goals while creating their projects; interviews about student artifacts also face a similar challenge [25]. Further research is merited to explore how to ask questions that are not overly complicated but cause students to demonstrate structural understanding of their code.

In this study, we investigated the space between the three common assessment techniques (artifact analysis, written assessments, and interviews) in elementary computing with a novel approach – integrating student code into written assessments through our tool *Personalized Assessment Worksheets for Scratch (PAWS)*. Our results revealed patterns in student responses and mismatches between the types of understanding demonstrated; future work will explore ways to address these mismatches.

CHAPTER 9

TIPP&SEE: A STRATEGY FOR USE → MODIFY ACTIVITIES

To overcome the disparities faced by diverse students delineated in Chapter **??** and the gaps in understanding described in Chapter 8, we developed TIPP&SEE (Figure 9.1), a learning strategy that scaffolds student exploration of provided programs for Use→Modify activities. The strategy is specifically designed for use with Scratch, a popular programming language and development environment used in elementary schools [71].

Similar to prior work [166, 254], we draw upon strategies from reading comprehension in designing TIPP&SEE as learning to program relies heavily on reading comprehension at several stages in the learning process. Students need to learn how to read (a) individual instructions, (b) a sequence of instructions provided as an example or starting code, (c) one's own partially-completed code, or (d) one's completed but incorrect code. Just as in reading, it is not enough to decode the letters into words; to succeed, the student needs to make meaning of the sequences of words into instructions (like sentences) and the sequences of instructions into functions or programs (like paragraphs). We draw from existing evidence-based reading comprehension strategies that may have connections to programming, in particular previewing and text structure.

Previewing [118, 145] guides students as they set goals for reading and activate prior knowledge. When reading example code containing a new concept, students might scan the code to quickly identify familiar and unfamiliar concepts. They could think about their prior knowledge of the concepts, predict how the new concept might work, and inspect the syntax of the new concept.

Text structure [81, 249] helps students recognize disciplinary-specific text structures and use this knowledge to plan for reading and guide comprehension. In CS, programming languages and environments have specific structures that students need to recognize in order to comprehend code as they learn new languages and environments.

Inspired by previewing strategies, the first half, *TIPP*, guides students in previewing different aspects of a new Scratch project before looking at any code. As a last step, they run the code with very deliberate observations of the events and actions that occur. The second half, *SEE*, draws from text structure strategies. *SEE* provides a roadmap for finding code in the Scratch interface (clicking on the sprite and finding the event) and proceduralizes the process by which they can learn how code works by methodical exploration (deliberate tinkering).

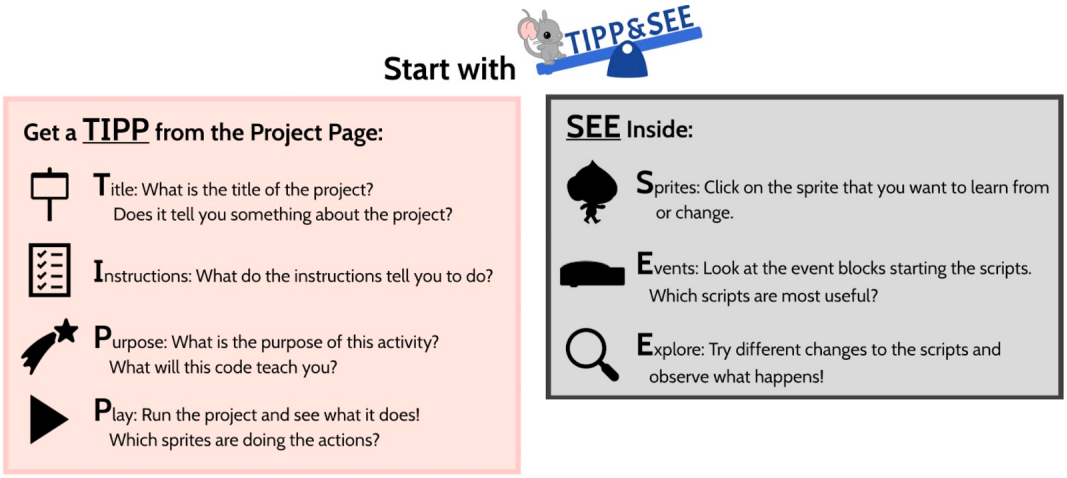


Figure 9.1: TIPP&SEE Learning Strategy

In this chapter, we outline three studies on the effectiveness of the TIPP&SEE learning strategy. The first showed TIPP&SEE to be associated with positive performance differences in summative CT assessments [208], the second showed that using TIPP&SEE was linked to narrowed gaps between students with and without academic challenges [207], and the third explored potential reasons why such performance differences occur [76].

9.1 Supporting the Learning of Introductory CT Concepts with TIPP&SEE

The Use → Modify → Create [127] pedagogical approach has been proposed to provide additional scaffolding to young learners, adding a Use→Modify task prior to an open-ended activity. In a Use → Modify task, students learn by example code. They are provided with something that works to illustrate how to code a particular construct then are first asked to perform some small modification before tackling a more open-ended problem in which they apply that knowledge to their own project. In this study, we investigate the following overarching question: *Does TIPP&SEE improve student performance in introductory CT concepts—events, sequence, & loops?*

In this study, I helped with the design of the TIPP&SEE strategy and the development of curricular materials. I also supported the computer science undergraduate students who helped the teachers in Austin Independent School District (AISD). I led the assessment design and data analysis in all studies. While I was involved in decisions on participant recruitment, recruitment was led by our collaborator Prof Cathy Thomas, a professor of education at Texas State University.

9.1.1 Methods

Study Design

Fifteen teachers were recruited from a large, urban school district and underwent the same professional development to teach the Scratch Act 1 curriculum to 4th grade students (ages 9-10). A total of 16 classrooms participated in the study, including six of bilingual classrooms. Teachers were randomly assigned to either the TIPP&SEE or the control condition, resulting in five English-only and three bilingual classrooms in each condition. Treatment classrooms used TIPP&SEE worksheets, whereas control classrooms used worksheets that introduced

the overall project and modify task without stepping students through the protocol. Lessons were taught by the teacher and assisted by an undergraduate CS student. After excluding students who switched schools or were chronically absent, there were a total of 96 and 88 students in the control and TIPP&SEE condition, respectively, for a total of 184 students.

Data Analysis

Scratch Act 1 end-of-module summative assessments were graded by two researchers to ensure reliability. To see if TIPP&SEE had an influence on their assessment performance, the ANOVA F-test was used. The ANOVA F-test returns a p-value; for this study, $p < .05$ is statistically significant. To handle the imbalance between groups, Type 3 Sum of Squares was used. Features of free-response questions were qualitatively coded by two researchers with a Fleiss' Kappa inter-rater reliability of at least 80%.

The eta squared (η^2) effect size was also calculated. η^2 measures the proportion of the total variance in a dependent variable (DV) that is associated with the membership of different groups defined by an independent variable (IV) [43]. For example, if an IV has a η^2 of 0.25, that means that 25% of a DV's variance is associated with that IV.

9.1.2 Results

Based on the test blueprint (Table 4.3 in Chapter 4), we discuss questions based on the CT concept they covered and their corresponding Bloom's taxonomy level. In addition, we also show the two most advanced questions as they illustrate the limits of TIPP&SEE. A summary of question results are shown in Table 9.1; we do not report on all questions as not all of them significantly loaded on latent variables as revealed in the exploratory factor analysis.

Scratch Basics

Q2 and Q3 from the Events & Sequence assessment covered a basic recall of Scratch. Q2 asked students to identify the last block in a sequence. Over 70% of students in both conditions answered Q2 correctly with no statistically-significant difference in performance, suggesting that the curriculum alone sufficiently supported students in learning a basic concept without any additional scaffolding.

Q3 asked students to identify a multi-block script triggered by the `when sprite clicked` event. TIPP&SEE students outperformed the control students on Q3, with 69.7% of them answering correctly, compared with 50.5% of the control students (see Figure 9.2). Closer inspection of the responses revealed that the control students were more likely to select the options that started with `when green flag clicked`, the event that students were most familiar with (75% of the control students vs 40.66% of the TIPP&SEE students).

Events

Q4 from the Events & Sequence assessment covered an understanding of events. It showed students a stage with two sprites saying different things after the green flag was clicked and asked which script ran for each sprite in two parts. Around 70% of TIPP&SEE students answered both parts correctly, compared with around 58% of the control students. However, this difference was not statistically significant.

Sequence

Q6 and Q7 from the Events & Sequence assessment and the three different parts of Q5 from the Loops assessment covered sequence at the Bloom's Taxonomy level of Understanding. Q5 covered both an understanding of sequence and loops, but since loops was its main concept, we will discuss in the following section.

Q6 and Q7 from the Events & Sequence assessment both asked students to explain a

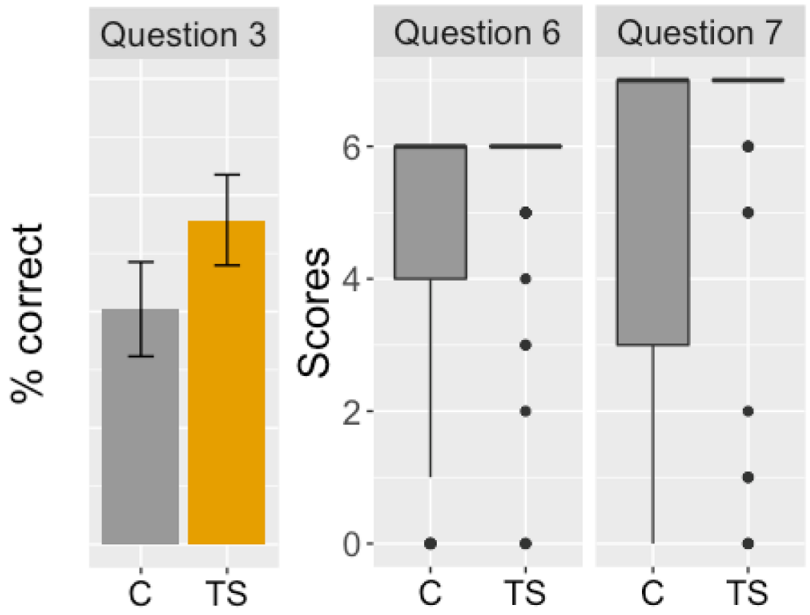


Figure 9.2: Q3: Scratch Basics and Q6 & Q7: Sequence x Understanding

script of 3 blocks in their own words. Both questions had blank lines preceded by "First", "Next", and "Last" to scaffold their answers; each line was worth 2 points. Q6 was worth 6 points, while Q7 was worth 7 points because it had an additional question asking about the event worth 1 point. As shown in Figure 9.2, there were statistically-significant differences in performance on both Q6 & Q7 between the TIPP&SEE and control students, suggesting that the additional scaffolding provided by TIPP&SEE encouraged a deeper understanding of events & sequence.

Qualitative analysis further illuminated some patterns. TIPP&SEE students were less prone than the control students to respond with an incorrect sequence or missing blocks (Q6: 10.11% vs 18.75%; Q7: 11.24% vs 31.25%). They also provided more precise responses & were less likely to leave out the block name or, if applicable, an important parameter when describing blocks (Q6: 8.98% vs 27.08%; Q7: 10.12% vs 16.66%).

Overall, the data from these free-response questions show that students in the TIPP&SEE condition could demonstrate a more sophisticated understanding of the blocks themselves, as well as the CT concepts of sequence, than students in the control condition.

Loops

Q1, Q2, Q4, and Q5 from the Loops assessment covered an understanding of loops. Q1 showed students a loop and asked them how many times it would repeat. While students in both conditions performed well with 94.4% of TIPP&SEE & 84.5% of control students answering correctly, there was still a statistically-significant difference (Figure 9.3).

In Q2, students were shown 4 code snippets and asked which snippet would cause the sprite to change costumes 3 times. One code snippet was inspired by a common misconception observed by teachers where students would wrap repeated blocks with a repeat loop that had the same number of iterations as the number of blocks. There were 2 correct answers; students received 2 points for each correct answer and lost 1 point for each wrong answer for a maximum of 4 points. TIPP&SEE students outperformed control students with mean of 3.2 points compared with 2.4 points (Figure 9.3). Students in the control condition were more than twice as likely as the students in the TIPP&SEE condition to choose the common misconception option (43.75% vs 18.39%), supporting the observations made by the teachers.

Q4 asked students to unroll a `repeat 3` loop with 2 blocks. Among its multiple choices, Q4 had a "split loop" option – where the first block was repeated 3 times followed by the second block repeated 3 times. 82.8% of TIPP&SEE students answered Q4 correctly, in comparison to only 48.9% of the control students (see Figure 9.3). An analysis of common mistakes revealed that the TIPP&SEE students who answered incorrectly tended to choose responses that suggested a closer, but flawed, understanding of loops – 14.94% of TIPP&SEE students chose the "split loop" option, compared with 12.5% of the control students. In contrast, 31.25% of the control students selected the option where the blocks were repeated only once, compared with only 2.29% of the TIPP&SEE students.

Q5 from the Loops assessment showed a script with a loop and asked 3 sub-questions: (a) code *in*, (b) *before*, and (c) *after* the loop. Part (a) was worth 4 points—students earned 2 points for each correct block circled and lost 1 point for each incorrect block circled; parts

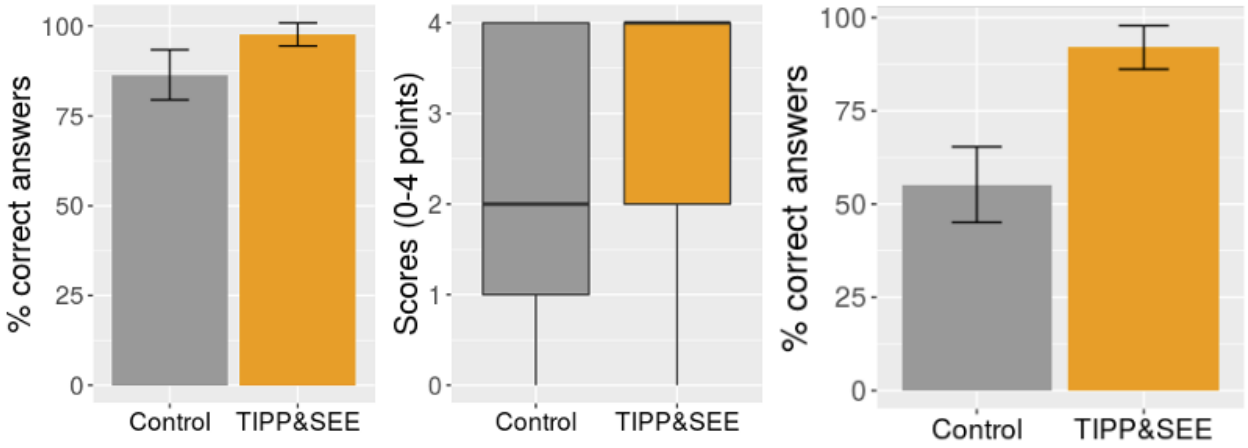


Figure 9.3: Q1, Q2, and Q4 Results (left to right): Loops x Understanding

(b) and (c) were worth 1 point.

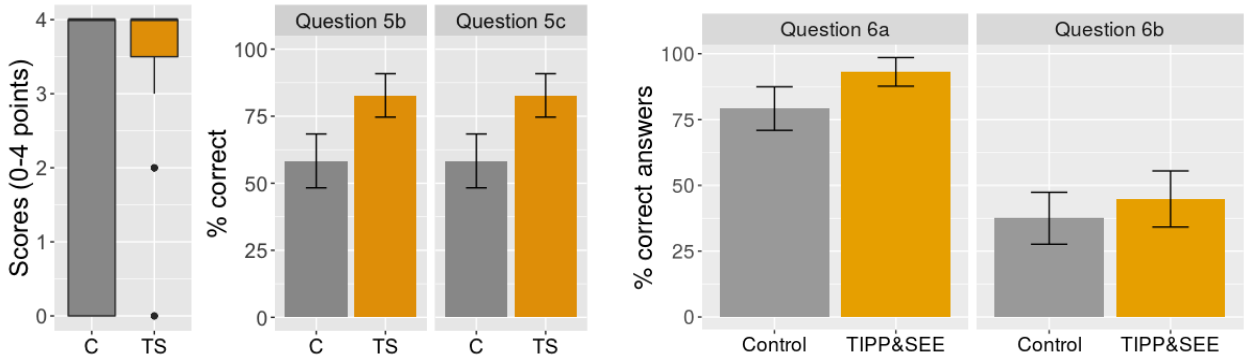
TIPP&SEE students outperformed the control students on all three parts, as shown in Figure 9.4a. On part (a), TIPP&SEE students scored a mean of 3.29 points, while the control students scored a mean of 2.51 points. On parts (b) and (c), 82.8% of the TIPP&SEE students answered correctly, compared with only 58.3% of the control students.

Taking these questions in perspective, we find that the control students displayed a more superficial understanding of loop functionality compared with the TIPP&SEE students. While many control students were able to answer the simplest question on repeat iteration count, they struggled with the more advanced questions on loop unrolling and loops in sequence.

Advanced Questions

We also highlight the two most advanced questions in our assessments: Q6 and an Extra Credit (EC) question from the Loops assessment.

Q6 asked students about the execution of two sprites' code: part (a) asked about a sprite with sequential loops, while part (b) asked about a sprite with two loops in parallel. For part (a), 93.1% of the TIPP&SEE students were able to correctly identify the sequential



(a) Sequence & Loops x Understanding

(b) Parallel Loops

Figure 9.4: Q5 (left) and Q6 (right) Results

behavior, as opposed to 79.2% of the control students. In contrast, students in both conditions struggled with part (b) with only 44.8% of TIPP&SEE and 37.5% control students answering correctly, suggesting the difficulty of parallel execution for this age group. Results for both parts are shown on Figure 9.4b.

The Extra Credit (EC) question asked about nested loops, a topic not explicitly covered in the curriculum. Unsurprisingly, students in both conditions similarly struggled with this question. The TIPP&SEE students performing slightly better with 22.9% of them answering correctly, compared with 12.5% of the control students. This difference, however, was not statistically significant.

9.1.3 Discussion

We now revisit our key research question: *How does TIPP&SEE influence student learning of introductory CT concepts—events, sequence, & loops?*

Our findings show that students using TIPP&SEE outperformed students who used an unmodified Use → Modify → Create approach on a Scratch Basics question and all Sequence questions (Figure 9.5; asterisks denote significance). Most students were able to demonstrate a simple understanding of events with just the scaffolding provided by Use → Modify →

Create, but with TIPP&SEE, they could also demonstrate an understanding of sequence.

On the loops assessment, the TIPP&SEE students performed better than the control students in almost all questions; only parallelism and nested loops (which was not explicitly covered in the curriculum) were beyond their grasp (Figure 9.6). This suggests that while students are able to make significant learning gains with TIPP&SEE, there is still room for improvement in the instruction of parallelism.

As momentum continues to build for integrating CS/CT into elementary school classrooms, it is imperative that CS/CT instruction be effective for diverse learners. A learning strategy like TIPP&SEE provides some much-needed scaffolding for such diverse learners, advancing not just equitable access, but also equitable outcomes in elementary computing.

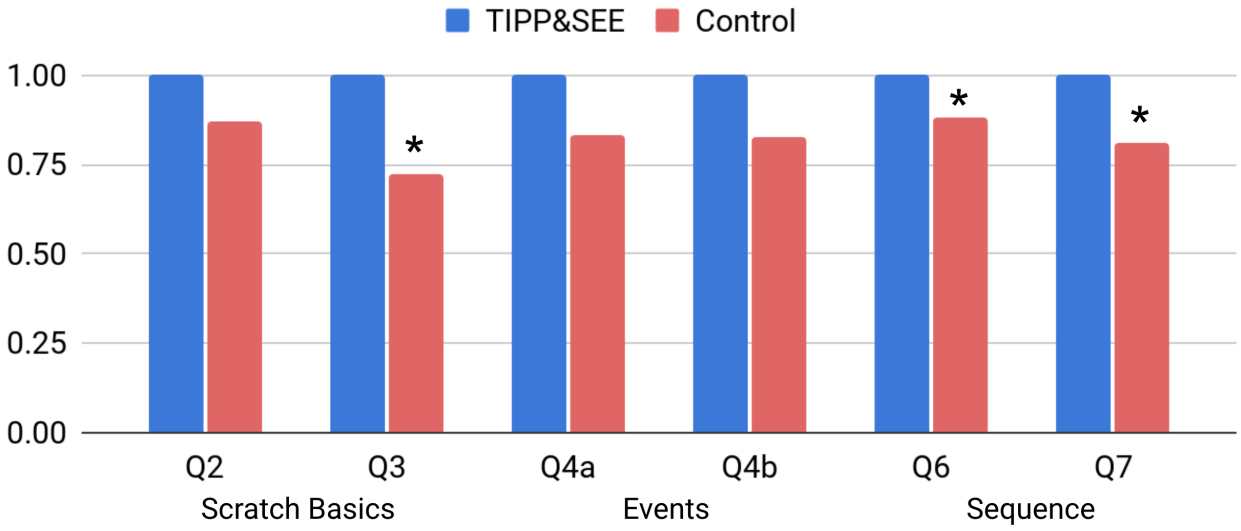


Figure 9.5: Results for Scratch Basics, Events, & Sequence

9.2 Supporting Diverse Learners with TIPP&SEE

The study in the previous section showed that students using the TIPP&SEE learning strategy vastly out-performed students who did not. The goal of this study is to see if TIPP&SEE was truly effective for all learners, not just students who have academic and/or economic advantages. Our objectives were two-fold: (1) to examine the relationships between learner

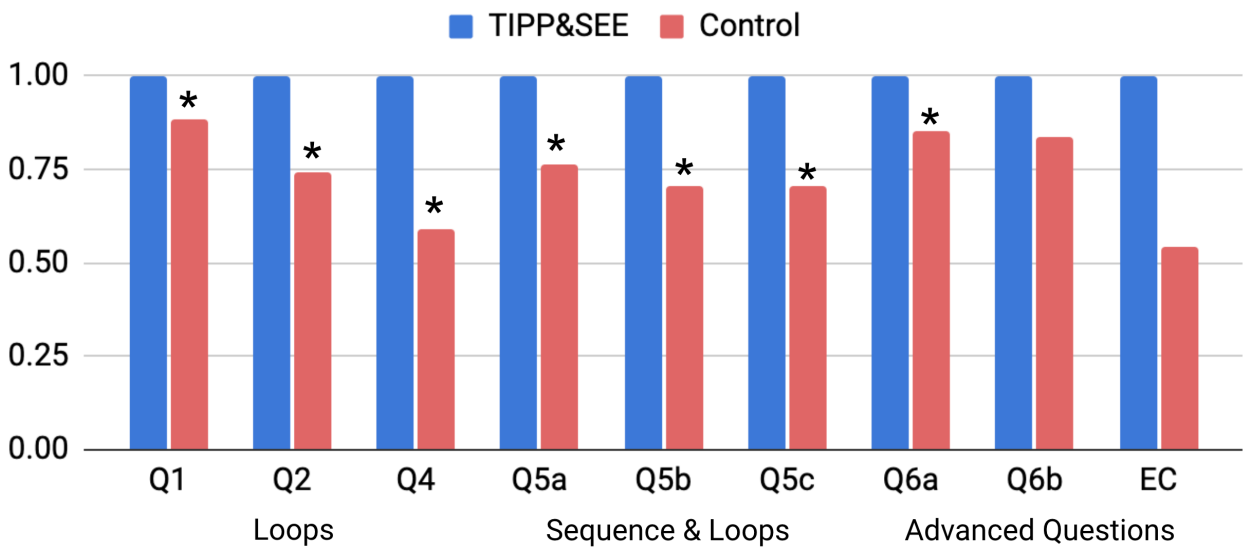


Figure 9.6: Results for Loops & Advanced Questions

characteristics and computer science learning at the primary/elementary level, and (2) to explore meta-cognitive strategy instruction as a method for providing equitable access to high quality CS/CT curricula with positive learning outcomes for all students, including diverse learners. It is only through research like this that traditionally underrepresented and marginalized students and those from under-resourced schools will experience accessible and equitable opportunities in school-based CS/CT. We were motivated by the following research questions:

- To what extent does the meta-cognitive strategy TIPP&SEE support diverse learners in CS/CT instruction?
- In which CS/CT concepts are diverse learners supported by TIPP&SEE?

In this study, I helped with the design of the TIPP&SEE strategy and the development of curricular materials. I also helped with participant recruitment led by our collaborator Prof. Cathy Thomas and supported the computer science undergraduate students who assisted the teachers in Austin Independent School District (AISD). I was in charge of the assessment design and data analysis for this study.

9.2.1 Methods

Study Design

Fifteen teachers were recruited from a large, urban school district in Texas, USA, and underwent the same professional development to teach the Scratch Act 1 curriculum. Eight fourth grade teachers were taught the TIPP&SEE learning strategy. A total of 16 classrooms participated in the study, six of which were bilingual classrooms. Each classroom was assisted by an undergraduate CS researcher. Teachers were randomly assigned to either the TIPP&SEE or the comparison condition, resulting in five English-only and three bilingual classrooms in each condition. Classrooms in the comparison condition were taught Scratch Act 1 without the TIPP&SEE worksheets guiding them through the Use/Modify projects. After excluding students who switched schools or were chronically absent, there were a total of 96 and 88 students in the comparison and TIPP&SEE condition respectively.

Students were identified as economically disadvantaged if they received free/reduced lunch at school. Students who have limited English proficiency, a disability, or were below proficiency in reading and math proficiency were identified through state testing and district-provided demographic data. Some students fulfilled more than one of these characteristics. The distribution of students in each condition is shown in Table 9.2.

Data Analysis

The Scratch Act 1 summative assessments (described in Chapter 4) were scored by two researchers to ensure reliability. To see if TIPP&SEE and/or any of the student categories had an influence on their performance, we transformed our data using the Aligned Rank Transform (ART), which allows for non-parametric factorial analyses, prior to running an ANOVA F-test [99, 251]. A non-parametric transformation was selected due to small sample sizes in the academic challenge categories. Type III sum of squares was used to account

for unequal sample sizes. Estimated marginal means were used for post-hoc comparisons between each group. For statistical significance, we report F and p values for both condition (TIPP&SEE vs Comparison) and academic challenge. We also report the eta squared (η^2) effect size. The effect size indicates the magnitude of the observed effect or relationship between variables [143]. η^2 measures the proportion of the total variance in a dependent variable (DV) that is associated with the membership of different groups defined by an independent variable (IV) [43]. For example, if an IV has a η^2 of 0.25, that means that 25% of a DV's variance is associated with that IV.

9.2.2 Results

We first discuss high-level results, describing overall performance on the two end-of-module assessments of each student category. We then delve deeper into performance in specific concepts.

Overall Results

Finding 1: All student groups performed statistically-significantly better when using TIPP&SEE.

Across all five categories, students using TIPP&SEE performed better than students in the comparison group for both the Events & Sequence and Loops assessments (Table 9.3).

Finding 2: The gap between students with and without academic challenges was narrowed by TIPP&SEE.

Students facing any academic challenge, except for limited English proficiency, still statistically-significantly under-performed students without any challenges in both assessments (Table 9.4). However, the gap between students with and without any academic challenge was smaller in the TIPP&SEE condition compared with the comparison condition (Figures 9.7, 9.8, 9.9, 9.10, & 9.11).

Most notably, post-hoc comparisons revealed that there were no statistically-significant

performance differences between comparison students *without* any academic challenges and TIPP&SEE students *with* economic disadvantages ($p = .66$), disabilities (E&S: $p = .12$; Loops: $p = .69$), and proficiencies below grade level in math (E&S: $p = .63$; Loops: $p = .37$) and reading (E&S: $p = .55$; Loops: $p = .14$). This suggests that TIPP&SEE scaffold CS/CT learning for diverse learners such that they achieve similarly to their peers who do not face academic challenges.

Finding 3: Limited English proficiency was the only student characteristic not associated with assessment performance.

The only exception to these trends was limited English proficiency, which did not have a statistically-significant association in either assessment (E&S: $p = .52$, Loops: $p = .19$). This may be due to bilingual instruction in both conditions. Not only were LEP students taught in Spanish and English, they also had access to Spanish CS materials and could even translate Scratch into Spanish.

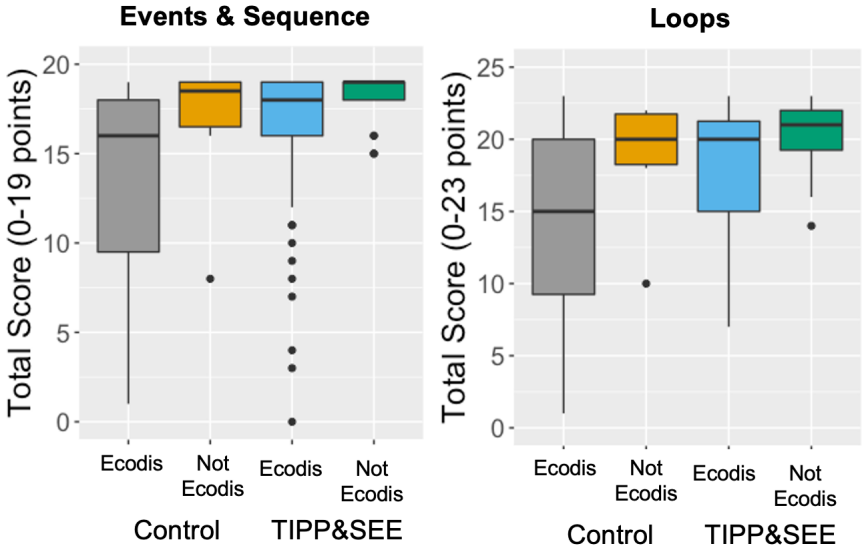


Figure 9.7: Performance of Economically Disadvantaged Students (ECODIS)

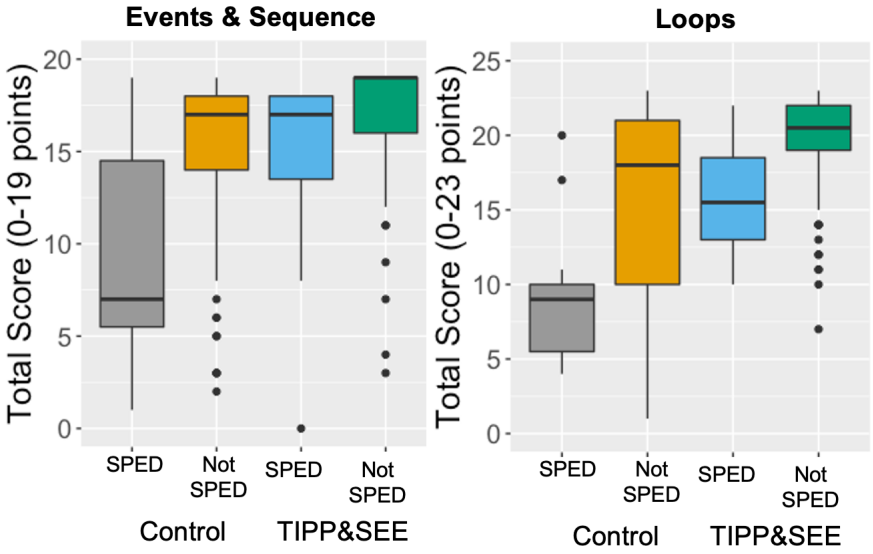


Figure 9.8: Performance of Students with Disabilities (SPED)

Concept-Specific Results

We now turn our attention to the specific concepts covered in the end-of-module assessments, organizing questions based on the results of an exploratory factor analysis.

Finding 4: There were statistically-significant interactions between condition and disability status.

At the concept level, the interaction terms between condition (TIPP&SEE vs Comparison) and special education/disability status were statistically significant for most questions, which limits our interpretation of the data. As such, we do not further discuss them in this section. Potential reasons for these interactions are explored in the next section. In this section, we delve deeper into the other student categories: students with economic disadvantages, students with limited English proficiency, and students performing below grade level in reading and math.

Events

For the two questions on Events (Q4a and Q4b from the Events & Sequence assessment; Table 4.3), students were shown a Scratch stage with two sprites that resulted from a green flag click and asked to identify the script that ran for each sprite.

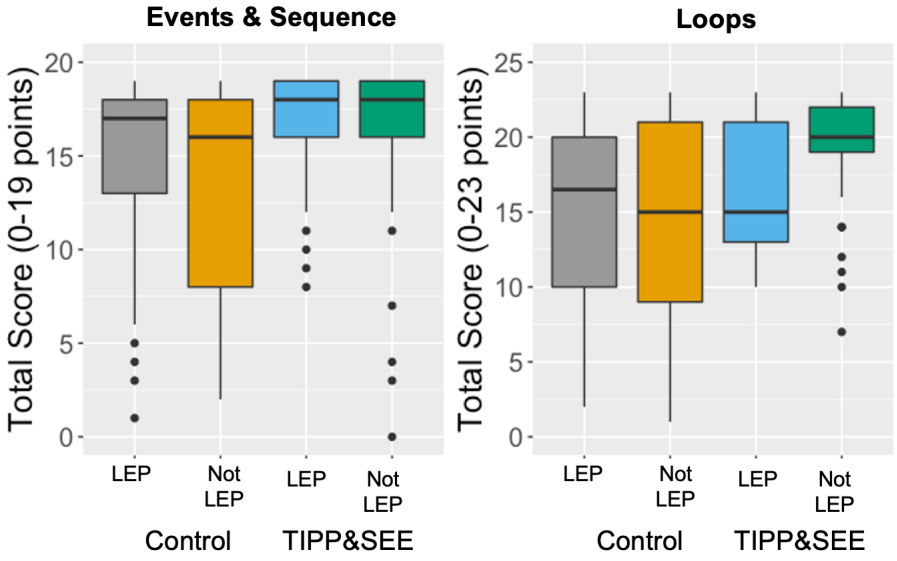


Figure 9.9: Performance of Limited English Proficiency Students (LEP)

Finding 5: LEP status was not associated with Events performance, while economic disadvantage and math proficiency had mixed results. Reading proficiency was associated, regardless of condition.

Just as in the overall results, for students with limited English proficiency, neither LEP status (Q4a: $p = .56$, Q4b: $p = .89$) nor condition (Q4a: $p = .78$, Q4b: $p = .91$) were statistically significant. In contrast, results for students with economic disadvantages and students performing below grade level in math were mixed, where one question would have neither student category nor condition as statistically-significant but the other question would have one of them significant.

Interestingly, students who were below grade level in reading struggled on these questions, regardless of condition (Q4a: $p < .01; \eta_p^2 = .075$; Q4b: $p < .01; \eta_p^2 = .069$). This finding may be further evidence of a trend shown in prior work where a text surface understanding of code was tied to reading comprehension (Chapter 5).

Sequence

In two of the questions on Sequence (Q6 and Q7b from the Events & Sequence assessment), students were shown a script and asked to articulate the order in which the different blocks

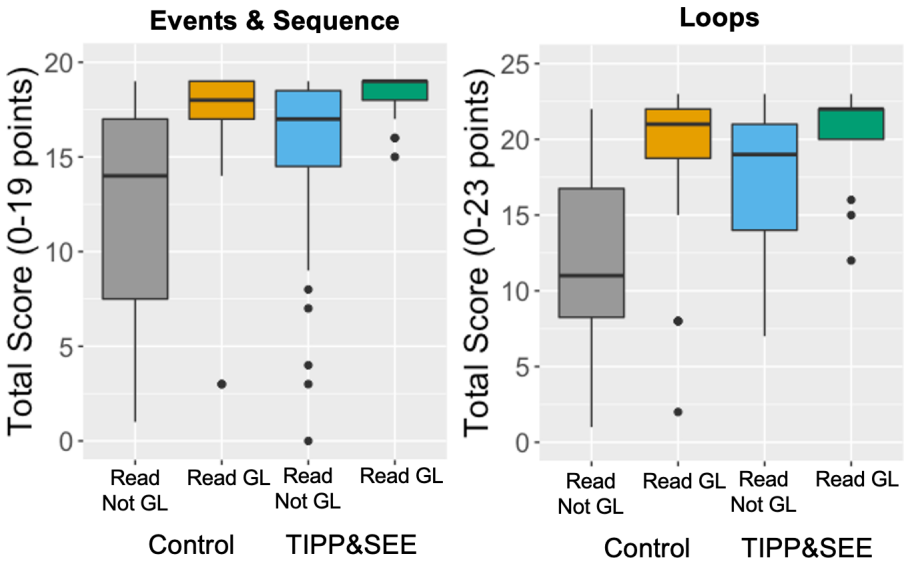


Figure 9.10: Performance of Students Reading Below Grade Level

would run. The remaining three Sequence questions (Q5a, b, c from the Loops assessment) asked about the same script, where a loop was sandwiched between two blocks. Students were asked to identify the blocks that ran before, after, and in the loop.

Finding 6: Sequence results were mixed for students with economic disadvantages, disabilities, and below grade level proficiency in reading and math.

For the remaining student categories, results were mixed, with some of the questions having the condition significant, the student category significant, both significant, or none significant (Table 9.5).

Loops

Q5a, b, and c from the Loops assessment also covered Loops in addition to Sequence. One of the Loops question (Q1 from the Loops assessment) showed students a loop and asked students how many times the loop would repeat. Two other Loops questions (Q2 and Q4 from the Loops assessment) asked students to unroll a loop, but with different answer choices. Q2 asked about a single-block loop repeating 4 times and had the answer choices of the block in the loop repeated 1, 2, 3, or 4 times. Q4 asked about a double-block loop repeating 3 times and had the answer choices of the two blocks alternating 3 times (the correct execution) and

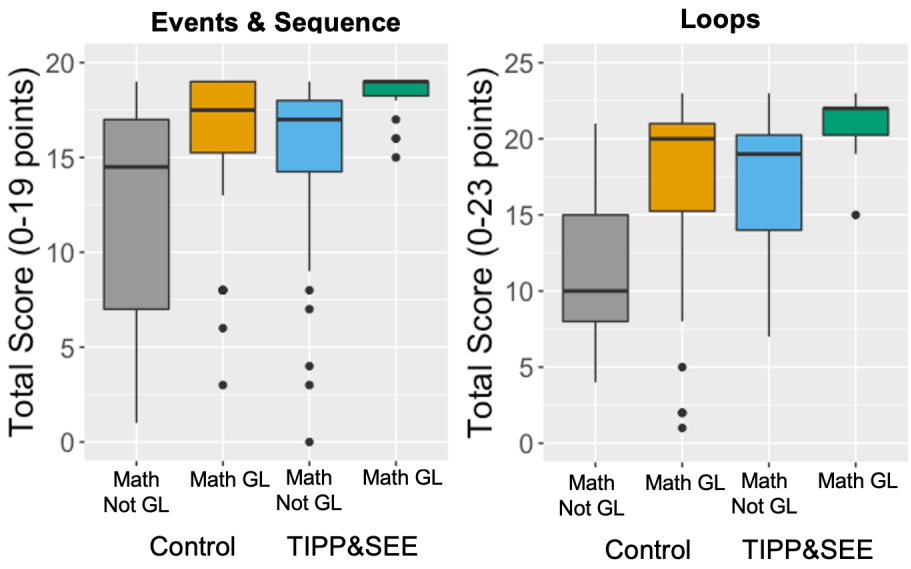


Figure 9.11: Performance of Students with Math Below Grade Level

a script with the first block repeated 3 times followed by the second block repeated 3 times (a common misconception).

Finding 7: Loops results were mixed for students with economic disadvantages, disabilities, and below grade level proficiency in reading and math.

Just like in Sequence, results were similarly mixed for the rest of the student categories, with different combinations of condition and student category found to be statistically significant for different questions (Table 9.5).

9.2.3 Discussion & Implications

We now return to our overarching research questions:

- To what extent does the meta-cognitive strategy TIPP&SEE support diverse learners in CS/CT instruction?
- In which CS/CT concepts are diverse learners supported by TIPP&SEE?

For our first research question, our findings provide preliminary evidence that support the use of meta-cognitive strategy instruction in CS/CT for diverse learners who typically

under-perform on critical academic outcomes, such as the state level assessments employed in this analysis, and on national assessments of math and reading [165]. In this study, CS instruction using the TIPP&SEE strategy to scaffold the Use→Modify→Create framework within a Scratch curriculum for fourth grade students effectively leveled the playing field. This squares with findings from math and science education, where open inquiry was less effective than scaffolded inquiry for students with disabilities [123, 150, 200]. TIPP&SEE enabled students in poverty, students with disabilities, and students who were performing below proficiency on state testing in reading and math to perform similarly to their typically achieving peers on CS tasks.

The only exception to this trend were multi-lingual learners. The performance of multi-lingual learners in bilingual classrooms was not enhanced by exposure to the learning strategy and their performance across instructional conditions was similar. In comparison to their typically developing peers, they slightly under-performed on the Loops assessment ($p < .05$), but did not perform differently on the Events & Sequence assessment ($p = .31$). Although prior studies have shown open inquiry to be less effective for multi-lingual learners [26, 62, 247], limited English proficiency was less of a barrier to their CS/CT instruction with bilingual instruction [203].

For our second research question, results were less definitive. There were statistically-significant interactions between condition and special education/disability status for a majority of the questions. While we balanced the number of students with disabilities in each condition as best as possible (see Table 9.2), a student classified as having a disability could have one of many different kinds of disabilities, ranging from visual impairment to dyslexia. We only had data on *if* they had a disability, but not *what type of* disability. It is possible that TIPP&SEE supported students with certain kinds of disabilities better than others, which would require further investigation.

On the Events questions, students with limited English proficiency exhibited the same

trend as the overall assessment results, while results were mixed for students with economic disadvantages and with below grade level proficiency in math. TIPP&SEE did not do much to support students who were reading below grade level on these questions, suggesting that reading may be a foundational skill to programming.

On questions covering both Sequence and Loops, results were inconclusive for all student categories. There are several potential reasons for this. We may need to look at more specific cognitive factors, such as working memory; these student categories may be obscuring these cognitive factors. We may also need to revise our questions as they may be too high-level or include too many steps, and design more questions that target different levels of the Bloom's taxonomy as our current test blueprint mainly targets understanding. It may also be a reason we have not yet considered; future exploration will be necessary for more conclusive results. While the gap between students with and without an academic challenge narrowed with TIPP&SEE in aggregate, further research is required to identify which concepts are and are not served by the TIPP&SEE strategy, and for which student demographics.

While this is exploratory research and a single study, the promise for fulfilling the goal of CS for All to support diverse learners is encouraging. We hope that learning strategies like TIPP&SEE will help foster meaningful participation in computing through the intentional focus on improving equity and access to CS/CT for all.

9.3 Exploring Student Behavior using TIPP&SEE

The prior studies have shown TIPP&SEE to be linked with positive performance differences in summative CT assessments, as well as with narrowed gaps between students with and without academic challenges. The objective of this study was to explore *why* these performance differences occurred. We sought to answer three research questions:

- To what degree do students follow the TIPP&SEE protocol, and how accurately can they answer the posed questions?

- How does using the TIPP&SEE learning strategy affect student behavior during the completion of Use→Modify and Create tasks?
- Are there any statistical correlations between behavior on the TIPP&SEE worksheets or project attributes and written assessment performance?

In this study, I helped design and develop the TIPP&SEE strategy and associated curricular materials. I also supported the participant recruitment process led by Prof. Cathy Thomas and the computer science undergraduate students who helped the teachers in Austin Independent School District (AISD). I led the theoretical framing, assessment design, and statistical analysis, while Prof. Diana Franklin directed the overall data analysis.

9.3.1 Methods

We analyzed three data sources: computational artifacts, TIPP&SEE worksheets, and the summative assessments described in Chapter 4. To see if either completion rates of project requirement from artifacts or TIPP&SEE worksheet were correlated with assessment scores on individual questions, the Spearman's rank correlation coefficient (ρ) was calculated. Spearman's correlation was used as some of our metrics were on an ordinal, not an interval scale. This test yields a p value for statistical significance.

Computational Artifacts

Student Scratch projects were statically analyzed by a group of undergraduate researchers to extract the completion of requirements in all the projects. These requirements were listed on their project planning worksheets. Some requirements were designed to help students demonstrate the CT concept, while others were designed to encourage creativity (Table 9.6). To see if there were any statistically-significant differences between the TIPP&SEE and control students in their requirement completion rates, we used two statistical tests suitable

for our large sample size. When comparing the proportion of students who completed a specific requirement, the two-proportion z-test was used; we report z and p values. When comparing countable code attributes, such as script length, the ANOVA F-test was used; we report F, p, and effect size η^2. η^2 measures the proportion of the total variance in a dependent variable (DV) that is associated with the membership of different groups defined by an independent variable (IV) [43]. For example, if an IV has a η^2 of 0.25, that means that 25% of a DV's variance is associated with that IV. $p < .05$ was used for statistical significance.

TIPP&SEE Worksheets

Students worked on TIPP&SEE worksheets prior to starting the UM projects. Questions were divided between the three types of questions: Observe, Predict, and Explore (Figure 9.12). Answers were transcribed electronically and analyzed for completion and accuracy by a team of undergraduate researchers. Completion rates of each question type varied due to classroom-level factors, such as instructional time constraints.

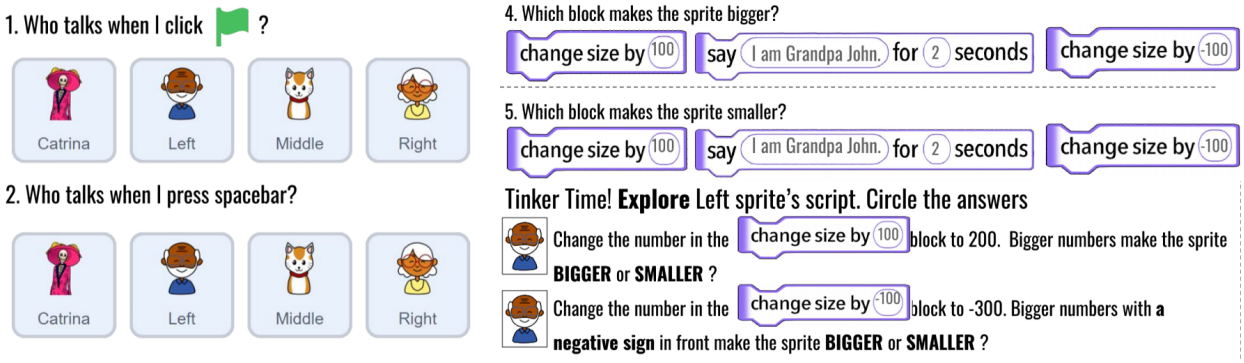


Figure 9.12: Example Observe (left), Predict (top-right), Explore (bottom-left) Questions

9.3.2 Results

We explored student behavior through their artifacts, worksheets, and assessments. Artifact analysis allows us to compare the behaviors of control classrooms to treatment classrooms. Their artifacts were examined for attributes that indicated the fulfilment of the project requirements. Treatment classrooms' TIPP&SEE worksheets were inspected for the completion of the Observe, Predict, and Explore phases. Finally, artifact attributes, worksheet correctness, and worksheet completion rates were analyzed for any correlations with assessment scores.

We present three sets of results: artifact attributes, TIPP&SEE worksheets, and correlations between data sources. Within each section, we highlight important individual findings alongside presentation of the evidence for those findings

Artifact Attributes

We compared the attributes of student projects in two ways: (1) across condition (control vs TIPP&SEE) and (2) across individual classrooms. Each student project was created in the context of either a Use→Modify or Create activity in the curriculum. Control and treatment students had different worksheets for Use→Modify activities, but identical materials for Create activities. We first present the overall results, then we present select detailed results.

Overall Results

Figure 9.13 depicts overall requirement completion rate across the entire curriculum. For each project, the left (blue) bar shows control, and the right (red) bar shows the treatment results.

Finding 1: TIPP&SEE students satisfied either the same or higher percentage of requirements than the control students.

TIPP&SEE students were more likely to complete *all* the project requirements for *5-Block Challenge* ($z = 10.25, p < .01$), *Ofrenda* ($z = 9.34, p < .01$), *Parallel Path* ($z = 9.34, p <$

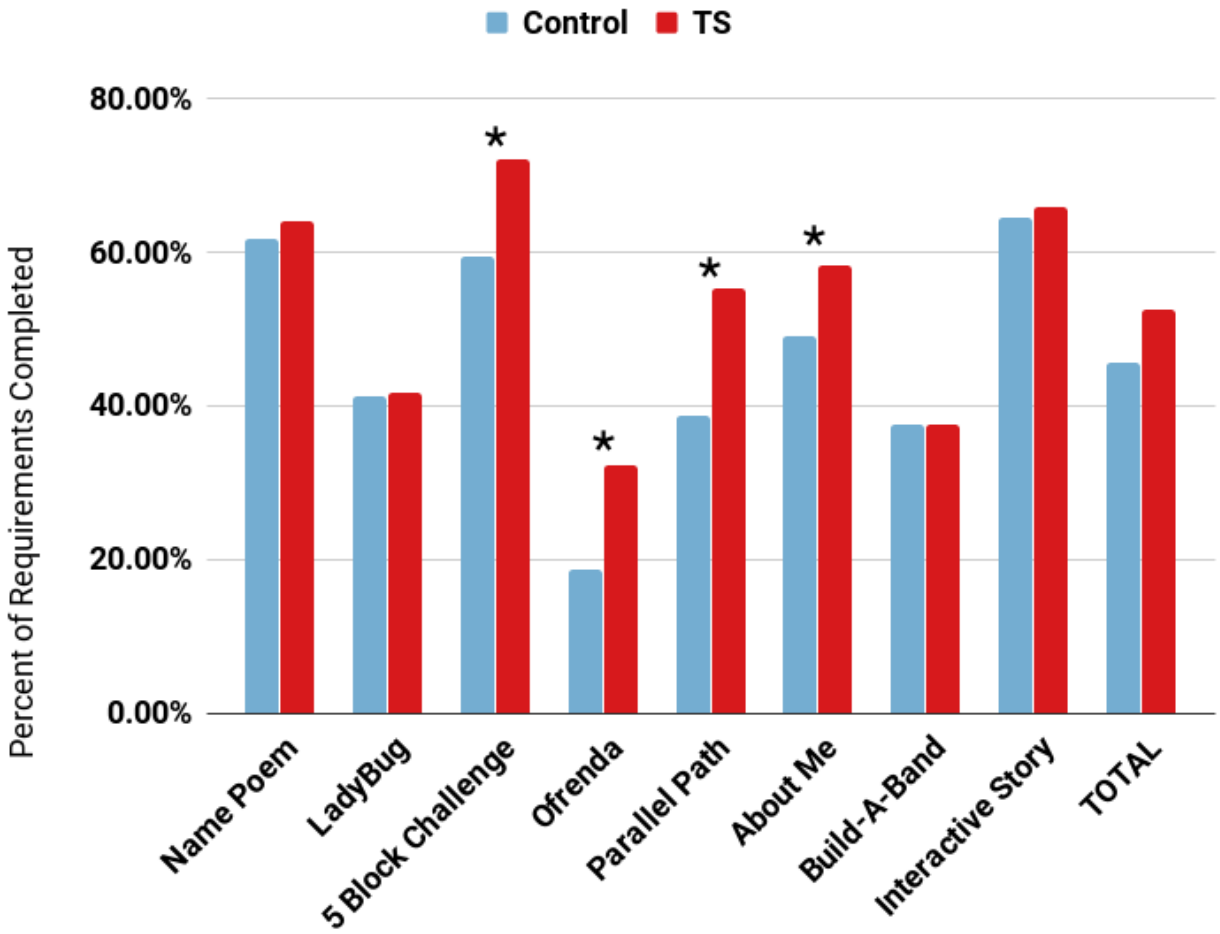


Figure 9.13: Requirement Completion Rate across Condition

.01), and *About Me* ($z = 6.12, p < .01$). Table 9.7 shows the individual requirements for each project where TIPP&SEE students had a statistically-significantly higher completion rate.

Finding 2: Completion rates varied for different classrooms, and with substantial overlap between treatment and control classrooms.

Figure 9.14 breaks down the overall results by classroom, ordered by percentage of requirements completed. We can see that not all treatment classrooms complete more requirements than all control classrooms. However, it is clear that treatment classrooms did better in general.

Finding 3: Individual requirements with higher completion rates by control students do

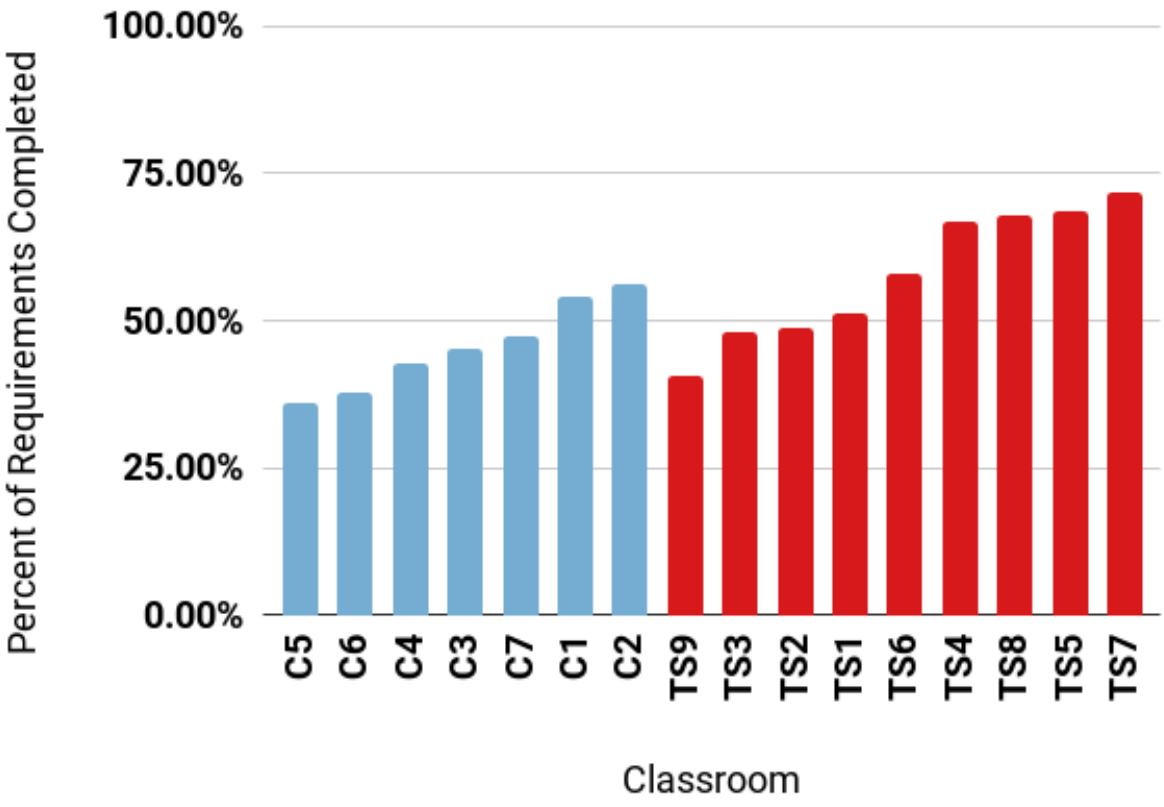


Figure 9.14: Per-Classroom Overall Requirement Completion

not utilize as much coding.

While TIPP&SEE students fulfilled more requirements than control students overall, there were some requirements that more control students completed. These exceptions were mostly in superficial, not programming-related, requirements. For *Name Poem*, a greater proportion of students in the control condition changed the backdrop. In *Build a Band*, a larger percentage of control students added a new sprite and added new blocks that would animate the Cat sprite that was already in the example project, with the difference in adding a new sprite being statistically significant ($z = 2.64; p < .01$). TIPP&SEE students outperformed control students in requirements that were programming-related, as showcased in the *5-Block Challenge*, *Ofrenda*, *Parallel Path*, and *Interactive Story* projects.

Discussion Overall, we see that students in treatment classrooms not only completed more elements (Finding 1) but focused more on programming elements (Finding 3). This shows that one purpose of the learning strategy and worksheets - to help students complete the projects - was successful. Our initial hypothesis was that, overall, completing more of the projects will lead to better performance on written assessments, a correlation we explored in Section 9.3.2.

Per-Project Results

We now examine more closely a subset of the projects. *Ofrenda* was chosen because it represents typical behavior for a project on which students generally did well on the written assessment for both control and treatment groups. *5-Block Challenge* and *Parallel Path* were chosen because they were redesigned based on analysis of the previous year's student work and written assessments. Finally, Interactive story was chosen because it is the culminating project for the curriculum. *Ofrenda* and *Parallel Path* are Use→Modify projects, whereas *5-Block Challenge* and *Interactive Story* are Create projects. We want to find out whether students completed these projects differently according to their group.

Ofrenda Inspired by the Mexican holiday Día de los Muertos (Day of the Dead), the *Ofrenda* Use→Modify project presented students with three ancestors as sprites. Students were then prompted to modify the project by adding their family members as sprites and making them interactive. There are three requirements in the Modify task, two requiring coding (Interaction and Speaking), and one involving changing the sprites' costumes.

In order to illustrate student behavior, we distinguish between fulfilling the requirement on a single sprite (practicing it once) and on all of the sprites (practicing it and completing the assigned task). Figure 9.15 depicts the results, with the top (red) portion of the bar showing the percentage of students who completed the task for a single sprite, and the bottom (blue) portion of the bar showing the percentage who completed the task for all sprites.

Finding 4: More treatment students completed requirements for at least one sprite as well as for all sprites.

The total height of the bars are higher for treatment students, indicating more completion of any sprite, as well as the bottom (blue) portion of the bar, indicating all sprites. This implies that students both demonstrate some understanding and are potentially more thorough in their work when following the TIPP&SEE strategy and worksheets. A statistical analysis revealed that TIPP&SEE students outperformed control students in making both one ($z = 2.47, p < .05$) and all sprites ($z = 2.12, p < .05$) interactive, changing all sprites costumes ($z = 3.42, p < .01$), and making all sprites speak ($z = 3.63, p < .01$). This indicates that TIPP&SEE students better demonstrated their ability to *apply* their knowledge of Events.

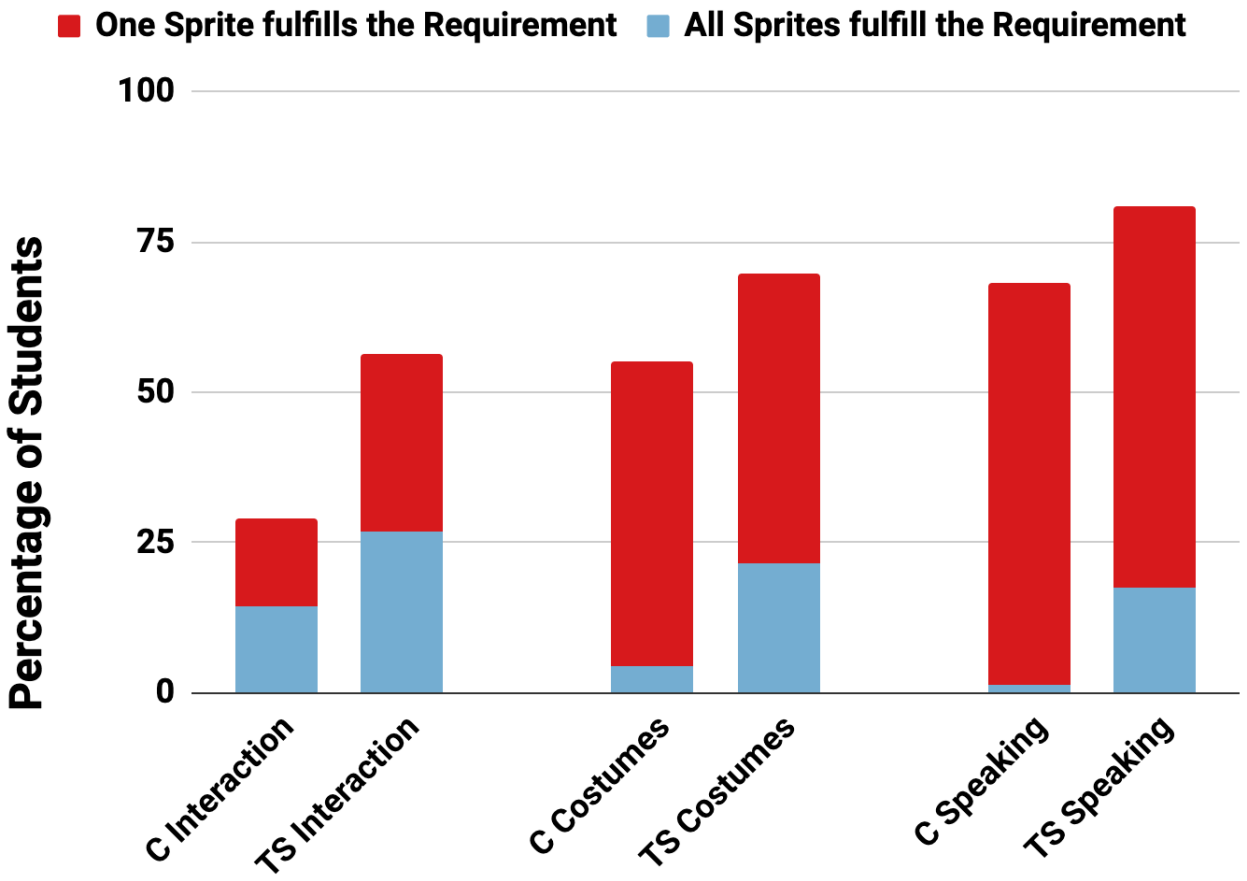


Figure 9.15: Ofrenda Completion Rate across Conditions

5-Block Challenge The *5-Block Challenge* Create project, with identical materials for both groups, prompts students to create a project using only 5 blocks: `When this Sprite Clicked`, `When Green Flag Clicked`, `Wait`, `Say`, and `Glide`. The goal is twofold: encourage students to build with blocks they haven't been explicitly taught, and encourage students to create scripts with multiple action blocks rather than lots of scripts with a single action block. This project was modified from the 10-Block Challenge because analysis of previous years' student artifacts (including final projects) revealed few scripts contained more than a single action block.

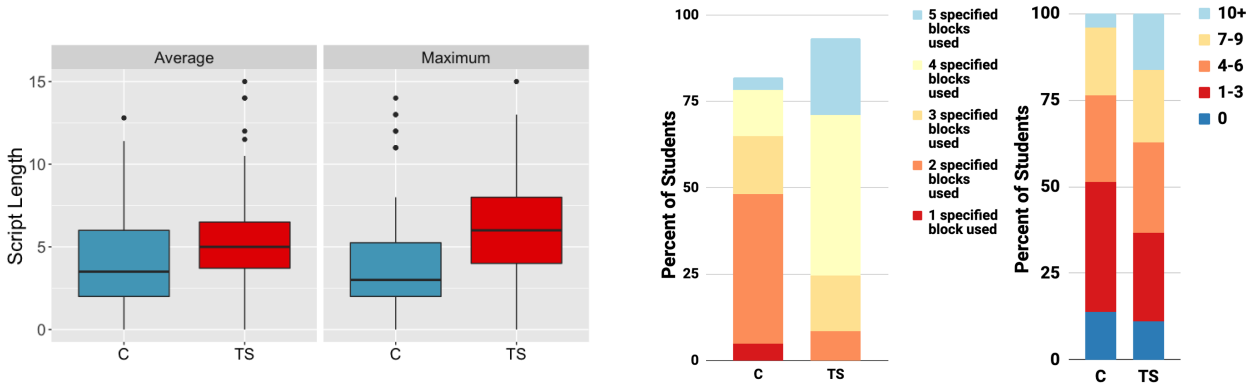
Finding 5: Students in treatment classes created longer scripts, on average, than control classrooms during the 5-Block Challenge.

We analyzed two major statistics, shown in Figure 9.16a. First, we calculated the average script length in each student's artifact. Second, we calculated the length of the longest script in each student's artifact. We can see that treatment classrooms, on average, create longer scripts, with treatment students creating scripts with 5.22 blocks and control students with 3.97 blocks ($F(1, 196) = 9.01, p < .01, \eta^2 = .044$). This means that treatment students, in general, went beyond the minimal two blocks per script to create sequential scripts. The mean maximum was slightly higher in the control groups, at 7.45 vs. 6.27, due to students with incredibly long scripts (over 45 blocks) in one classroom, whereas the median maximum script length was higher in the treatment groups. However, an analysis of variance of the maximum script lengths between the two conditions showed that these differences were not statistically significant ($F(1, 196) = 1.37, p = .24$). Therefore, in general, students in the treatment group created at least one script that was of non-trivial length, showing more practice at creating sequential scripts.

Finding 6: Students in treatment classes used more required blocks, on average, during the 5-Block Challenge.

The left bars on Figure 9.16b shows the percentage of students who used different

numbers of the required blocks. The best case would be entirely blue bars, in which all students utilized all 5 blocks. A majority of treatment students used 4-5 blocks, whereas a majority of control students used 2-3 blocks. While students in the control condition were more likely to use only two specified blocks ($z = 5.63, p < .01$), students using TIPP&SEE used more of the specified blocks. A statistically-significantly greater proportion of treatment students used four ($z = 4.81, p < .01$) and five ($z = 3.46, p < .01$) blocks. Along the Matrix Taxonomy, higher block usage by the TIPP&SEE students suggests that they were better able to *create* artifacts in the context of a cumulative project on Sequence.



(a) Block Lengths across Condition

(b) Block Use for 5-Block Challenge (left) & Interactive Story (right)

Figure 9.16: 5-Block Challenge Results

Parallel Path The *Parallel Path* Use→Modify project was created in response to poor student performance on written assessment questions involving parallel versus sequential code. The project presents students with two sprites that had actions either in sequence or in parallel, depending on which number they pressed. The TIPP&SEE worksheet had students identify what actions were sequential vs parallel and then inspect the corresponding code. Students were then asked to modify the project such that upon a mouse click, each sprite would do two actions in parallel, and when the number '9' key was pressed, both sprites would do an action in parallel. These results are shown because they represent the

most staggering difference in behavior between the two groups.

Finding 7: A significantly larger percentage of TIPP&SEE students satisfied the requirements of Parallel Path than the control group.

Figure 9.17 depicts the percentage of students who completed each requirement. It shows that there was no single requirement that 20% of control students completed, yet for all requirements, at least 45% of TIPP&SEE students completed them. In fact, less than 25% of students in the control group completed any single requirement. In contrast, almost 75% of TIPP&SEE students completed a single requirement, and over 50% of these students completed the entire project. TIPP&SEE students significantly outperformed the control students in every requirement: programming 1 sprite ($z = 7.46, p < .01$) and at least 2 sprites ($z = 5.67, p < .01$) to do two parallel actions on click, programming 1 sprite ($z = 6.77, p < .01$) and at least 2 sprites ($z = 6.06, p < .01$) to act when the '9' key is pressed. TIPP&SEE students were also more likely to fulfil all requirements ($z = 6.67, p < .01$). The TIPP&SEE students' better performance is especially noteworthy because parallelism is a concept with which students commonly struggle [121, 134].

Interactive Story *Interactive Story* is the culminating Create project in this curriculum, designed to encourage students to demonstrate their knowledge of the three CT concepts covered: events, sequence, and loops.

The right two bars of Figure 9.16b illustrate the number of unique blocks that students utilize in their final projects. A greater percentage of the control group used fewer distinct block types, while the TIPP&SEE group used more distinct block types. Most notably, TIPP&SEE student projects were more likely to have at least 10 unique blocks relative to the control student projects ($z = 2.19, p < .05$). Further, TIPP&SEE students outperformed the control students in using either the `switch backdrop` or `when backdrop changes` block to make their backdrop interactive ($z = 2.23, p < .05$).

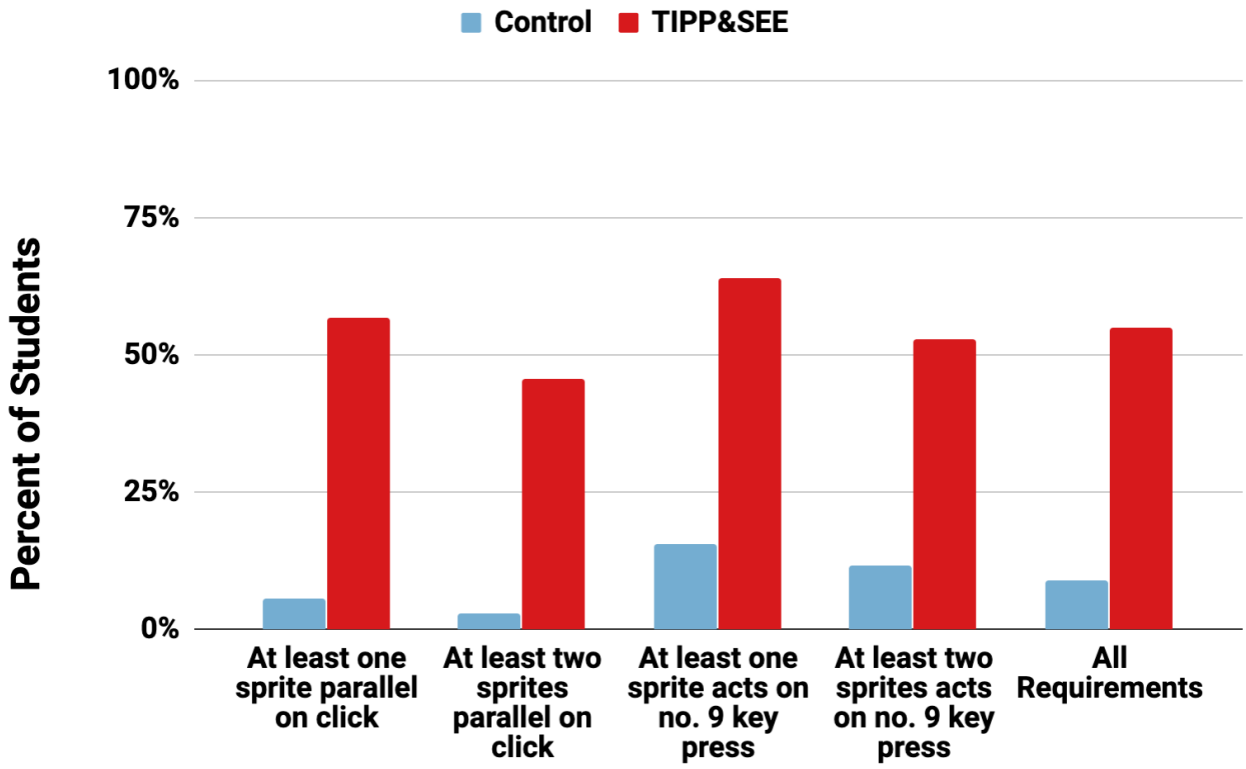


Figure 9.17: Parallel Path Completion Rate across Conditions

TIPP&SEE Worksheets

We now turn our attention to analysis of the TIPP&SEE worksheets. These worksheets were completed only by students in the treatment group; the worksheets for control students presented the project and had the modify tasks listed, but they did not have a set of questions for students to answer.

All figures in this section break up student responses into four categories: Correct, Incorrect, Blank, and No Sheet. The distinction between Blank and No Sheet is that a Blank answer was collected but was not answered by the student, whereas No Sheet indicates that we are missing the entire worksheet for that student.

We begin by exploring student behavior on different types of TIPP&SEE questions. There are three categories of questions we analyzed. The Observe questions are first, asking students to record their observations from running the provided project. All worksheets have

Observe questions. The other two question categories are only on a subset of worksheets. Predict questions ask students to look at the code and predict what blocks caused which actions they observed. Explore questions have two parts. First, they asked students to make a change to the code and run it, such as changing the number in the `wait` block. Next, they record what happened in response, such as whether the sprite moved faster or slower. There are other question categories, but these are the three we analyze.

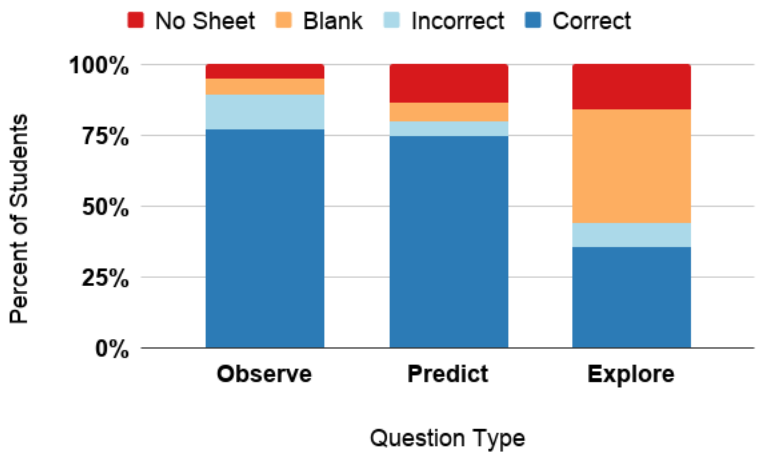


Figure 9.18: TIPP&SEE Worksheet Responses across Question Types

Finding 8: A majority of students completed and correctly answered Observe and Predict questions, while Explore questions were largely left blank.

Figure 9.18 shows the percentage of students that completed and correctly answered questions across all TIPP&SEE worksheets, sorted by the type of question. It shows that, overall, there were few incorrect answers. However, a majority of students did not record answers to Explore questions.

It is unclear if the reason for skipping Explore questions was because students did not follow the Explore prompt or because they did not record their observations. There are several reasons, however, that students could have skipped them. First, because explore questions were only included in a few projects, following and recording explore prompts may not have become a routine. On a related note, students may have needed more scaffolding

with this type of questions, requiring the teacher to model and practice them. In addition, making code changes is a more difficult task than merely answering a question about what one observes or is thinking, so this may have been cognitively difficult for some students.

Correlations between Projects, Worksheets, and Assessments

Having analyzed assessments, project completion, and worksheets independently, we now investigate relationships between them.

Worksheets vs Projects

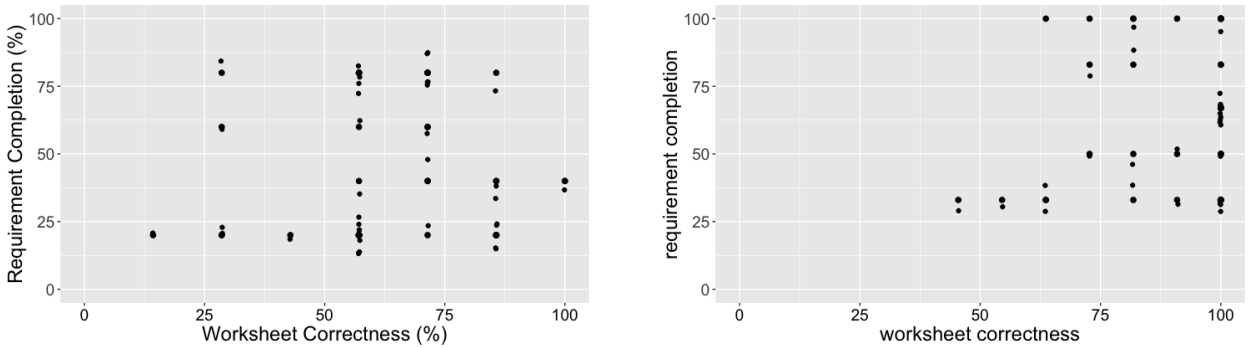
We begin by analyzing correctness and completeness of TIPP&SEE worksheets compared with requirement completion on the projects. Only Use→Modify activities have worksheets. Our question is, does higher completion or correctness of worksheet questions correlate with higher requirement completion on modify tasks?

Finding 9: There was very little correlation between TIPP&SEE worksheets and project completion.

The only UM project with any correlation between worksheet correctness and project completion was *Ofrenda* ($\rho = .33, p < .05$). For the rest of the projects, the distribution of these metrics per student fell into two broad categories. In the first category, worksheet correctness and requirement completion rates were scattered all over the place, such as *Name Poem* and *Ladybug Scramble* (Figure 9.19a). In the second category, these metrics were concentrated in the right half of the plot (i.e. at least 50% worksheet correctness), but do not follow any pattern beyond that, such as *Parallel Path* (Figure 9.19b) and *Build a Band*.

Worksheets and Projects vs Assessments

We now consider any correlations between worksheet completion, worksheet correctness, project completion, compared with written assessments.



(a) Ladybug Scramble (b) Parallel Path

Figure 9.19: Worksheet Correctness vs Project Completion

Finding 10: There was very little correlation between project attributes, worksheets, and assessments.

The correlations that were statistically significant were relatively weak and came from the *Name Poem*, *Ladybug Scramble*, and *Ofrenda* projects.

In *Name Poem*, the first Sequence Use→Modify Project, there was a weak correlation between requirement completion and scores on a question from the Events and Sequence assessment ($\rho = .33, p < .05$). This question showed two sprites with `Say` bubbles on the Scratch stage, triggered by clicking the green flag. Students were then asked to identify the script that belonged to one of the sprites.

In *Ladybug Scramble*, the second Sequence Use→Modify Project, there were weak correlations between TIPP&SEE worksheet *completion* and a two-part question on Events and Sequence (a: $\rho = .33, p < .05$, b: $\rho = .34, p < .05$). This question presented students with a script; the first part asked students to identify the event that triggered the script and the second part asked students to describe the sequence of blocks in the script. There was also a weak correlation between the second part of this question and the overall project completion rate ($\rho = .32, p < .05$). We also found weak correlations between worksheet *correctness* and two questions: one asked students to identify the last `Say` block in a script ($\rho = .40, p < .01$) and another asked them to identify the scripts triggered by the `Green`

`Flag`($\rho = .34, p < .05$).

9.3.3 Discussion

We now revisit our overarching research questions and relate our individual findings to these questions.

To what degree do students follow the TIPP&SEE protocol, and how accurately can they answer the posed questions?

Students followed the Observe and Modify closely and, for the most part, answered the questions accurately. A majority of students, however, did not complete the Explore questions when they are present. We cannot tell if students were truly disengaged from the Explore questions or if students were exploring implicitly, which can occur with metacognitive strategies. Therefore, further research should address either improving the participation for Explore questions or determining that they are not useful for student learning.

How does using the TIPP&SEE learning strategy affect student behavior during the completion of Use→Modify and Create tasks?

There was a significant difference in behavior, overall, between control and treatment students. We had findings on a variety of measures, including project requirement completion, length of scripts, and required block usage. The results from *Parallel Path* are particularly staggering, with treatment students completing requirements at about 8-10 times the rate of control students. Students in the treatment group stayed on task much better than control students, even on Create projects in which the materials were identical. This finding suggests that treatment students were more capable of applying their new knowledge, the first 'production' step in the Matrix Taxonomy, and that they benefited from the Zone of Proximal Flow encouraged by the curriculum design. In addition, when looking at the *5-Block Challenge* and *Interactive Story* results, TIPP&SEE students were better able to *create* more complex projects, the highest 'production' level in the Matrix Taxonomy.

This leads to an interesting question - is TIPP&SEE a learning strategy or an instructional strategy? We have strong evidence that it leads to positive outcomes from an instructional perspective. That is, when students follow this sequence of actions, mediated by a worksheet, it leads to positive outcomes. However, whether the students internalize this into a sequence they can complete without the worksheet, which would make TIPP&SEE a *meta-cognitive* learning strategy, is a question this study does not address. The cognitive aspects are harder to measure, and therefore harder to evaluate.

Are there any statistical correlations between behavior on the TIPP&SEE worksheets or project attributes and written assessment performance?

We find very few statistical correlations between any of the behavioral measures: individual requirement completed, percentage of requirements completed, worksheet questions completed, and individual written assessment question performance.

The lack of correlations between project attributes and assessments is not entirely surprising. On the Matrix Taxonomy, project attributes reflect the 'Producing' dimension, while assessments reflect the 'Interpreting' dimension; it is possible for both dimensions to develop independently [80]. Further, Brennan et al. [25] have shown that students frequently use code that they do not fully understand. Another prior study also revealed that student artifacts can have false positives, where students use code that they do not understand, and false negatives, where students understand a concept but do not use related code constructs (Chapter 8). Students may have run out of time to include these code constructs or simply did not see the need for those constructs in their projects.

In contrast, the fact that the worksheet behaviors (both completeness and correctness) were hardly correlated with the assessments was more unexpected, as both reflect the same 'Interpreting' dimension of the Matrix Taxonomy. Previous studies have found relationships between formative activities or assignments and learning in Scratch [88, 225]. These activities and assignments varied widely in structure. Even within our curriculum, the TIPP&SEE

worksheets differed in structure as well. The influence of TIPP&SEE worksheet design on learning merits further exploration.

Question	$F(1, 184)$	η^2
	Scratch Basics x Remember	
Events & Sequence Q2	3.95	–
Events & Sequence Q3	7.27**	.038
	Events x Understand	
Events & Sequence Q4a	2.95	–
Events & Sequence Q4b	2.86	–
	Sequence x Understand	
Events & Sequence Q6	6.86 **	.036
Events & Sequence Q7	10.93**	.056
Loops Q5a	11.8**	.061
Loops Q5b	13.8**	.071
Loops Q5c	13.8**	.071
	Loops x Understand	
Loops Q1	7.92**	.042
Loops Q2	17.26**	.087
Loops Q4	25.9**	.13
Loops Q5a	11.8**	.061
Loops Q5b	13.8**	.071
Loops Q5c	13.8**	.071
	Advanced Questions	
Loops Q6a	7.49**	.039
Loops Q6b	1.01	–
Loops EC	3.59	–
* $p < .05$,	** $p < .01$	

Table 9.1: Results from the ANOVA F-Test with Questions Organized by CT Concept and Bloom's Level

	TIPP&SEE	Comparison
Economically Disadvantaged	70	91
Special Education/Disability	16	15
Limited English Proficiency	25	52
Below Grade Level in Reading	54	46
Below Grade Level in Math	55	59

Table 9.2: Diverse Students in Each Condition

	E & S		**Loops**	
	$F(1, 181)$	η_p^2	$F(1, 178)$	η_p^2
Economic Disadvantage	8.06**	.043	11.92**	.063
Disability Status	21.25**	.11	19.53**	.098
Limited English Proficiency	18.93**	.095	17.23**	.088
Below Grade Level in Reading	21.64**	.11	32.92**	.16
Below Grade Level in Math	9.95**	.052	36.52**	.17

$^{*}p < .05; ^{**}p < .01$

Table 9.3: Significance Values for *Condition (TIPP&SEE vs Comparison)* in each Student Category

	E & S		**Loops**	
	$F(1, 181)$	η_p^2	$F(1, 178)$	η_p^2
Economic Disadvantage	10.76**	.056	8.72**	.047
Disability Status	25.26**	.12	27.96**	.14
Limited English Proficiency	–	–	–	–
Below Grade Level in Reading	54.48**	.23	64.31**	.27
Below Grade Level in Math	34.05**	.16	53.92**	.23

$^{*}p < .05; ^{**}p < .01$

Table 9.4: Significance Values for each *Student Characteristic (Disability, LEP, etc)*

		Condition		Category	
		F	η_p^2	F	η_p^2
Economic Disadvantage					
Sequence	E&S: Q6	8.58**	.045	8.38**	.044
	E&S: Q7b	13.99**	.072	18.59**	.093
	L: Q5a	—	—	—	—
Sequence & Loops	L: Q5b	17.43**	.089	4.56*	.025
	L: Q5c	—	—	5.07*	.0028
	L: Q1	5.98*	.033	—	—
Loops	L: Q2	—	—	5.48*	.029
	L: Q4	—	—	8.45**	.0045
Limited English Proficiency					
Sequence	E&S: Q6	18.22**	.091	5.01*	.027
	E&S: Q7b	15.31**	.078	10.59**	.055
	L: Q5a	—	—	—	—
Sequence & Loops	L: Q5b	—	—	4.09*	.022
	L: Q5c	53.17**	.23	—	—
	L: Q1	25.19**	.12	13.25**	.069
Loops	L: Q2	26.64**	.13	5.46*	.029
	L: Q4	29.65**	.14	17.55**	.089
Below Grade Level in Reading					
Sequence	E&S: Q6	7.11**	.038	20.71**	.10
	E&S: Q7b	8.65**	.046	29.86**	.14
	L: Q5a	12.01**	.064	36.44**	.17
Sequence & Loops	L: Q5b	8.99**	.049	21.67**	.11
	L: Q5c	8.60**	.047	19.87**	.10
	L: Q1	7.05**	.039	—	—
Loops	L: Q2	42.25**	.19	24.69**	.12
	L: Q4	24.10**	.12	8.79**	.048
Below Grade Level in Math					
Sequence	E&S: Q6	8.56**	.045	11.83**	.062
	E&S: Q7b	16.94**	.086	22.95**	.11
	L: Q5a	20.50**	.10	31.53**	.15
Sequence & Loops	L: Q5b	—	—	30.13**	.15
	L: Q5c	—	—	30.13**	.15
	L: Q1	5.25*	.028	—	—
Loops	L: Q2	50.8**	.22	22.01**	.11
	L: Q4	39.29**	.18	31.75**	.15

*$p < .05$;** $p < .01$

Table 9.5: Significance Values for Sequence & Loops Questions

Project	Requirements
Name Poem	Modify at least half the sprites
	Modify backdrop
	Avg Script Length at least 2
Ladybug Scramble	Ladybug eats at least 1 aphid
	Use `Eat Aphid` Block
	Use `Move Steps` Block
	Use `Turn` Block
5-Block Challenge	Only use the 5 required blocks
	Add new backdrop
	Add at least 2 sprites
Ofrenda	Modify `Say` block for at least 1 sprite
	Modify at least 1 sprite's costume
	Add interactivity for at least 1 sprite
Parallel Path	At least 1 sprite has parallel actions on click
	2 sprites have actions on "9" key press
About Me	At least 1 sprite
	At least 1 interactive sprite
Build a Band	Add a script for guitar
	At least 1 new sprite
	at least 1 new sprite with a script
	Cat sprite is animated
Interactive Story	Interactive backdrop
	At least 1 sprite with a script
	At least 1 event block
	At least 1 loop block

Table 9.6: Scratch Act 1 Project Requirements

Project	Attribute	z
Ladybug Scramble	Ladybug eats 1 aphid	2.47*
5 Block Challenge	Used 2 specified blocks	5.63**
	Used 4 specified blocks	4.81**
	Used all 5 specified blocks	3.46**
Ofrenda	1 interactive sprite	2.47*
	More than 1 interactive sprite	2.12*
	All sprites with a different costume	3.42**
	All sprites have a different `Say` block	3.63**
Parallel Path	1 sprite with parallel actions on click	7.46**
	2 or more sprites with parallel actions on click	6.57**
	1 sprite acts on "9" key press	6.77**
	2 or more sprites act on "9" key press	6.06**
About Me	Has a `Say` block	2.38*
	Has an interactive sprite	3.51**
Build a Band	Modified scripts for at least 1 sprite	3.24**
Interactive Story	Interactive Backdrop	2.23*
	Using at least 10 blocks	2.19*
* $p < .05$	** $p < .01$	

Table 9.7: Attributes with Significant TIPP&SEE Out-Performance

CHAPTER 10

DIAGRAMS AS A SCAFFOLD FOR DECOMPOSITION

While TIPP&SEE is broadly analogous to code reading and tracing strategies at the university level, the objective of these studies is to explore another commonly used practice in university computing instruction, *diagramming*, as a strategy for younger learners [137]. More specifically, we studied a diagram designed to scaffold the decomposition of conditional interactions between sprites in Scratch. To accomplish conditional interactions in Scratch, students would need to incorporate their knowledge of sequence, repetition, and conditionals from prior modules in Scratch Encore.

For our diagramming studies, we focus on a module called *Decomposition by Sequence*. It was designed based on the Decomposition learning trajectory [193], targeted at Scratch programming language-specific constructs. In particular, this module focuses on a sequence of events across multiple sprites, where different actions were triggered by between-sprite interactions for which Scratch provides sensing blocks (touching color or touching sprite). The role of the diagram in this module is to scaffold students' planning of their projects by encouraging the decomposition of their sequence of events. The learning goals for this module are for students to be able to:

- decompose a sequence of events into separate actions and their triggering events,
- create scripts that will trigger the action of one sprite dependent on the action of another sprite,
- use sensing blocks to stop and start actions, and
- plan and create an animation based on a set of events and actions.

In this chapter, we outline two studies in Chicago Public Schools that explore the role of this diagram in the *Decomposition by Sequence* module, the first with teachers and the

second with students. The first study revealed that when using a vertical diagram, teachers were more able to decompose a sequence of events and to make connections between CT concepts. The second study showed that students performed similarly with a text-based and diagram-based scaffold, with implications for the design of diagram-based scaffold for this age group.

10.1 Diagram Design

A plethora of diagrams exists in computing, from control flow diagrams to software engineering design patterns, but these diagrams were designed for more mature audiences and to encapsulate more complex topics. Thus, we also draw on education research for guidance on strategy design for younger learners. For example, the Draw-It Problem Solving Cycle is a strategy to support students with learning disabilities in solving word problems [239] (see Chapter 2 for a more complete review of related work).

10.1.1 Motivation

Teachers and students had struggled with the diagram that was previously used in the Decomposition by Sequence module (Figure 10.1a). In the first lesson of the module, students were shown a video of a sequence of events, such as player running towards a soccer ball and kicking into a goal. After watching the video, students would complete a partially filled diagram based on the sequence of events that they watched with the guidance of their teacher. The objective of the diagram was to scaffold the decomposition of the sequence of events into "events" and "actions" for each sprite. In this module, "events" encompass more than just the Scratch-defined event blocks; "events" are broadly defined as situations where a condition starts or stops an action of a sprite. In this module, sensing blocks, such as the `touching` block, are introduced as way to start and stop sprite actions when placed in a conditional.

Conversations with teachers revealed that they were especially nervous when teaching the first lesson of Decomposition by Sequence. Classroom observations also indicated a heavy-handed approach when teaching the module, where teachers would tell students what to write in each box and it was not clear if the diagram was truly scaffolding student understanding. As a result, student worksheets based on the example sequence of events were remarkably uniform. However, when students worked on a diagram to help them plan their culminating project for this module, they frequently mixed up the events and actions or did not decompose them sufficiently. Students also mixed up which events and actions belonged to which sprites and the order in which events and actions occurred.

10.1.2 Design Process

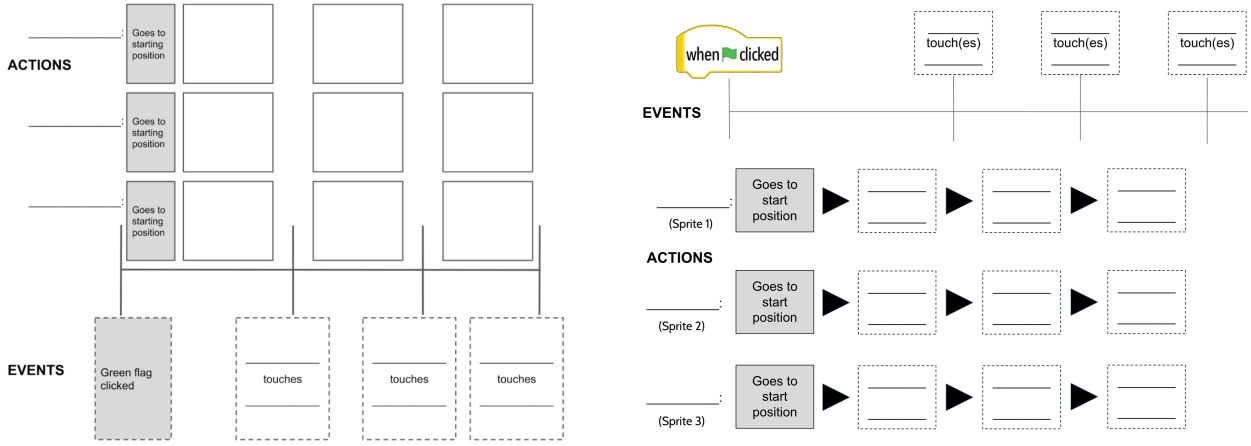
Based on teacher feedback, classroom observations, and prior student work, we revised this diagram, with existing CS diagrams serving as a subject matter guide and diagramming strategies from math education serving as a pedagogical guide. This diagram underwent several rounds of revision. Through weekly meetings over the span of three months, the diagram was iteratively refined through feedback from researchers and practitioners. Lastly, they were sent to Scratch Encore lead teachers, experienced teachers who serve as consultants for other teachers adopting the curriculum, for a final round of feedback and revisions.

10.1.3 Diagram Revisions

From the previous diagram, we preserved the separation between events and actions as the key decomposition method, the timeline as an indication of chronological events, the lines for each sprite, and the color-coding of gray for complete boxes and white for incomplete boxes. The timeline was moved to the top of the worksheet so that it would be less likely for students to miss. Mad libs-style suggestions under the blanks for each sprite were added to make the structure of one-sprite-per-line clearer to students. Arrows were also added

between the different actions to better indicate directionality, similar to control flow diagrams in university computing.

We also designed two versions of the revised diagram in horizontal and vertical orientations. The horizontal version, which is read from left to right, aligns with representations of timelines from other subjects [28] and is more similar to the way students would read natural language text in English. In contrast, the vertical version, which is read from top to bottom, is akin to the way students would read programs and aligns with the code structure of Scratch. Reading direction has been found to be associated with directionality in motion perception [159], spatial processing [199], perceptual span (area of effective vision) [110], and other neurocognitive processes [115]. By comparison, code reading direction and its effects are not as well-understood. Prior work has developed a technique to study code reading [34] and identified differences in reading natural language and code [33]. With teachers, we explored how orientation and in turn, reading direction influences how effective our diagrams are at scaffolding the learning of decomposition.



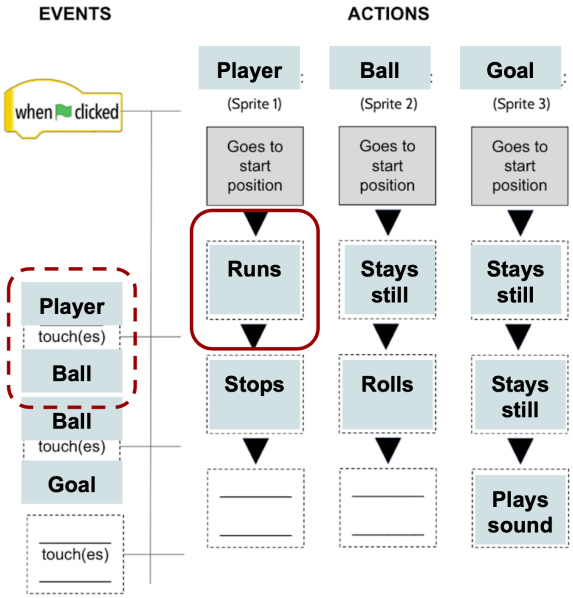
(a) Previous

(b) Revised

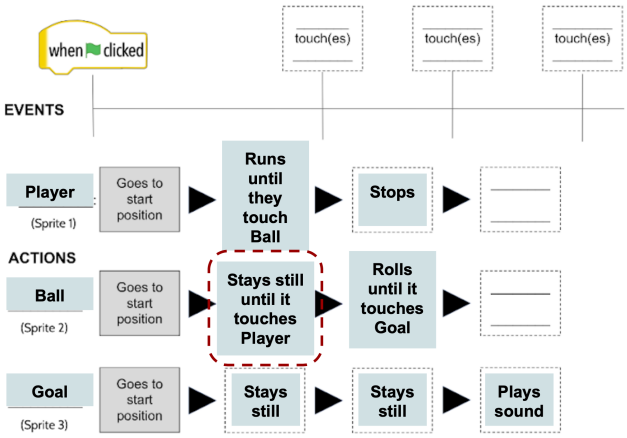
Figure 10.1: Diagrams used in Decomposition by Sequence

Figures 10.2a and 10.2b depict examples of the vertical and horizontal "Create" diagrams, respectively. The blue boxes represent student input. In this example, a student is planning to program a sequence of events from a soccer game where a player runs and kicks

a ball into a goal with spectator cheers. The events in this interaction are when the player touches the ball and when the ball touches the goal. The player's actions are to run until they touch the ball. The ball's actions are to stay still until the player touches it, after which it rolls until it touches the goal. The goal's actions are to stay still until the ball touches, after which it plays a celebratory sound. The scripts for the player, ball, and the goal are shown in Figure 10.3.



(a) Vertical with Complete Decomposition



(b) Horizontal with No Decomposition

Figure 10.2: Diagrams with Different Levels of Decomposition

10.2 Role of Visual Orientation in Designing Diagrams for CT Development in Teachers

With the global growth of computing education at younger ages [103], it is critical that computing instruction supports all learners, and that, even more crucially, teachers are equipped to provide such instruction. One way to support all learners is to provide them with sufficient scaffolding as they are introduced to new concepts. Other discipline-based education research fields, such as math [36, 171, 239], reading [28, 240] and science [171], are

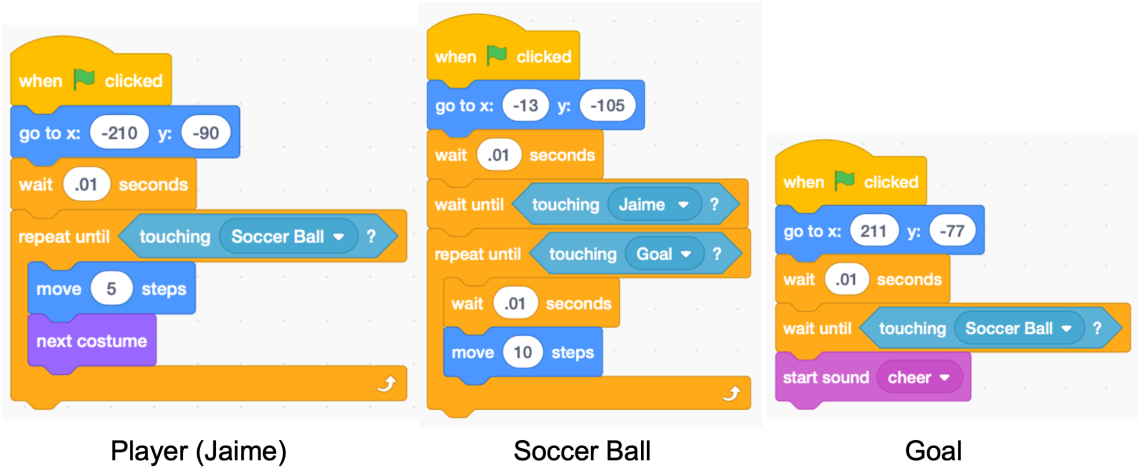


Figure 10.3: Code for Each Example Sprite

rich with instructional scaffolds that can potentially be adapted for computing. Diagramming is one such strategy. Although diagrams are a staple in university computing [137], little is known about its use in K-8 computing — how they should be designed, which concepts are they appropriate for, how they support both students and teachers, among other avenues for exploration.

To address this gap, we investigated two different orientations of a diagram, vertical and horizontal, in a virtual teacher professional development. In particular, we focus on the following research questions:

a) How does diagram orientation influence the learning of decomposition in teachers?

b) How does diagram orientation influence the development of technological pedagogical content knowledge (TPACK) in teachers?

In this study, I led the development of the strategy and accompanying classroom materials, study design, assessment design, data collection and data analysis. I also helped with the teacher professional development of the module in which the strategy will be tested, as well as teacher recruitment which was led by our lab's School Development Specialist Donna Eatinger. Donna was also my second coder for any qualitative analyses.

10.2.1 Methods

Study Context

In Summer 2020, 45 teachers participated in a virtual professional development (PD) for Scratch Encore. Scratch Encore is an intermediate Scratch curriculum designed for students with a year of introductory programming experience [78]. The curriculum is comprised of modules covering computational thinking concepts such as sequence, repetition, conditionals, decomposition, and variables [193, 197, 194].

The PD covered one module per week, with a 30 minute synchronous introduction and an hour-long collaborative coding session in small groups of 4-6. Teachers were encouraged to work through the materials asynchronously between the two sessions. All materials that were previously designed for printed use were adapted for virtual instruction using Google forms and slides.

In the *Decomposition by Sequence* module, we designed horizontal and vertical versions of a diagram to scaffold the learning of decomposition. As prior experience can influence how much scaffolding they would need, teachers were assigned to either the horizontal (H) or vertical (V) condition such that each condition would have the same median years of computer science teaching experience. In the horizontal condition, there were 22 teachers, with a median of 4 years of experience. In the vertical condition, there were 23 teachers, with a median of 4.25 years.

To fill out the diagram in the "Use/Modify" worksheet for this module, teachers first watched a video that demonstrated a complete Scratch project with a sequence of events. Based on the video, they completed a partially-filled out diagram that decomposes the sequence of events. After they filled out the diagram, the worksheet directed them to an incomplete version of the demonstrated Scratch project, which they explored using the TIPP&SEE learning strategy. Using their completed diagrams to support their coding, teachers were

then prompted to modify the project such that it worked like the demonstrated project. In the "Create" worksheet, teachers were prompted to plan their culminating projects with the help of an empty fill-in-the-blank diagram (Figures 10.2b & 10.2a).

Data Analysis

Diagram Worksheets

We analyzed teachers' diagrams in the "Create" worksheet for evidence of decomposition, where they planned their own final project. Two researchers developed a coding manual of categories that described the extent to which they decomposed the sequence of events — separated all the events and actions (complete decomposition), separated at least one event or action (partial decomposition), or did not separate events and actions (no decomposition). Figure 10.2a depicts a complete decomposition, with the events (e.g. "when Player touches Ball") to the left of the timeline and actions (e.g. "Runs") to the right. Figure 10.2b depicts no decomposition with the events and actions together below the timeline (e.g. "Ball stays still until it touches the Player").

The manual also classified characteristics of responses in the "event" and "action" boxes in the diagram. In Figure 10.2a, the event boxes are the boxes to the left of the vertical timeline while the action boxes are to the right. In Figure 10.2b, the event boxes are the boxes above the horizontal timeline while the action boxes are below. Correct responses would be writing one event in an event box (e.g. box in the dotted red circle from Figure 10.2a) and one action in an action box (e.g. box in the solid red circle from Figure 10.2a). Common incorrect responses included writing in an action box: an event (e.g. box in the dotted red circle from Figure 10.2b), a sprite (e.g. if they wrote "Player" in that box), and multiple actions (e.g. if they wrote "stays still then rolls" in that box).

Two researchers coded each worksheet separately and then resolved any disagreements through discussion for 100% agreement. To see if there was a dependence between condition

and the extent of decomposition, we used a Chi-square test of independence; we report χ and p values. This test, however, was not appropriate for their response characteristics as the expected value for each cell was less than five [151].

Scratch Projects

Features of teachers' culminating "Create" Scratch projects were automatically scraped to determine if they fulfilled the assigned requirements. There were five requirements: adding a new backdrop, using at least 3 sprites, using the `go to x: y:` block in at least 2 sprites (for initialization), animating at least 2 sprites, and using a `repeat until` or `wait until` `blocks` to program a sequence of events. Additionally, there were two extra extensions that teachers could complete: adding a new event to a third sprite and using a sound block. Lastly, we also checked if teachers used `broadcast/receive` blocks as it was a frequently mentioned theme in interviews. To see if there was a statistically-significant difference between the two conditions' completion rates, we conducted the Chi-squared test on proportions, from which we provide χ and p values.

Semi-structured Interviews

At the end of the PD (1 month after the module), retrospective semi-structured interviews were conducted with some teachers to learn more about their mental models of their final Scratch projects and to better understand to what extent diagrams scaffolded their learning. 13 teachers in each group (horizontal vs vertical) were interviewed for about 15-20 minutes and were selected based on availability. Teachers were able to reference their completed diagrams and Scratch projects throughout the interview. The interview protocol is shown in Table 10.1.

Interview transcripts were first open coded by two researchers to identify emerging themes. Based on these themes, a qualitative coding manual was developed, covering CT concepts, the levels in the SOLO taxonomy, and aspects of the TPACK framework (Chapter 3). Two researchers coded each interview separately and then resolved any disagreements

through discussion for 100% reliability. The Chi-square test of independence was not appropriate to use on the themes identified in the interviews as the expected value for each cell was less than five, with some cells even being zero [151].

Topic	Questions
Warm Up	Can you show me what your project does?
	Was this diagram helpful to you in planning your project? What went well? What was confusing?
	Do you have any suggestions for improvement for this diagram?
Diagram	Can you explain your thinking as you filled out this diagram?
	How did you fill it out for *Sprite X*?
	How did you decide what to write in this box?
	What went well?
	What was confusing?
	Can you explain how you used this diagram while coding your Scratch project?
	How did you program *Sprite X*?
	How did you program *event/action X*?
	What went well?
	What was confusing?
TPACK	How would you explain how your project works to your students?
	How would you explain how to fill out this diagram to your students?
	Do you think this diagram would be helpful for your students?
	Do you have any suggestions for student-facing diagrams?

Table 10.1: Teacher Interview Questions

10.2.2 Results

We first present results from analyzing diagram worksheets and end-of-module projects, followed with results from teacher interviews.

Finding 1: 77.8% of teachers in the vertical condition decomposed the sequence of events in their diagrams, either completely or partially, compared with 55.6% of teachers in the horizontal condition.

Figure 10.4 depicts the proportion of teachers who did not separate events and actions, separated at least one event or action, and separated all events and actions when decomposing a sequence of events. A majority of teachers in the vertical condition either completely or partially decomposed the sequence of events into its constituent events and actions, while a majority of the teachers in the horizontal condition did not decompose or only partially decomposed the sequence of events. The dependence between condition and the extent of decomposition, however, was not statistically significant ($\chi = 4.08, p = .129$).

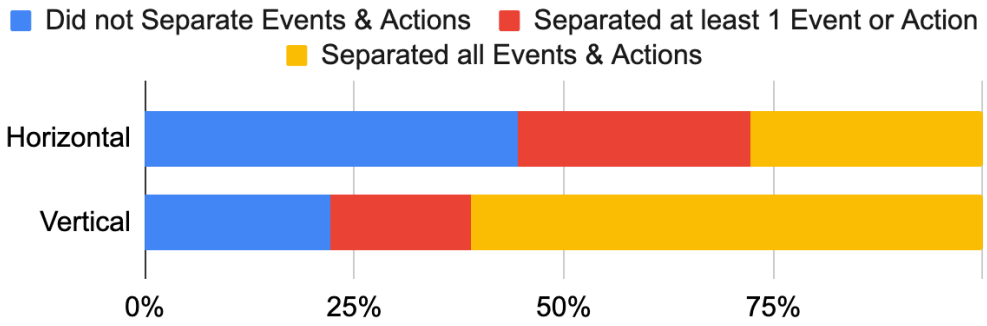


Figure 10.4: Diagram Worksheet Features

Taking a closer look at the worksheets, we found that more vertical teachers described one action in each action box (10 V vs 4 H) and one event in each event box (12 V vs 8 H). This demonstrates a complete decomposition of the sequence of events. In contrast, when filling out the action boxes, more teachers in the horizontal condition wrote at least one event or sprite, more than one sequential action, and actions that did not belong to the sprite in that row/column (Figure 10.5). This suggests partial or no decomposition of the

sequence of events. The mistakes that were more common among the vertical teachers were idiosyncratic responses in the action boxes, such as x-y coordinates or time, and writing at least one sprite in an event box. The latter mistake could be interpreted as incomplete events, as events in this module involve conditional sprite interactions.

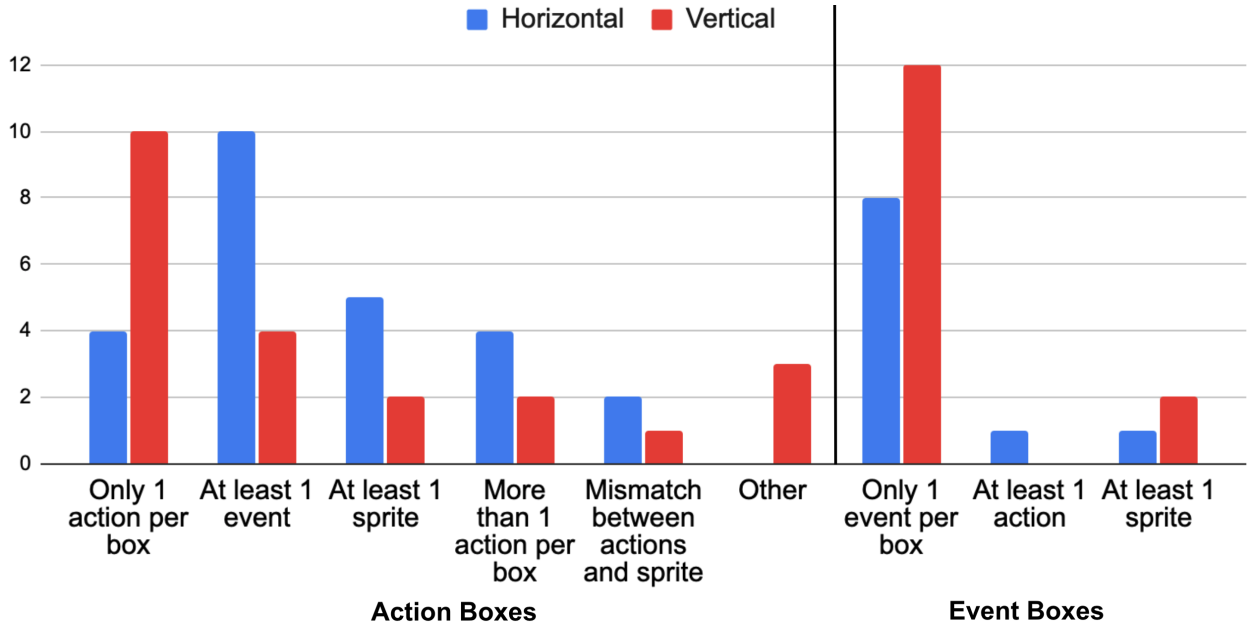


Figure 10.5: Diagram Worksheet Responses

Finding 2: Teachers in both conditions completed project requirements at similar rates. Figure 10.6 shows the completion rates of project requirements and extensions (adding third sprite with a new event and adding a sound block). Chi-square tests on proportions revealed no statistically-significant differences in the completion rates of any of the requirements or extensions. The only exception is the sound block extension, which vertical teachers were more likely to complete ($\chi = 3.91, p = .0479$). In addition to the stated project requirements and extensions, we also analyzed the usage of `broadcast/receive` blocks in their projects based on the information gathered during teacher interviews. These blocks are commonly used for message passing and synchronization in Scratch, and had not been introduced in this module or prior lessons. While the difference was not statistically significant, it is important to note that more horizontal teachers used these blocks to program their final project, instead

of the using the new `repeat until` or `wait until` blocks introduced in this module.

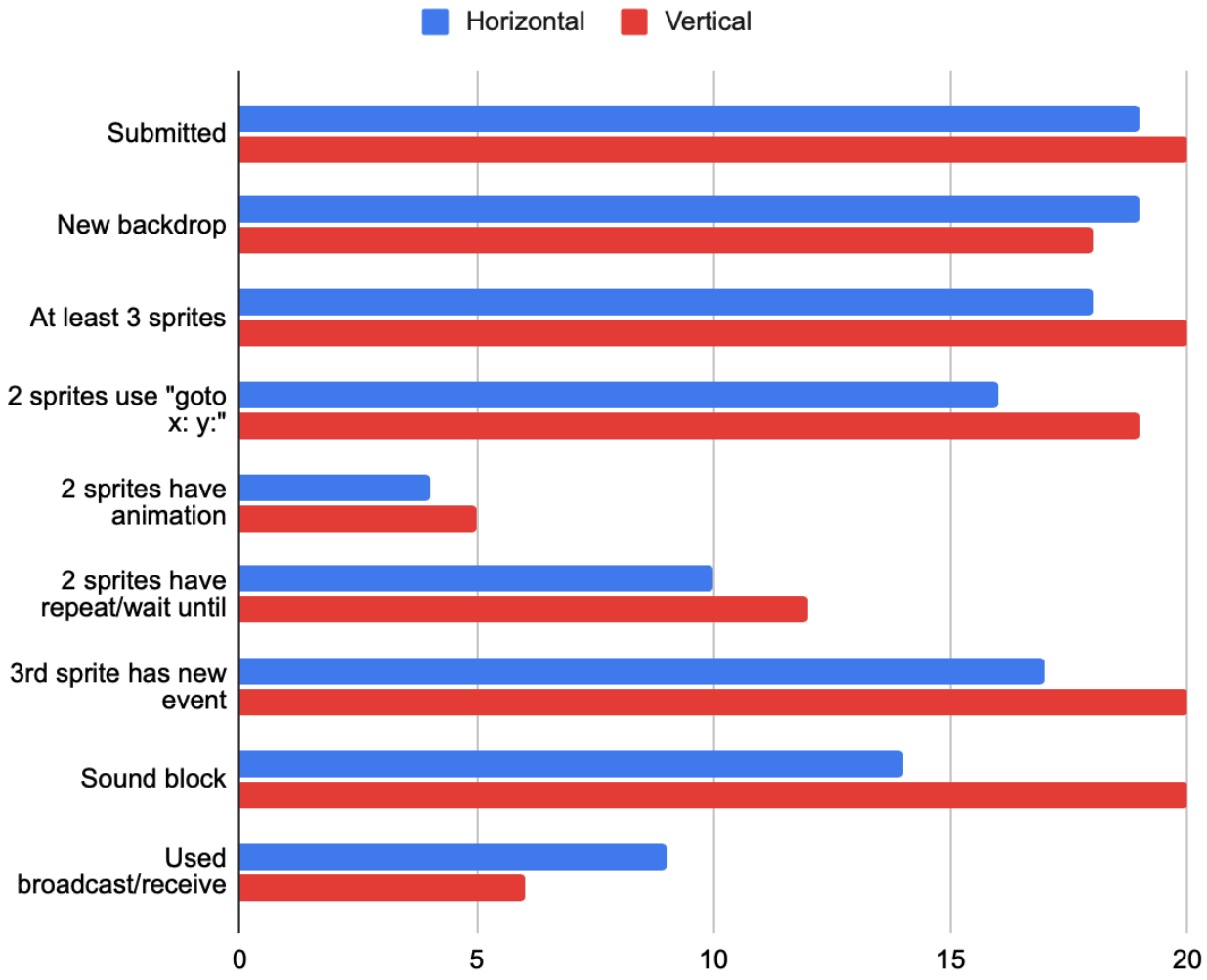


Figure 10.6: End-of-Module Project Features

Finding 3: When explaining their projects or diagrams, teachers in both conditions were similar in the CT concepts they described.

Figure 10.7 details the number of teachers in each condition who described each CT concept when prompted to explain their Scratch projects or diagrams. While teachers in both conditions were similar in the CT concepts they detailed, it is interesting to note that nearly twice as many teachers in the vertical condition brought up *decomposition*, the focal CT concept in this module.

Finding 4: Teachers in both conditions were similar in the different types of knowledge

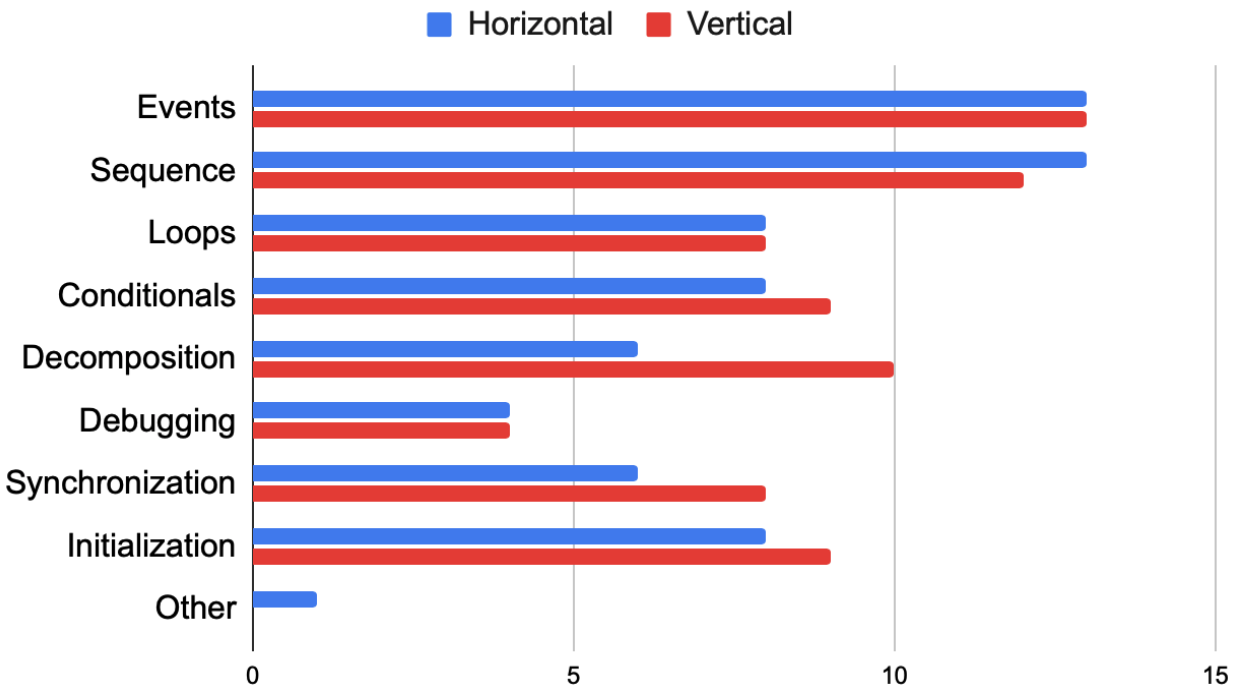


Figure 10.7: CT Concepts Described in Interviews

in the TPACK model they expressed in interviews.

Figure 10.8 illustrates the number of teachers in each condition who exhibited each aspect of TPACK. The two singular aspects of TPACK, technological and content knowledge, were omitted from the graph; the former because none of the teachers described technological knowledge alone and the latter because all of the teachers exhibited some level of content knowledge (Finding 3). As shown in Figure 10.8, a similar number of teachers in both conditions exhibited different aspects of TPACK.

The most commonly expressed types of knowledge were pedagogical knowledge (PK) and pedagogical content knowledge (PCK). A teacher in our study exhibited their PK when describing how they would pair struggling students with more advanced students for peer instruction, while another teacher displayed their PCK when drawing students' attention to different events in a thinkaloud explanation of their project. One teacher exhibited technology content knowledge, who explained that sprites in Scratch need a `goto x: y:` block to initialize in the correct starting position. Six teachers demonstrated technological pedagog-

ical knowledge, such as a teacher explaining XY coordinates before students learned it in math so they can move their sprites in Scratch. Lastly, five teachers even exhibited TPACK, demonstrating an intersection of technological, pedagogical, and content knowledge. For example, one teacher described making videos of them modeling their project, breaking down the different events and actions, so that their students could watch their explanations at their own pace.

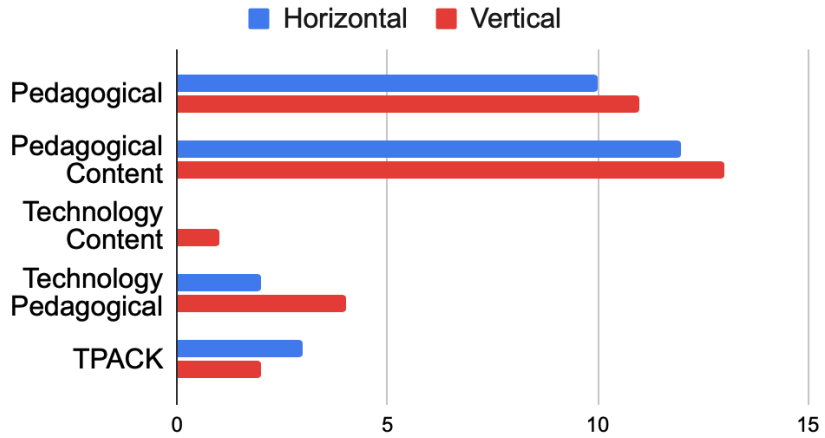


Figure 10.8: TPACK Aspects Described in Interviews

Finding 5: More teachers in the vertical condition described CT concepts at the relational level (11 V vs 4 H) of the SOLO taxonomy.

Figure 10.9 displays the number of teachers who demonstrated an understanding of CT concepts at each level of the SOLO taxonomy. The lowest level in the hierarchy, the prestructural level, was omitted as all teachers knew at least one relevant concept. A unistructural explanation consists of one relevant concept. Both multistructural and relational explanations are comprised of several concepts, but connections between concepts are made in relational explanations.

Most teachers in the horizontal condition explained CT concepts at the unistructural and multistructural levels, while the explanations of most teachers in the vertical condition were at the multistructural and relational levels. An example of a unistructural explanation would be a teacher in the horizontal condition who described the conditional interaction in their

project as "The first person is saying that she needed to get the blue potion[...]And then the elf lady says something else and then a little fairy drags over and tries to get her potion." While their project consisted of blocks corresponding to other concepts, they only described sequence. For a multistructural example, a teacher in the horizontal condition said, "When she touches the guitar, it would start playing[...]then it would send out a message to the target and it would repeat until the color red was touching that the brown of the boy's shoes". They described the concepts of events, sequence, loops, and synchronization, but the concepts were specific to their project and not integrated into a broader structure. In contrast, a relational example would be from a teacher in the vertical condition: "Sprites are characters that you can program to interact with each other in many different ways...when a sprite reaches that position and comes in contact with another sprite,[...]there's going to be an action that occurs when two sprites come in contact with each other." While this teacher described fewer concepts than the teacher with the multistructural explanation (only events and sequence), this teacher linked the concepts into a larger structure.

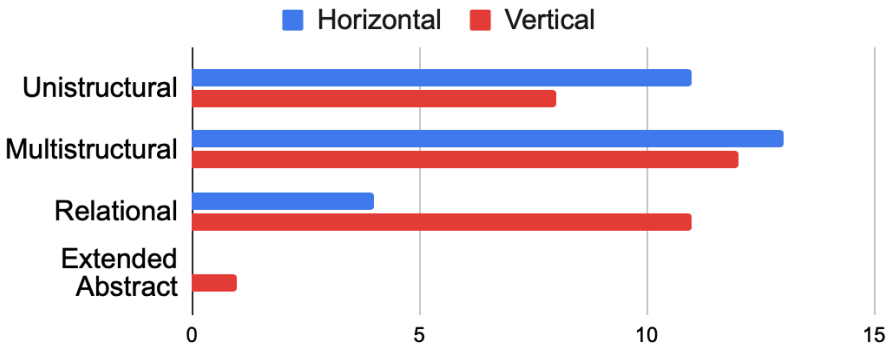


Figure 10.9: SOLO Taxonomy Levels of Interview Responses

10.2.3 Discussion

For RQ1, our analysis of diagram worksheets revealed that most teachers using the vertical diagrams either partially or completely decomposed the sequence of events into their composite sprite actions and events. In the horizontal condition, most teachers either partially

or did not decompose the sequence of events. While teachers in each condition differed in the degree of decomposition in their diagrams, they performed similarly in their Scratch projects, completing requirements at similar rates. This result suggests that, regardless of condition, teachers were able to produce code artifacts at the highest 'Producing' level of the Matrix Taxonomy, *Create* [80]. However, prior work has shown that students frequently create with code that they do not fully understand [25, 205]. A similar phenomenon may have occurred with the teachers in the horizontal condition, given their responses to the diagram worksheets and their unistructural and multistructural descriptions of CT concepts in interviews based on the SOLO taxonomy.

For RQ2, we found that teachers in both conditions developed some level of content knowledge, one of the pillars of TPACK. They described CT concepts with similar frequency when asked to explain their projects or diagrams. While we could not run statistics on the CT concepts described, it is important to note that nearly double the number of teachers in the vertical condition mentioned *decomposition*, the key concept in this module. In addition to content knowledge, teachers in both conditions also exhibited other aspects of TPACK with similar frequency, with the two most common aspects of TPACK being pedagogical and pedagogical content knowledge. Nonetheless, most teachers in the vertical condition demonstrated deeper knowledge of the CT concepts, either describing multiple concepts (multistructural) or drawing connections between several concepts (relational). In contrast, most teachers in the horizontal condition described concepts in isolation (unistructural or multistructural).

While our small sample size limited the applicability of quantitative statistics, our results, when combined with qualitative analysis, provided insights into the design of diagrams for the learning of computational thinking concepts, such as decomposition and conditionals. The reason for better performance with vertical diagrams may be due to its alignment with the way programs are read, the fact that text is read vertically for high-level structure and

horizontally for detail, or something else entirely. Further research would be needed to uncover the underlying reasons; we hope that this study will lead to further research into the use of diagrams in K-8 computing.

10.3 To Diagram or not to Diagram: Comparing Text-Based and Diagram-Based Scaffolds for Learning Decomposition

With our insights from the teacher phase, we next wanted to investigate how a diagram-based scaffold for planning the creation of a project with conditional interactions between sprites would compare with a text-based scaffold. More specifically, we are motivated by the following research questions:

a) How do students respond to text-based and diagram-based scaffolds?

b) What is the relationship between the two different types of scaffolds and student performance on coding projects and assessments?

For this study, I was in charge of the development of the strategy and associated curricular materials, study design, assessment design, data collection and data analysis. I also helped with the classroom recruitment, which was led by our lab's School Development Specialist Donna Eatinger. I observed and supported instruction in all classrooms in this study. A postdoc in my lab, Dr. Jennifer Tsan, was my second coder for any qualitative analyses.

10.3.1 Methods

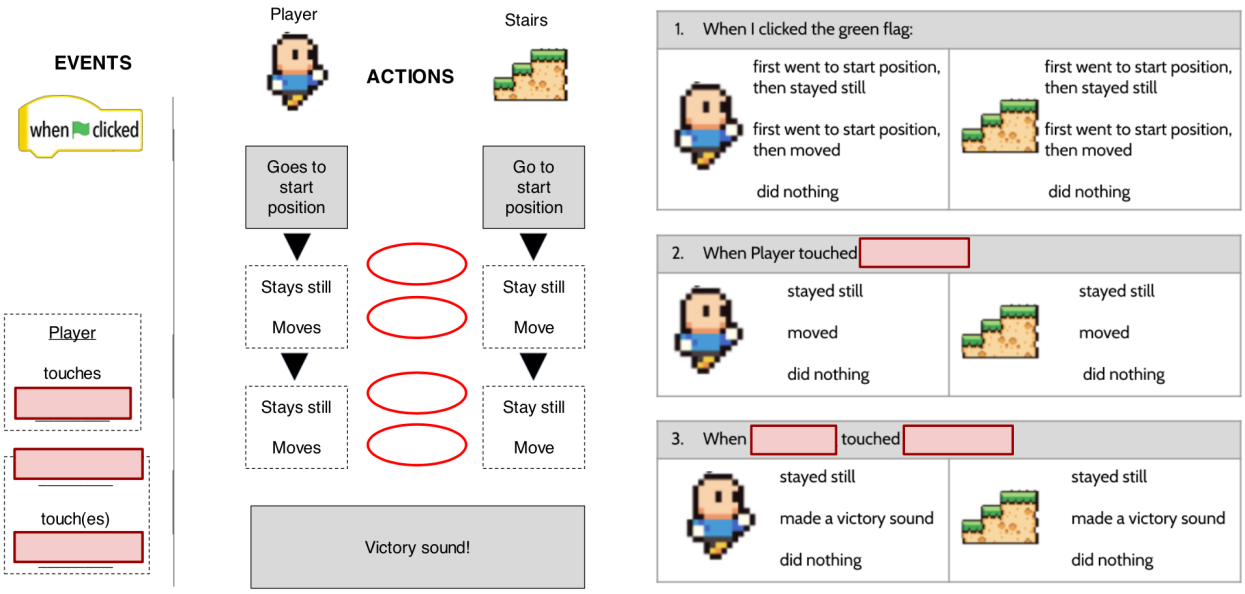
Study Context

Five teachers were recruited from Chicago Public Schools and underwent the same professional development, either in person or virtually, to teach Scratch Encore (Chapter 4). A total of nine classrooms were participated in the study. Classrooms were assigned to either

the Diagram or No Diagram condition for the *Decomposition by Sequence* module based on grade level and class size. This resulted in 2 sixth grade, 1 seventh grade, and 1 eighth grade classrooms for a total of 4 classrooms in the No Diagram condition, and 1 fifth grade, 1 sixth grade, 2 seventh grade, and 1 eighth grade classroom for a total of five classrooms in the Diagram. There were 68 students in the No Diagram condition and 69 students in the Diagram condition. Students in the Diagram condition were given the vertical version of the diagram because it showed more success in our teacher study. While our results could not reach statistical significance because of the small number of participants, teachers using the vertical diagram were better able to decompose a sequence of events and make connections between CT concepts.

In "Use → Modify" phase of this module, students in both conditions first watched the same video that showed a finished Scratch project with a sequence of events. In the Scratch project, when the green flag was clicked, a Player sprite walked towards a Stairs sprite. When the Player kicked the Stairs, the Stairs moved towards a Cliff sprite and once the Stairs touched the Cliff, a victory sound was played. Based on the video, students in the Diagram condition filled an incomplete vertical diagram that decomposes the sequence of events, while students in the No Diagram condition answered multiple-choice questions analogous to the blanks in the diagram. After completing their observations of the complete sequence of events, they were instructed to explore an unfinished version of the demonstrated Scratch project using the TIPP&SEE learning strategy. Students were next directed to modify the project so that it worked like the Scratch project in the video.

In the Create phase, students in both conditions were given a text-based plan in their worksheet, where they could fill in the blanks for the sprites, events, and actions in their final projects (Figure 10.11). The text-based plan underwent a similar design process as the diagrams. The text-based plan was designed over the same three-month time period as the diagrams in weekly meetings, with iterative refinements based on feedback from researchers



(a) Diagram Condition

(b) No Diagram Condition

Figure 10.10: Use→Modify Activity

and practitioners. They were also given to Scratch Encore lead teachers for a final round of feedback and revisions. In the Diagram condition, students were given the vertical diagram (Figure 10.2a) described in the teacher study in addition to the text-based plan.

Data Analysis

We used a mixed methods approach to analyze student worksheets, culminating "Create" Scratch projects, and assessments. The number of students who attempted each form of work for each condition is shown Table 10.2.

Student Worksheets

We analyzed students' plans, both text-based and diagram-based, in the "Create" worksheet for evidence of decomposition, where they planned their own final project. For the diagram-based plans, two researchers developed a coding manual of categories that described the extent to which they decomposed the sequence of events — separated all the events and actions (complete decomposition), separated at least one event or action (partial decomposi-

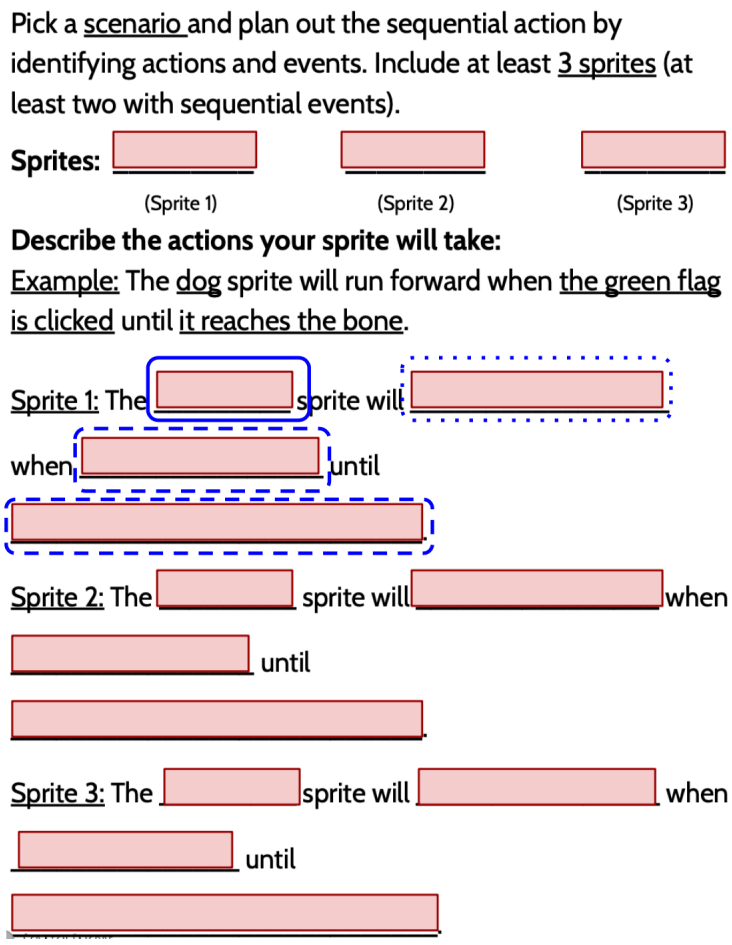


Figure 10.11: Text-Based Plan

	Diagram	No Diagram
Number of Students	69	68
Text-Based Plan	18	59
Diagram Plan	14	—
Scratch Project	55	62
Assessment	39	63

Table 10.2: Number of Students

tion), or did not separate events and actions (no decomposition), similar to the prior teacher study.

For both types of plans, the manual also classified characteristics of responses in the "event" and "action" boxes in the diagram. The classification process for the diagram-based plan was the same as in the teacher study, and was adapted for the text-based plan. The key difference is that the "event" and "action" boxes were inline as blanks to fill in as part of a larger sentence. Figure 10.11 depicts the text-based plan; the "sprite" box is highlighted in the solid line, the "action" box is highlighted in the dotted line, and the "event" boxes are highlighted in the dashed line.

Two researchers coded each worksheet separately and then resolved any disagreements through discussion for 100% reliability. To see if there was a dependence between condition and the extent of decomposition or worksheet features, we used a Chi-square test of independence; we report χ and p values.

Scratch Projects

Features of students' final "Create" Scratch projects were automatically scraped to determine if they completed the assignment requirements. There were five requirements: adding a new backdrop, using at least 3 sprites, using the `go to x: y:` block in at least 2 sprites (for initialization), animating at least 2 sprites, and using a `repeat until` or `wait until`

`blocks` to program a sequence of events. There were also two extensions that students could complete: adding a new event to a third sprite and using a sound block. To see if there was a statistically-significant difference between the two conditions' completion rates, we conducted the Chi-squared test on proportions, from which we provide χ and p values.

Assessments

To see if condition was associated with performance on assessment questions, the nonparametric Kruskal-Wallis test was used; this test yields a χ^2 value and a p value. We use $p < .05$ as our threshold for significance. A nonparametric test was chosen because of small and unbalanced sample sizes for some assessment questions. Details on assessment design and validation are in Chapter 4.

10.3.2 COVID-19 Limitations

This study was conducted during the 2020-21 school year amidst the COVID-19 pandemic. All instruction in Chicago Public Schools was virtual from September 2020 to February 2021, and instruction for elementary and middle schools was hybrid from March to June 2021. In the No Diagram condition, all of the classrooms were taught entirely online. In the Diagram condition, the eighth-grade classroom was taught virtually, while the rest of the classrooms were taught hybrid. By the time teachers and students encountered this module, they would have spent three to nine months in virtual instruction. In contrast, when students and teachers encountered it in hybrid, they had only spent a third of the time (one to three months) in hybrid instruction. Both formats presented their unique instructional challenges, but hybrid instruction introduced a different set of challenges. From classroom observations, hybrid instruction was especially challenging for teachers, having to manage students both in person and online. Difficulties from emergency remote and hybrid teaching are likely to have affected the quality of instruction, so the results outlined in the following section may be different had the experiment been done during pre-pandemic, in-person instruction.

As such, we consider this study to be a pilot study with a run through all methods and analysis. However, the results themselves are not completely actionable due to these adverse conditions.

10.3.3 Results

Finding 1: Responses on the text-based plan from students in both conditions were similar. Figure 10.12 depicts the percentage of students who attempted the text-based plan with each response type. The first three response types, "Sprite in Sprite Box", "Action in Action Box", and "Event in Event Box", are correct responses, while the last three response types are incorrect responses. We did not find a statistically-significant dependence between condition and any of the response types classified in the text-based plan (Sprite in Sprite box: $\chi^2 = 2.10 \times 10^{-30}, p = 1$, Action in Action box: $\chi^2 = 1.38 \times 10^{-}30, p = 1$, Event in Event box: $\chi^2 = .107, p = .744$, Action in Event box: $\chi^2 = .726, p = .394$, Sprite in Event box: $\chi^2 = 1.23, p = .266$, Something else in Event box: $\chi^2 = .674, p = .412$).

Finding 2: Students in the Diagram condition responded differently in the event boxes in the text-based and diagram plans.

Since students in the Diagram condition were exposed to both the text-based and diagram-based plans, we also compared their responses in each type of scaffold. Figure 10.13 portrays the percentage of students in the Diagram condition who provided each response characteristic. We found statistically-significant dependencies between the type of scaffold and where students wrote events (Event in Action box: $\chi^2 = 5.15, p = .023$, Event in Event box: $\chi^2 = 6.09, p = .014$) and what was written in event boxes (Something else in Event box: $\chi^2 = 4.89, p = .0271$). Students were more likely to write events in action boxes when using the diagram-based plan, compared with the text-based plan. They were more likely to correctly write events in the event boxes when using the text-based plan. However, they were also more likely to write responses that were neither a sprite, action, nor event,

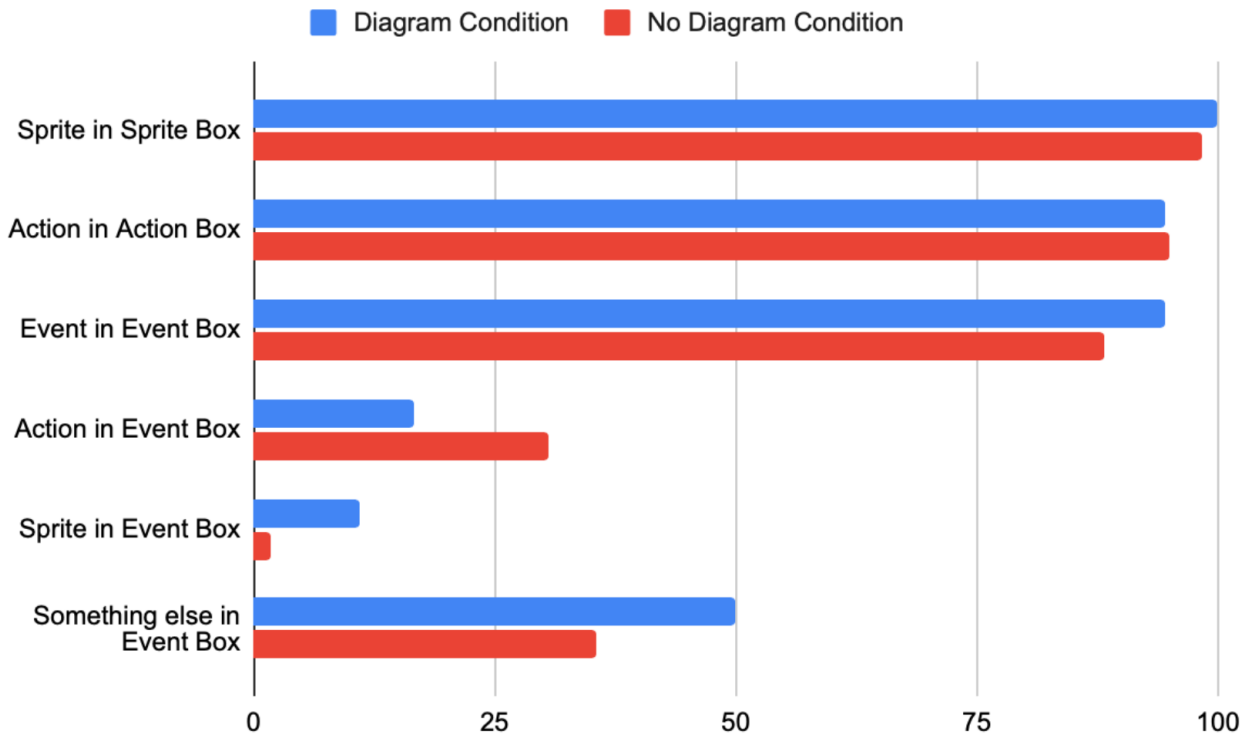


Figure 10.12: Responses on Text-based Plan across Conditions

such as "the code is over" or "the sprite is done talking" in the text-based plan. There were no statistically-significant dependencies between the type of plan and responses in the action and sprite boxes or where actions and sprites were written (Sprite in Sprite box: $\chi^2 = .847, p = .358$, Action in Action box: $\chi^2 = 2.07 \times 10^{-31}, p = 1$, Sequential actions in an Action box: $\chi^2 = 3.56, p = .059$, Action in Event box: $\chi^2 = .0726, p = .788$, Sprite in Event box: $\chi^2 = .0941, p = .759$)

Finding 3: Students in both conditions completed most Scratch project requirements at the same rate.

Figure 10.14 shows the percentage of students in both conditions who completed each project requirement. There was only a statistically-significant dependency between conditions and whether students added a new backdrop ($\chi^2 = 5.06, p = .024$). For the rest of the requirements, there were no dependencies with condition (At least 3 sprites: $\chi^2 = 2.95, p = .0856$, 2 sprites use "goto x: y:": $\chi^2 = 2.18, p = .139$, 2 sprites have animation: $\chi^2 = 1.81, p = .179$,

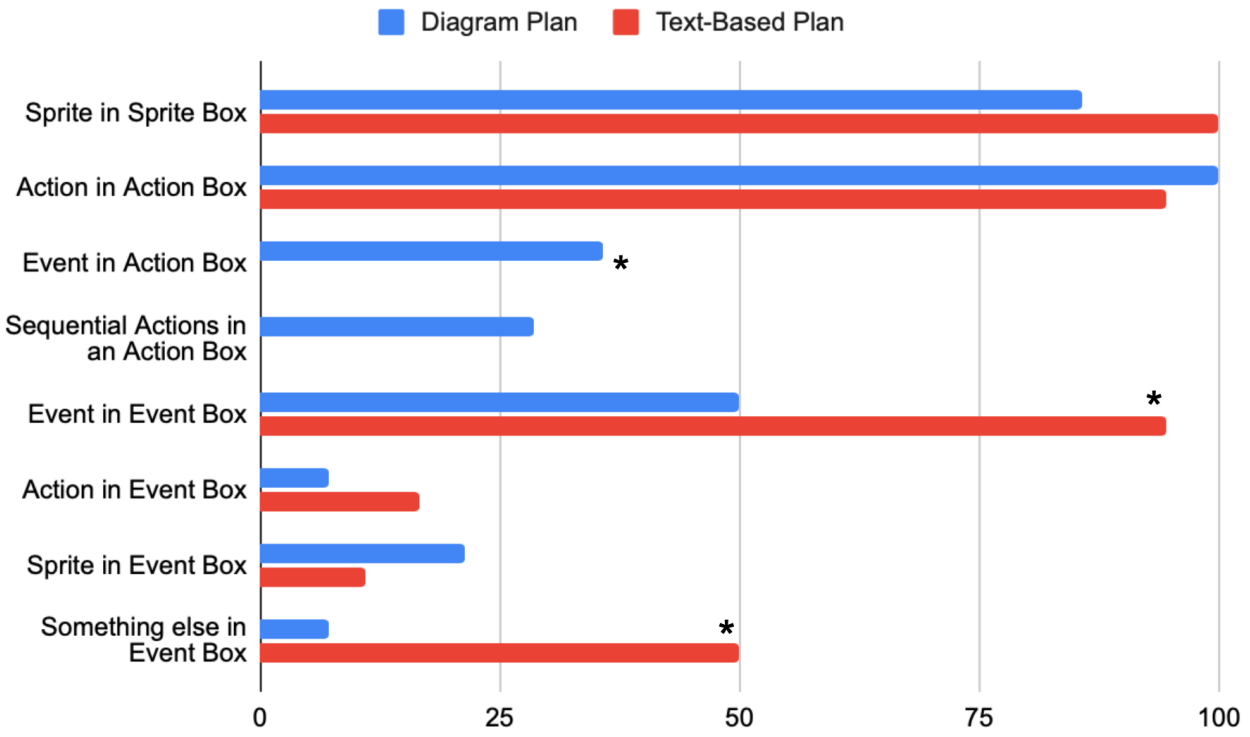


Figure 10.13: Responses of Students in Diagram Condition on Both Plans

2 sprites have `repeat until` or `wait until`: $\chi^2 = 4.17 \times 10^{-31}, p = 1$, 3rd sprite has a new event: $\chi^2 = 2.95, p = .0856$, and sound block: $\chi^2 = 9.04 \times 10^{-31}, p = 1$)

Finding 4: Students in both conditions performed similarly on assessment questions. There were only two questions where there were statistically-significant performance differences between the two conditions. In one question, students in the Diagram condition performed better while in the other question, students in the No Diagram condition performed better.

The first question with performance differences asked students to identify examples of an "action" in coding from two actions programmable in sprites, playing a sound for 5 seconds and moving 10 steps, and two user actions, clicking a sprite and pressing the spacebar (Appendix Figure A.2). The correct answers were playing a sound for 5 seconds and moving 10 steps. 1 point was awarded for each correct action identified and 1 point was subtracted for each incorrect action, for a maximum of 2 points. In this question, students in the Diagram

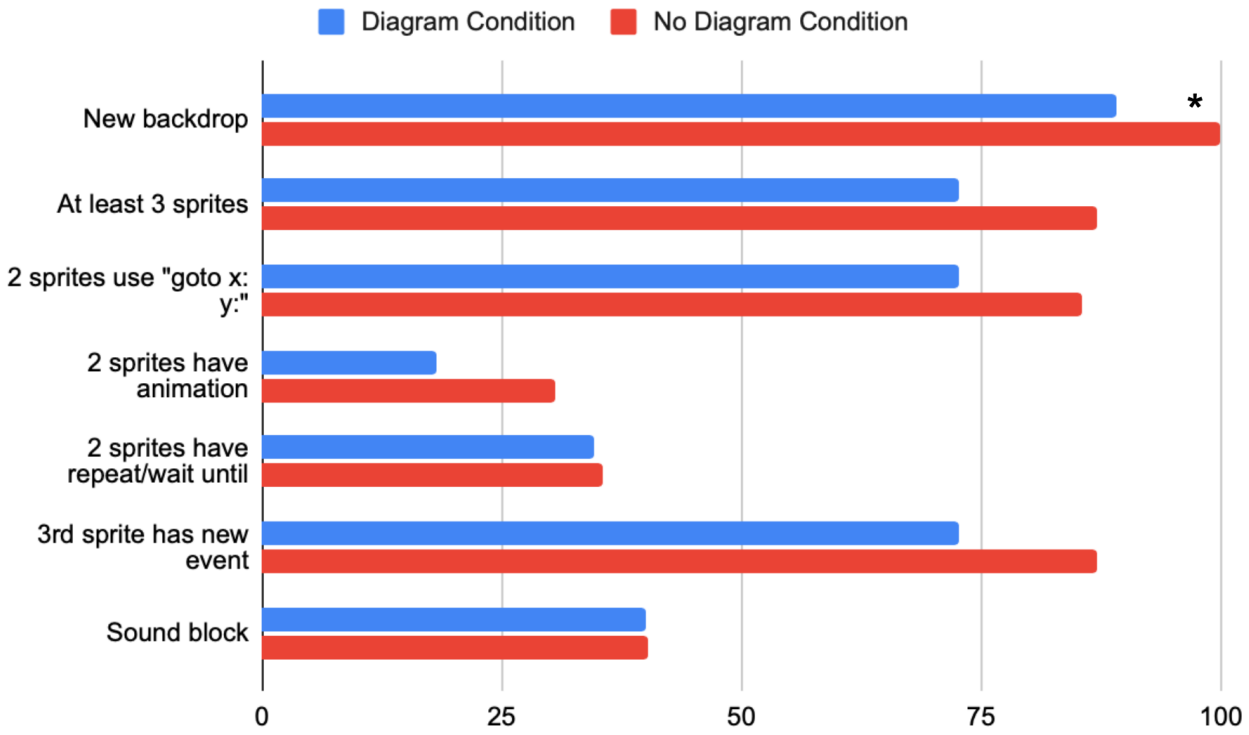
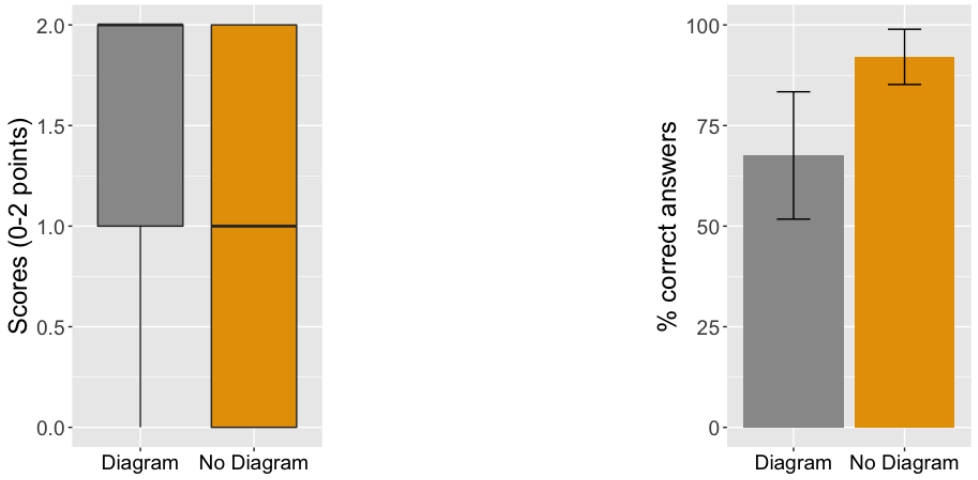


Figure 10.14: Requirement Completion Rates of Students in Both Conditions

condition outperformed students in the No Diagram condition ($\chi^2 = 8.37, p = .00382$). Figure 10.15a portrays the score distribution of this question for students in both conditions.

In the other question, students were shown two scripts that belonged to a Needle sprite and Balloon sprites, respectively. In this Scratch program, when the Green Flag was clicked, the Needle sprite would start moving towards the Balloon sprite and when the Needle sprite touched the Balloon sprite, the Balloon sprite would pop (see Appendix Figure A.8). Based on the scripts, students were asked to identify the event that caused the Needle to start moving and received 1 point if identified correctly. In contrast to the previous question, students in the No Diagram condition performed better than students in the Diagram condition ($\chi^2 = 9.81, p = .00173$). Figure 10.15b depicts the percentage of students in each condition who answered this question correctly.

For the rest of questions, there were no statistically-significant differences between students in both conditions. Since an exploratory factor analysis (see Chapter 4) revealed



(a) Q: Examples of Actions in Coding (b) Q: Event that caused Needle to start moving

Figure 10.15: Questions with Performance Differences between Diagram & No Diagram Conditions

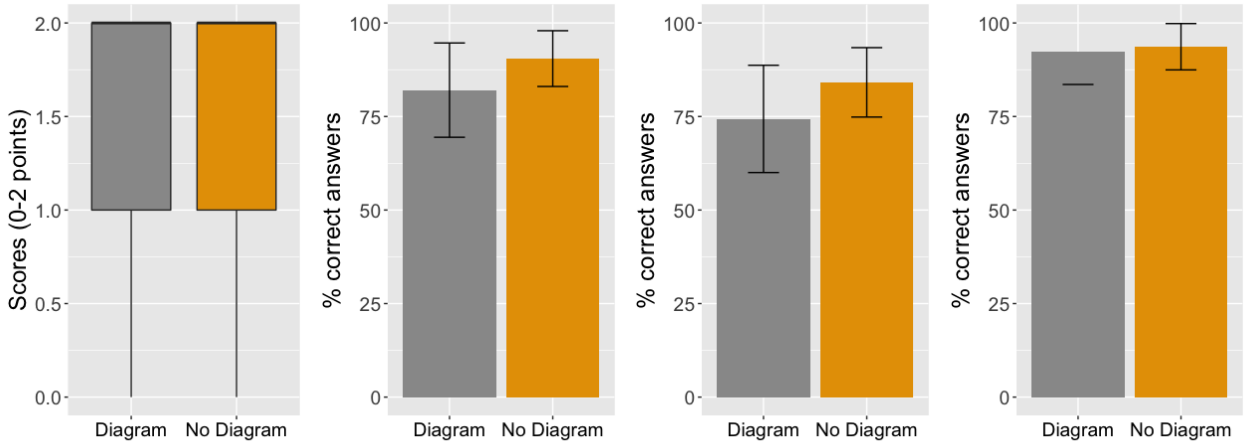
that the underlying structure of the questions fell along the levels of Bloom's taxonomy, the rest of this section will be organized based on the two levels targeted in this assessment, *Remember* and *Understand*.

Questions at the "Remember" Level

The first "Remember" question asked students to identify examples of "events" in programming from a list of two Scratch and custom events: when the user clicks the green flag and when a sprite touches another sprite, and two real-life events: when you play basketball every week and when you go to a dance concert (Appendix Figure A.1). 1 point was awarded for each correct response and 1 point was deducted for each incorrect response, for a maximum of 2 points. There was no statistically-significant difference in performance between the two conditions on this question (Figure 10.16a; $\chi^2 = .0724, p = .788$).

The rest of the "Remember" questions asked students to select the response that best described what an end condition was (Appendix Figure A.3), when to use a `Repeat Until` block (Appendix Figure A.5), and when to use a `Wait Until` block (Appendix Figure A.6). Students received 1 point for a correct response on all three questions. Performance

differences between the two conditions in these questions were not statistically significant (Figure 10.16b; $\chi^2 = 1.52, p = .217$, Figure 10.16c; $\chi^2 = 1.44, p = .229$, Figure 10.16d; $\chi^2 = .0673, p = .795$).



(a) Q: Examples of Events in Coding

(b) Q: What is an End Condition

(c) Q: When to use a Repeat Until Block

(d) Q: When to use a Wait Until Block

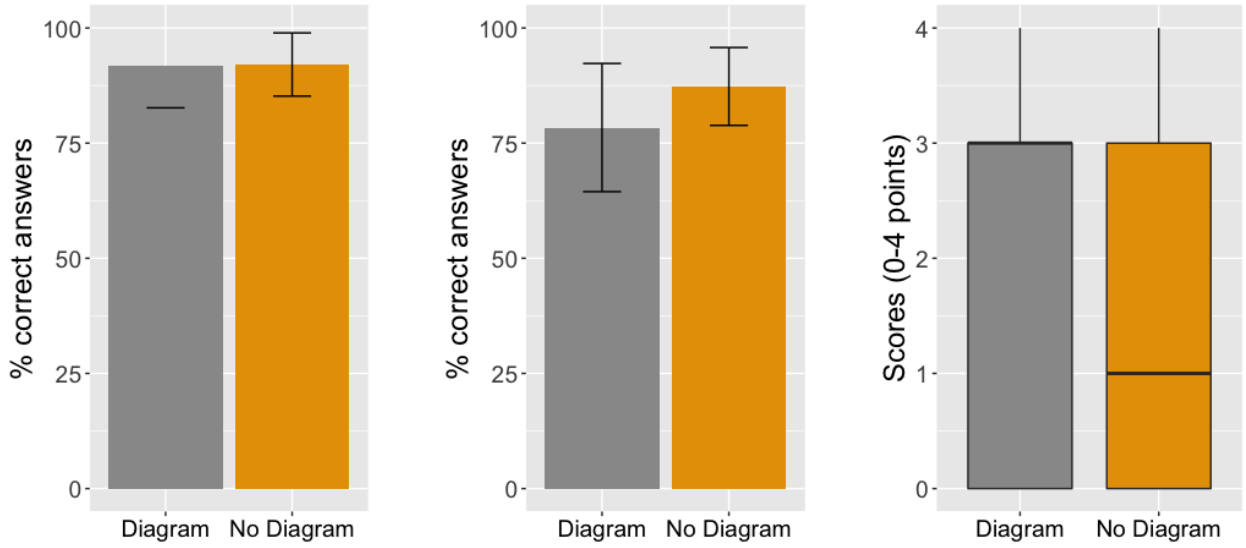
Figure 10.16: "Remember" Questions with No Performance Differences

Questions at the "Understand" Level

The first "Understand" question asked students to identify the event that caused a Balloon sprite to pop based on scripts for a Needle and Balloon sprite (Appendix Figure A.9). Responses to this question were simply marked as correct or incorrect. The percentage of students who answered correctly in both conditions were not statistically-significantly different (Figure 10.17a; $\chi^2 = .000923, p = .976$).

The last two "Understand" questions were based on a video of a Scratch project where a golf club hits a golf ball into a hole. One of the questions asked students to choose the best decomposition of the sequence of events they observed in the video (Appendix Figure A.4). The correct response decomposed the sequence of events into its composite events and actions; students were awarded 1 point for answering this question correctly. The difference between the percentage of students who answered this question correctly was not statistically significant ($\chi^2 = 1.37, p = .242$). The other question asked students to order code snippets

to assemble the script for the golf ball (Appendix Figure A.7). There were 4 code snippets necessary to assemble the script. Responses were awarded 1 point for each correct code snippet in the correct position. 1 point was deducted for each incorrect code snippet or incorrect position. There was no statistically-significant performance differences between conditions on this question ($\chi^2 = 3.16, p = .0754$).



(a) Q: Event that caused Balloon to pop

(b) Q: Best Decomposition for Programming

(c) Q: Drag & Order Blocks for the Golf Ball Script

Figure 10.17: "Understand" Questions with No Performance Differences

10.3.4 Discussion & Implications

For RQ1, we found that students exposed to only a text-based scaffold and students exposed to both a text-based and diagram-based scaffold responded similarly to text-based scaffold, with more than 75% of students correctly filling in the sprite, action, and event boxes. However, comparing the two different types of scaffold for the students in the Diagram condition revealed that the main differences in student responses to text-based and diagram-based scaffolds stemmed from their understanding of events in this module. While students were more prone to write events in the boxes designated for events in the text-based diagram, they

were also more prone to writing responses that were neither sprites, actions, nor events. This may be indicative of misconceptions around events, especially since the definition of events has expanded beyond pre-programmed Scratch events in this module to include situations where a condition starts or stops an action of a sprite.

For RQ2, we only found one difference in requirement completion rates in student Scratch projects. The sole difference was in a higher completion rate in adding a new backdrop from the students in the No Diagram condition. Even then, this requirement was more aesthetic, rather than a functional or structural change to the code. Students in both conditions were comparable in demonstrating their ability to "Create", the highest level of the *Producing* dimension of the Matrix taxonomy. Results from assessments were similar. There were only two questions where there were performance differences between the two conditions. Even so, students in the Diagram condition performed better in one question, while students in the No Diagram condition performed better in the other question. Not only were students similar in the *Producing* dimension, they were also similar in their ability to understand and interpret existing code, the *Interpreting* dimension of Matrix taxonomy.

Since prior work in learning strategies has shown a combination led to performance improvements in almost all cases [59], it was unexpected that a combination of both text-based and diagram-based scaffolds led to similar outcomes as only using a text-based scaffold. The most direct explanation would be that four of the five classrooms in the Diagram condition were instructed in a hybrid setting, while all four classrooms in the No Diagram condition were instructed remotely. Hybrid instruction was more challenging than remote instruction for teachers because they had to manage students both in-person and online. Frequent disruptions due to hybrid classroom management issues likely had an adverse effect on instruction. It need not have been in-person instruction either; having all classrooms on virtual instruction could also have an impact on results.

It may also have been the case that students in the Diagram condition may have had

difficulty knowing which plan to use because they were given two different types of plans. Prior work has shown that older students struggled with planning, strategy selection, and self-regulation, and that planning and self-regulation vary widely between students [119, 141]. It would not be surprising that students in our study, who are younger than the students in previous studies, had the same difficulties and such difficulties may have been exacerbated by virtual or hybrid instruction, which made it more difficult for students to ask for help and for teachers to identify if students need help. Future work could include thinkalouds or observations of students as they create their projects to better understand their thought processes.

Another possible explanation could be that the diagram used in this study may not have been the best match for students' mental model of the sequence of events they observed in the *Decomposition by Sequence* module. Prior work [201] has suggested that not all students, especially at the K-6 level, can readily reason about diagrams, timelines, or other graphics designed by others, and that graphical literacy is a skill that should be fostered in students. Further, students as young as sixth-grade have been shown to have profound meta-representational competence [94], which can be leveraged in the design of similar graphical representations of computational thinking ideas. A future direction for this work would be to have students think aloud to decompose their code, perhaps even designing their own representations in the process; such a study was not possible in the 2020-21 school year because of pandemic-induced restrictions on research. Identifying whether our results were due to hybrid instruction challenges, a mismatch between the diagrams and students' mental models, or something else entirely would require a replication of this study in a less turbulent school year with more consistent modes of instruction.

CHAPTER 11

CONCLUSION, LIMITATIONS, AND FUTURE WORK

In this chapter, I revisit the research questions underlying the studies in this dissertation, summarize the contributions of those studies, outline their limitations, and discuss some avenues for future work.

11.1 Research Contributions

Across all my studies, I was motivated by the following overarching research questions:

- Which factors, from academic skills to demographics, are associated with program comprehension in a formal school setting?
- What kinds of comprehension do young learners demonstrate after open-ended, exploratory instruction?
- How can we support the development of program comprehension in young learners?

For the first research question, we have identified social factors, such as school performance level, gender, and race/ethnicity, and academic factors, such reading, math, and cognitive skills, to be associated with program comprehension. We are one of the few researchers to have identified these factors in a formal school setting. For reasons outlined in Chapter 2, such as socioeconomic barriers, opportunity and awareness gaps, and self-selection, a formal school setting allows for a more representative sample of students compared with the informal learning environments explored in prior work. By discovering the factors that apply to a more representative sample of students, we have a better understanding of the skills crucial to developing different computational literacies and therefore, we can better draw from other discipline-based education research fields that encompass these skills to improve computing instruction for elementary-age learners.

For the second research question, we have found that students frequently use code that they do not fully understand and that they show a strong functional understanding of their code, but struggle with structural details. Students understand *what* their code does, but not necessarily *how* their code accomplishes it. While similar phenomena have been identified in older students [166, 254], we are one of the few to have identified this disconnect in young learners. With this discovery, we can delve deeper to better understand why such disconnect occurs, as well as design interventions, curricula, and other instructional supports that help bridge this gap and develop both a functional and structural understanding of code in young learners.

For the last research question, we have developed and studied two learning strategies for this age group to support their development of program comprehension, TIPP&SEE and diagramming. TIPP&SEE, a mnemonic to remind students of the steps in exploring a new Scratch project, has been linked to better student performance in code comprehension questions and in code quality, as well as narrowed gaps between students with and without academic challenges in code comprehension questions. As for a diagram designed to scaffold the learning of decomposition, we found that teachers using a vertical orientation of the diagram were better able to decompose a sequence of events and make connections between computing concepts. However, we also found similar performance between students who did and did not use a diagram to scaffold their learning of decomposition. With those explorations of learning strategies for elementary computing, we have set an example for how strategies from other discipline-based education research fields can be adapted in computing, provided insights for what scaffolds do and do not work for computing, and opened the door for further exploration of learning strategies for this age group.

11.2 Limitations

There are several limitations to the studies in this dissertation. First, all of the studies were conducted in one of three school districts. School districts can differ based on many factors that can influence results, such as student and teacher populations, computing resources, and class time. Variability across school districts would make generalization difficult. We would need randomized controlled trials in a variety of different school districts and large-scale replications to see if results apply more broadly beyond a specific school district.

Another limitation stems from the use of quantitative measures for different student skills. The measures used within this dissertation, such as those used to measure proficiency in reading and math and cognitive skills, are outcomes-oriented and only one part of a larger picture of a students' ability. More process-oriented measures, such as thinkalouds and interviews, would be complementary to understanding students' abilities and skills.

Lastly, while teachers have been shown to have a strong influence on student work [74], we could not account for teacher variability in our studies due to resource constraints. Teachers play an important role in implementing the interventions in our studies. Teachers' confidence in the content knowledge, their use of the curriculum and scaffolds, their classroom management, their choice in lesson structure and the time spent on each activity, and a variety of other factors could impact our results. We would need studies on teacher fidelity, self-efficacy, adaptations to the curriculum, and other teacher factors to better understand their role in these interventions.

11.3 Future Work

The insights from this dissertation present interesting directions for future work. First, the identification of a gap between functional and structural understanding of code [215] in elementary school students prompts further questions: *How does existing computing in-*

struction lead to the disconnect between a functional and structural understanding of code? How can we improve existing instruction to alleviate this disconnect? How can we design instruction that promotes both structural and functional understandings of code effective and engaging for learners in this age group? Other researchers [50, 166] have begun exploring interventions to develop a structural understanding of code, but for university-age students. Understanding and bridging the gap between functional and structural understandings of code offer an important and fascinating area for future study.

Our work on learning strategies also opens up further investigation on the role of strategies in programming instructions. Our informal observations of the use of TIPP&SEE in the classroom suggest that the value of TIPP&SEE may lie in scaffolding meta-cognition, or proceduralizing how students explore a new Scratch project, rather than supporting self-regulation, or monitoring progress towards a goal. We observed students following the TIPP&SEE steps when exploring a new Scratch project but were frequently distracted in the process, needing a teacher to keep them on track. The teachers were regulating student progress, not TIPP&SEE or the students themselves, which is not surprising given that prior work has shown that even university students struggle with self-regulation [141]. The role of teachers as the "regulators" of a strategy was not only for the benefit of the students, but the teachers themselves. Observations also indicated that TIPP&SEE was integrated as part of classroom management; for example, some teachers used TIPP&SEE worksheets as prerequisite activities before students were allowed work on their own Scratch projects. Future work can include investigating scaffolds for self-regulation in program learning for elementary-age students and the roles and perspectives of teachers in the enactment of learning strategies.

As for diagramming, results from student study highlight a feature unique to computing not present in other subjects: interactivity in programs and systems. We drew from timelines in other subjects as inspiration for the diagrams used in this module, but in those subjects, students are frequently breaking down static, not dynamic, processes such as histor-

ical events or plot points in a story. Some researchers have explored programming scaffolds that account for such interactivity, such as coding strips or comic strips associated with pieces of code [226], with early promise and engagement from students. Visual scaffolds that capture the interactive and dynamic nature of programs, such as storyboards, merit further investigation.

With the large body of work on learning strategies both from studies in this dissertation and prior work (Chapter 2), interesting follow-up research questions would be: *What makes a learning strategy effective and why? What features do successful learning strategies share?* For example, TIPP&SEE scaffolds meta-cognition more than it does self-regulation, but to what extent does that apply to other learning strategies? Future research can include synthesizing prior work in learning strategies and developing a framework or meta-design of successful learning strategies in programming.

APPENDIX A

DECOMPOSITION BY SEQUENCE ASSESSMENT QUESTIONS

We present the questions in the Scratch Encore Decomposition by Sequence end-of-module summative assessments, organized by learning goal. Within each learning goal, we identify which level of the Bloom's taxonomy the question is targeting.

A.1 Learning Goal 1: Decompose a Sequence of Events

The first learning goal of this module is for students to decompose a sequence of events. There are four questions targeting this learning goal, three at the "Remember" level and one at the "Understand" level. The three "Remember" questions ask students to recall definitions for events (Figure A.1), actions (Figure A.2), and end conditions (Figure A.3) introduced in the curriculum. For the "Understand" question, students were shown a video of a golf club hitting a ball into a hole and asked to decompose the sequence of events in a way that would be suitable for programming (Figure A.4).

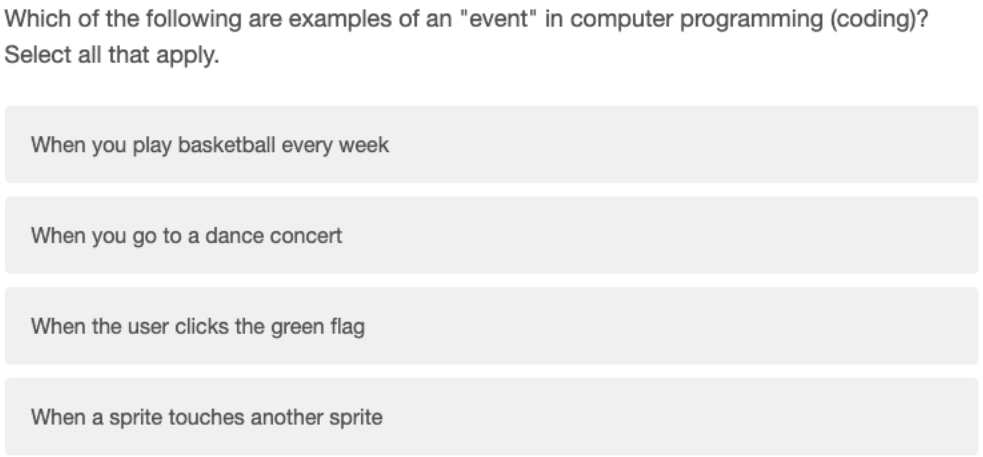


Figure A.1: Event "Remember" Question

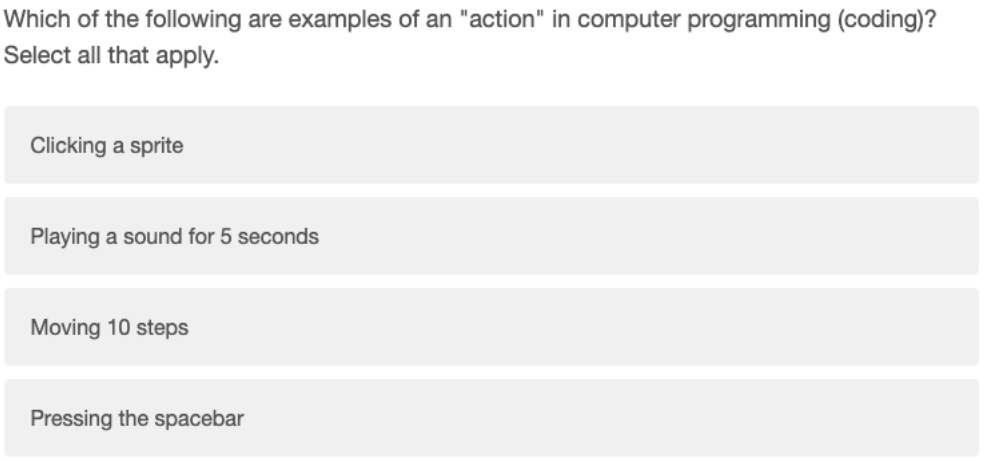


Figure A.2: Action "Remember" Question

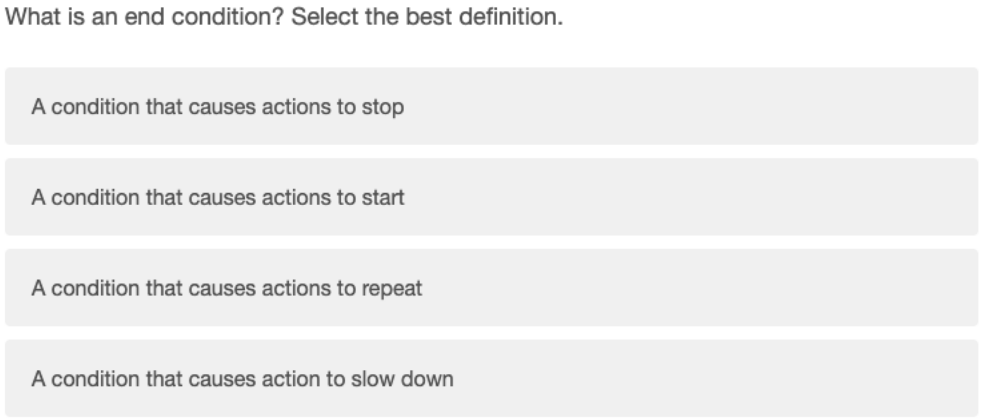


Figure A.3: End Condition "Remember" Question

A.2 Learning Goal 2: Create Scripts with Dependent Sprite Actions

The second learning goal is for students to create scripts that will trigger the action of one sprite dependent on the action of another sprite. We designed three questions for this learning goal, two targeting the "Remember" level of Bloom's taxonomy and one targeting the "Understand" level. The two "Remember" questions asked students to recall situations when they would use the `Repeat Until` (Figure A.5) and `Wait Until` (Figure A.6) blocks. The "Understand" question asks students to watch a video of a golf club hitting a ball into a

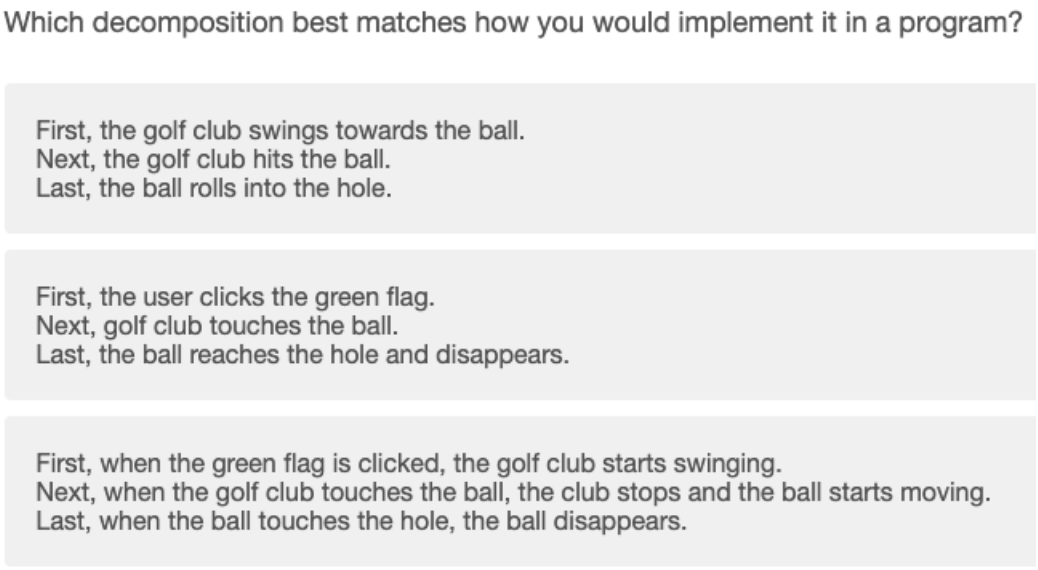


Figure A.4: Decomposition "Understand" Question

hole and then build the script for the ball from code snippets like a Parsons problem (Figure A.7).

A.3 Learning Goal 3: Use Sensing Blocks to Start & Stop Actions

The last learning goal is for students to use sensing blocks to start and stop actions. There are three questions targeting this learning goal, all at the "Understand" level. For two of the questions, students are shown two scripts, one for a needle and another for a balloon, where the needle moves to the balloon and pops it. Students are asked which events triggered the needle moving (Figure A.8) and the balloon popping (Figure A.9). The last question is the Parsons-style problem mentioned in the previous section (Figure A.7).

When would you use a "repeat until" block?

- When you want a sprite to do the same action a specific number of times
- When you want a sprite to do the same action forever
- When you want a script to pause until a condition is true
- When you want a sprite to do the same action until a condition is true

Figure A.5: Repeat Until "Remember" Question

When would you use a "wait until" block?

- When you want a sprite to do the same action until a condition is true
- When you want a script to pause until a condition is true
- When you want a sprite to do the same action a specific number of times
- When you want a sprite to do the same action forever

Figure A.6: Wait Until "Remember" Question

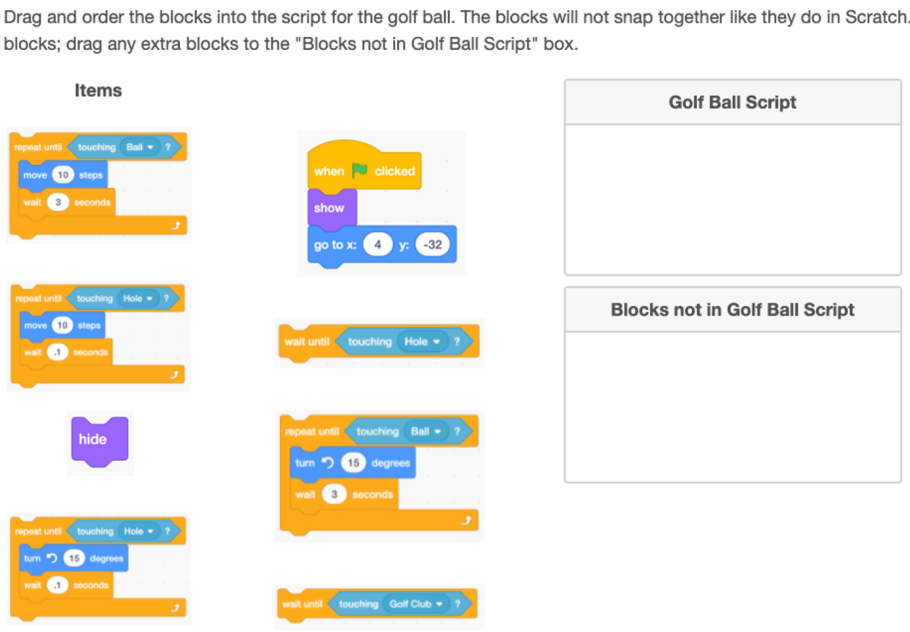
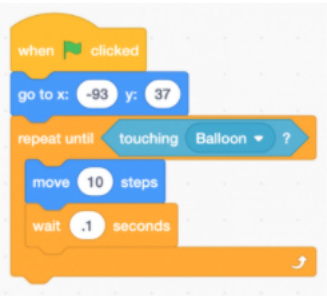


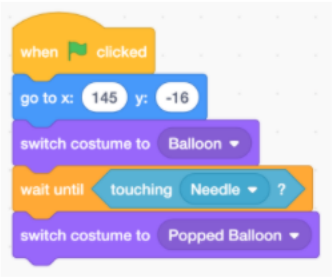
Figure A.7: Parsons-Style Problem for Golf Ball Script

The scripts below belong to a needle and a balloon:

Needle



Balloon



Which event caused the **Needle** to start moving?

- When the needle touches the balloon
- When the green flag is clicked
- When the needle is clicked
- When the spacebar is pressed

Figure A.8: Needle Event "Understand" Question

Which event caused the **Balloon** to pop?

- When the needle is clicked
- When the green flag is clicked
- When the spacebar is pressed
- When the needle touches the balloon

Figure A.9: Balloon Event "Understand" Question

REFERENCES

[1] National assessment of education program report card, 1990.

[2] *Scholastic Reading Inventory Technical Guide*, 2014.

[3] *Content Specifications for the Summative Assessment of the Common Core State Standards for Mathematics*, 2015.

[4] http://www.corestandards.org/, 2019.

[5] Smarter balanced validity research: Reporting scores, 2019.

[6] https://www.canonlab.org/scratchact1modules, 2021.

[7] https://www.csforall.org/, 2021.

[8] Jamal Abedi and Patricia Gándara. Performance of english language learners as a subgroup in large-scale assessment: Interaction of research and policy. *Educational Measurement: Issues and Practice*, 25(4):36–46, 2006.

[9] Joel C Adams and Andrew R Webster. What do students learn about programming from game, music video, and storytelling projects? In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 643–648. ACM, 2012.

[10] B. Akram, W. Min, E. Wiebe, B. Mott, K. E. Boyer, and J. Lester. Assessing middle school students' computational thinking through programming trajectory analysis. In *Proceedings of the 50th ACM technical symposium on Computer science education*, page 1269. ACM, 2019.

[11] Sabah Al-Fedaghi and Esraa Haidar. Programming is diagramming is programming. *J. Softw.*, 14(9):410–422, 2019.

[12] Ashok R Basawapatna, Alexander Repenning, Kyu Han Koh, and Hilarie Nickerson. The zones of proximal flow: guiding students through a space of computational thinking skills and challenges. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 67–74. ACM, 2013.

[13] Michael T Battista. Conceptualizations and issues related to learning progressions, learning trajectories, and levels of sophistication. *The Mathematics Enthusiast*, 8(3):507–570, 2011.

[14] Mark Benisz, John O Willis, and Ron Dumont. 16 abuses and misuses of intelligence tests: Facts and misconceptions. *Pseudoscience: The conspiracy against science*, page 351, 2018.

[15] Susan Bergin and Ronan Reilly. Programming: factors that influence success. In *ACM SIGCSE Bulletin*, volume 37, pages 411–415. ACM, 2005.

[16] Susan Bergin, Ronan Reilly, and Desmond Traynor. Examining the role of self-regulated learning on introductory programming performance. In *Proceedings of the first international workshop on Computing education research*, pages 81–86. ACM, 2005.

[17] John B Biggs and Kevin F Collis. *Evaluation the quality of learning: the SOLO taxonomy (structure of the observed learning outcome)*. Academic Press, 1982.

[18] John B Biggs and Kevin F Collis. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, 2014.

[19] Benjamin S Bloom et al. Taxonomy of educational objectives. vol. 1: Cognitive domain. *New York: McKay*, pages 20–24, 1956.

[20] Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Diana Franklin. Hairball: Lint-inspired static analysis of scratch projects. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 215–220, New York, NY, USA, 2013. ACM.

[21] Steven A Bogaerts. Limited time and experience: Parallelism in cs1. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 1071–1078. IEEE, 2014.

[22] Steven A Bogaerts. One step at a time: Parallelism in an introductory programming course. *Journal of Parallel and Distributed Computing*, 105:4–17, 2017.

[23] Suzanne M Bouffard, Christopher Wimer, Pia Caronongan, Priscilla Little, Eric Dearing, and Sandra D Simpkins. Demographic differences in patterns of youth out-of-school time activity participation. *Journal of Youth Development*, 1(1):24–40, 2006.

[24] Karen Brennan, M Chung, and J Hawson. Creative computing: A design-based introduction to computational thinking. *Retrieved May*, 9:2012, 2011.

[25] Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*, volume 1, page 25, 2012.

[26] Rusty Bresser, Sharon Fargason, and In this excerpt from Chapter. Opportunities and challenges for ells in the science inquiry classroom (part 1).

[27] Ruven Brooks. Towards a theory of the comprehension of computer programs. *International journal of man-machine studies*, 18(6):543–554, 1983.

[28] Kristy A Brugar and Kathryn Roberts. Timelines: An opportunity for meeting standards through textbook reading. *The Social Studies*, 105(5):230–236, 2014.

[29] Jack Buckley, Mark Schneider, and Yi Shang. Fix it and they might stay: School facility quality and teacher retention in washington, dc. *Teachers College Record*, 107(5):1107–1123, 2005.

[30] Mohamad Adam Bujang, Evi Diana Omar, and Nur Akmal Baharum. A review on sample size determination for cronbach's alpha test: a simple guide for researchers. *The Malaysian journal of medical sciences: MJMS*, 25(6):85, 2018.

[31] Eileen D Bunderson and Mary Elizabeth Christensen. An analysis of retention problems for female students in university computer science programs. *Journal of Research on Computing in Education*, 28(1):1–18, 1995.

[32] Quinn Burke and Yasmin Kafai. The writers' workshop for youth programmers digital storytelling with scratch in middle school classrooms. *ACM Proceedings of the Annual ACM Symposium on Computer Science Education*, pages 433–438, 02 2012.

[33] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. Eye movements in code reading: Relaxing the linear order. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 255–265. IEEE, 2015.

[34] Teresa Busjahn, Carsten Schulte, and Andreas Busjahn. Analysis of code reading to gain more insight in program comprehension. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, pages 1–9, 2011.

[35] Pat Byrne and Gerry Lyons. The effect of student attributes on success in programming. *ACM SIGCSE Bulletin*, 33(3):49–52, 2001.

[36] AJ Cañas, P Reiska, M Åhlberg, and JD Novak. Concept mapping & vee diagramming a primary mathematics sub-topic:"time". 2008.

[37] Susan Chambers Cantrell. Effective teaching and literacy learning: A look inside primary classrooms. *The Reading Teacher*, 52(4):370–378, 1998.

[38] Robbie Case. Intellectual development from birth to adulthood: A neo-piagetian interpretation. *Children's thinking: What develops*, pages 37–71, 1978.

[39] Francisco Enrique Vicente Castro and Kathi Fisler. On the interplay between bottom-up and datatype-driven program design. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 205–210, 2016.

[40] Raymond B Cattell. The scree test for the number of factors. *Multivariate behavioral research*, 1(2):245–276, 1966.

[41] Jeanne Century, Kaitlyn A Ferris, and Huifang Zuo. Finding time for computer science in the elementary school day: a quasi-experimental study of a transdisciplinary problem-based learning approach. *International Journal of STEM Education*, 7:1–16, 2020.

[42] Alexandra Chen, Catherine Panter-Brick, Kristin Hadfield, Rana Dajani, Amar Hamoudi, and Margaret Sheridan. Minds under siege: Cognitive signatures of poverty and trauma in refugee and non-refugee adolescents. *Child development*, 90(6):1856–1865, 2019.

[43] Jacob Cohen. Statistical power analysis for the behavioural sciences, 1988.

[44] Michael Lamport Commons. Introduction to the model of hierarchical complexity and its relationship to postformal action. *World Futures*, 64(5-7):305–320, 2008.

[45] Creative Computing. An introductory computing curriculum using scratch.

[46] Damien C Cormier, Okan Bulut, Kevin S McGrew, and Deepak Singh. Exploring the relations between cattell–horn–carroll (chc) cognitive abilities and mathematics achievement. *Applied Cognitive Psychology*, 31(5):530–538, 2017.

[47] National Research Council et al. *How people learn: Brain, mind, experience, and school: Expanded edition*. National Academies Press, 2000.

[48] Mihaly Csikszentmihalyi, Sami Abuhamdeh, and Jeanne Nakamura. Flow. In *Flow and the foundations of positive psychology*, pages 227–238. Springer, 2014.

[49] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. Using tracing and sketching to solve programming problems: Replicating and extending an analysis of what students draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, page 164–172, New York, NY, USA, 2017. Association for Computing Machinery.

[50] Quintin Cutts, Matthew Barr, Mireilla Bikanga Ada, Peter Donaldson, Steve Draper, Jack Parkinson, Jeremy Singer, and Lovisa Sundin. Experience report: Thinkathon–countering an" i got it working" mentality with pencil-and-paper exercises. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 203–209, 2019.

[51] Quintin Cutts, Sally Fincher, Patricia Haden, Anthony Robins, Ken Sutton, Bob Baker, Ilona Box, Michael de Raadt, John Hamer, Margaret Hamilton, et al. The ability to articulate strategy as a predictor of programming skill. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pages 181–188. Australian Computer Society, Inc., 2006.

[52] Valentina Dagiene and Gabriele Stupuriene. Bebras–a sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in education*, 15(1):25–44, 2016.

[53] Junhua Dang, Shanshan Xiao, and Siegfried Dewitte. Commentary:"poverty impedes cognitive function" and "the poor's poor mental power". *Frontiers in psychology*, 6:1037, 2015.

[54] Susan De La Paz. Effects of historical reasoning instruction and writing strategy mastery in culturally and academically diverse middle school classrooms. *Journal of educational psychology*, 97(2):139, 2005.

[55] Andreas Demetriou, Michael Shayer, and Anastasia Efklides. *Neo-Piagetian theories of cognitive development: Implications and applications for education*. Routledge, 2016.

[56] DD Deshler and JB Schumaker. Strategies instruction: A new way to teach. *Salt Lake City: Worldwide Media*, 1984.

[57] Betsy DiSalvo, Cecili Reid, and Parisa Khanipour Roshan. They can't find us: the search for informal cs education. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 487–492. ACM, 2014.

[58] Peggy Doerschuk, Jiangjiang Liu, and Judith Mann. Increasing participation of females and underrepresented minorities in computing. *Computer*, 42(4):110–113, 2009.

[59] Anouk S Donker, Hester De Boer, Danny Kostons, CC Dignath Van Ewijk, and Margaretha PC van der Werf. Effectiveness of learning strategy instruction on academic performance: A meta-analysis. *Educational Research Review*, 11:1–26, 2014.

[60] Barbara Ericson and Tom McKlin. Effective and sustainable computing summer camps. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, page 289–294, New York, NY, USA, 2012. Association for Computing Machinery.

[61] Barbara Ericson and Tom McKlin. Effective and sustainable computing summer camps. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 289–294. ACM, 2012.

[62] Gabriel Estrella, Jacky Au, Susanne M Jaeggi, and Penelope Collins. Is inquiry science instruction effective for english language learners? a meta-analytic review. *AERA open*, 4(2):2332858418767402, 2018.

[63] Gary W Evans and Thomas E Fuller-Rowell. Childhood poverty, chronic stress, and young adult working memory: The protective role of self-regulatory capacity. *Developmental science*, 16(5):688–696, 2013.

[64] Katrina Falkner, Rebecca Vivian, and Nickolas JG Falkner. Identifying computer science self-regulated learning strategies. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 291–296. ACM, 2014.

[65] Cheri Fancsali, Linda Tigani, Paulina Toro Isaza, and Rachel Cole. A landscape study of computer science education in nyc: Early findings and implications for policy and practice. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 44–49. ACM, 2018.

[66] Febrian Febrian et al. The instruction to overcome the inert knowledge issue in solving mathematical problem. *Jurnal Gantang*, 1(1):16–25, 2016.

[67] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to design programs: an introduction to programming and computing*. MIT Press, 2018.

[68] Kurt W Fischer. A theory of cognitive development: The control and construction of hierarchies of skills. *Psychological review*, 87(6):477, 1980.

[69] Kurt W Fischer and Daniel Bullock. Cognitive development in school-age children: Conclusions and new directions. *Development during middle childhood: The years from six to twelve*, pages 70–146, 1984.

[70] Dawn P Flanagan and Shauna G Dixon. The cattell-horn-carroll theory of cognitive abilities. *Encyclopedia of special education: A reference for the education of children, adolescents, and adults with disabilities and other exceptional individuals*, 2013.

[71] Louise P Flannery, Brian Silverman, Elizabeth R Kazakoff, Marina Umaschi Bers, Paula Bontá, and Mitchel Resnick. Designing scratchjr: support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children*, pages 1–10. ACM, 2013.

[72] Eric Fouh, Monika Akbar, and Clifford A Shaffer. The role of visualization in computer science education. *Computers in the Schools*, 29(1-2):95–117, 2012.

[73] Baker Franke, Jeanne Century, Michael Lach, Cameron Wilson, Mark Guzdial, Gail Chapman, and Owen Astrachan. Expanding access to k-12 computer science education: research on the landscape of computer science professional development. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 541–542. ACM, 2013.

[74] Diana Franklin, Merijke Coenraad, Jennifer Palmer, Donna Eatinger, Anna Zipp, Marco Anaya, Max White, Hoang Pham, Ozan Gökdemir, and David Weintrop. An analysis of use-modify-create pedagogical approach's success in balancing structure and student agency. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pages 14–24, 2020.

[75] Diana Franklin, Phillip Conrad, Bryce Boe, Katy Nilsen, Charlotte Hill, Michelle Len, Greg Dreschler, Gerardo Aldana, Paulo Almeida-Tanaka, Brynn Kiefer, et al. Assessment of computer science learning in a scratch-based outreach program. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 371–376. ACM, 2013.

[76] Diana Franklin, Jean Salac, Zachary Crenshaw, Saranya Turimella, Zipporah Klain, Marco Anaya, and Cathy Thomas. Exploring student behavior using the tipp&see learning strategy. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pages 91–101, 2020.

[77] Diana Franklin, Gabriela Skifstad, Reiny Rolock, Isha Mehrotra, Valerie Ding, Alexandria Hansen, David Weintrop, and Danielle Harlow. Using upper-elementary student performance to understand conceptual sequencing in a blocks-based curriculum. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 231–236. ACM, 2017.

[78] Diana Franklin, David Weintrop, Jennifer Palmer, Merijke Coenraad, Melissa Cobian, Kristan Beck, Andrew Rasmussen, Sue Krause, Max White, Marco Anaya, et al. Scratch encore: The design and pilot of a culturally-relevant intermediate scratch curriculum. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 794–800, 2020.

[79] Lynn S Fuchs, Amelia S Malone, Robin F Schumacher, Jessica Namkung, and Amber Wang. Fraction intervention for students with mathematics difficulties: Lessons learned from five randomized controlled trials. *Journal of Learning Disabilities*, 50(6):631–639, 2017.

[80] Ursula Fuller, Colin G Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L Lewis, Donna McGee Thompson, Charles Riedesel, et al. Developing a computer science-specific learning taxonomy. *ACM SIGCSE Bulletin*, 39(4):152–170, 2007.

[81] Russell Gersten, Lynn S Fuchs, Joanna P Williams, and Scott Baker. Teaching reading comprehension strategies to students with learning disabilities: A review of research. *Review of educational research*, 71(2):279–320, 2001.

[82] Michail N Giannakos, Spyros Doukakis, Ilias O Pappas, Nikos Adamopoulos, and Panagiota Giannopoulou. Investigating teachers' confidence on technological pedagogical and content knowledge: an initial validation of tpack scales in k-12 computing education context. *Journal of Computers in Education*, 2(1):43–59, 2015.

[83] Allison F Gilmour, Douglas Fuchs, and Joseph H Wehby. Are students with disabilities accessing the curriculum? a meta-analysis of the reading achievement gap between students with and without disabilities. *Exceptional Children*, 85(3):329–346, 2019.

[84] Google and Gallup. *Diversity Gaps in Computer Science: Exploring the Underrepresentation of Girls, Blacks and Hispanics*. 2016.

[85] Google and Gallup. *Trends in the State of Computer Science in U.S. K-12 Schools*. 2016.

[86] Annagret Goold and Russell Rimmer. Factors affecting performance in first-year computing. *ACM SIGCSE Bulletin*, 32(2):39–43, 2000.

[87] Steve Graham, Debra McKeown, Sharlene Kiuhara, and Karen R Harris. A meta-analysis of writing instruction for students in the elementary grades. *Journal of educational psychology*, 104(4):879, 2012.

[88] Shuchi Grover, Roy Pea, and Stephen Cooper. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25(2):199–237, 2015.

[89] Shuchi Grover, Roy Pea, and Stephen Cooper. Factors influencing computer science learning in middle school. In *Proceedings of the 47th ACM technical symposium on computing science education*, pages 552–557. ACM, 2016.

[90] Sigrun Gudmundsdottir and Lee Shulman. Pedagogical content knowledge in social studies. *Scandinavian Journal of Educationl Research*, 31(2):59–70, 1987.

[91] Daniel A Hackman, Laura M Betancourt, Robert Gallop, Daniel Romer, Nancy L Brodsky, Hallam Hurt, and Martha J Farah. Mapping the trajectory of socioeconomic disparity in working memory: Parental and neighborhood factors. *Child development*, 85(4):1433–1445, 2014.

[92] Dianne Hagan and Selby Markham. Does it help to have some programming experience before beginning a computing degree program? In *ACM SIGCSE Bulletin*, volume 32, pages 25–28. ACM, 2000.

[93] Graeme Sydney Halford. Children's understanding: The development of mental models. 1993.

[94] David Hammer, Bruce Sherin, Tina Kolpakowski, et al. Inventing graphing: Meta-representational expertise in children. *Journal of mathematical behavior*, 10(2):117–160, 1991.

[95] David Hammer and TIFFANY-ROSE SIKORSKI. Implications of complexity for research on learning progressions. *Science Education*, 99(3):424–431, 2015.

[96] Alexandria K Hansen, Eric R Hansen, Hilary A Dwyer, Danielle B Harlow, and Diana Franklin. Differentiating for diversity: Using universal design for learning in elementary computer science education. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 376–381, 2016.

[97] Eric A Hanushek, Paul E Peterson, Laura M Talpey, and Ludger Woessmann. The unwavering ses achievement gap: Trends in us student performance. Technical report, National Bureau of Economic Research, 2019.

[98] Idit Ed Harel and Seymour Ed Papert. *Constructionism.* Ablex Publishing, 1991.

[99] James J Higgins, R Clifford Blair, and Suleiman Tashtoush. The aligned rank transform procedure. 1990.

[100] Dennis E Hinkle, William Wiersma, and Stephen G Jurs. *Applied statistics for the behavioral sciences*, volume 663. Houghton Mifflin College Division, 2003.

[101] Dennis E Hinkle, William Wiersma, Stephen G Jurs, et al. Applied statistics for the behavioral sciences. 1988.

[102] Jennifer Jellison Holme. Buying homes, buying schools: School choice and the social construction of school quality. *Harvard Educational Review*, 72(2):177–206, 2002.

[103] Peter Hubwieser, Michail N Giannakos, Marc Berges, Torsten Brinda, Ira Diethelm, Johannes Magenheim, Yogendra Pal, Jana Jackova, and Egle Jasute. A global snapshot of computer science education in k-12 schools. In *Proceedings of the 2015 ITiCSE on working group reports*, pages 65–83. ACM, 2015.

[104] Human Resources Research Organization. Independent evaluation of the validity and reliability of staar grades 3-8 assessment scores, 2016.

[105] Maya Israel, Cecelia Ribuffo, and Sean Smith. Universal design for learning innovation configuration: Recommendations for teacher preparation and professional development (document no. ic-7). *Retrieved from University of Florida, Collaboration for Effective Educator, Development, Accountability, and Reform Center website: http://ceedar. education. ufl. edu/tools/innovation-configurations*, 2014.

[106] Maya Israel, Quentin M Wherfel, Jamie Pearson, Saadeddine Shehab, and Tanya Tapia. Empowering k–12 students with disabilities to learn computational thinking and computer programming. *TEACHING Exceptional Children*, 48(1):45–53, 2015.

[107] Janis E Jacobs and Martha M Bleeker. Girls' and boys' developing interests in math and science: Do parents matter? *New directions for child and adolescent development*, (106):5–21, 2004.

[108] Asha Jitendra and Yan Ping Xin. Mathematical word-problem-solving instruction for students with mild disabilities and students at risk for math failure: A research synthesis. *The Journal of Special Education*, 30(4):412–438, 1997.

[109] Colin G Johnson and Ursula Fuller. Is bloom's taxonomy appropriate for computer science? In *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, pages 120–123, 2006.

[110] Timothy R Jordan, Abubaker AA Almabruk, Eman A Gadalla, Victoria A McGowan, Sarah J White, Lily Abedipour, and Kevin B Paterson. Reading direction and the central perceptual span: Evidence from arabic and english. *Psychonomic bulletin & review*, 21(2):505–511, 2014.

[111] Yasmin Kafai, Kristin Searle, Crîstobal Martinez, and Bryan Brayboy. Ethnocomputing with electronic textiles: culturally responsive open design to broaden participation in computing in american indian youth and communities. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 241–246. ACM, 2014.

[112] Henry F Kaiser. The application of electronic computers to factor analysis. *Educational and psychological measurement*, 20(1):141–151, 1960.

[113] Michael T Kane. Validating the interpretations and uses of test scores. *Journal of Educational Measurement*, 50(1):1–73, 2013.

[114] Michael T Kane. Explicating validity. *Assessment in Education: Principles, Policy & Practice*, 23(2):198–211, 2016.

[115] Seta Kazandjian and Sylvie Chokron. Paying attention to reading direction. *Nature Reviews Neuroscience*, 9(12):965–965, 2008.

[116] Aaron Keen and Kurt Mammen. Program decomposition and complexity in cs1. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 48–53, 2015.

[117] Paul A Kirschner. Cognitive load theory: Implications of cognitive load theory on the design of learning, 2002.

[118] Janette K Klingner and Sharon Vaughn. Using collaborative strategic reading. *Teaching exceptional children*, 30(6):32–37, 1998.

[119] Amy J Ko, Thomas D LaToza, Stephen Hull, Ellen A Ko, William Kwok, Jane Quichocho, Harshitha Akkaraju, and Rishin Pandit. Teaching explicit programming strategies to adolescents. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 469–475, 2019.

[120] Matthew Koehler and Punya Mishra. What is technological pedagogical content knowledge (tpack)? *Contemporary issues in technology and teacher education*, 9(1):60–70, 2009.

[121] Yifat Ben-David Kolikant. Gardeners and cinema tickets: High school students' preconceptions of concurrency. *Computer Science Education*, 11(3):221–245, 2001.

[122] David R Krathwohl. A revision of bloom's taxonomy: An overview. *Theory into practice*, 41(4):212–218, 2002.

[123] Jennifer Krawec and Marissa Steinberg. Inquiry-based instruction in mathematics for students with learning disabilities: A review of the literature. *Learning Disabilities: A Multidisciplinary Journal*, 24(2):27–35, 2019.

[124] Richard E Ladner and Maya Israel. For all" in" computer science for all. *Communications of the ACM*, 59(9):26–28, 2016.

[125] Daniël Lakens. Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and anovas. *Frontiers in psychology*, 4:863, 2013.

[126] Thomas D LaToza, Maryam Arab, Dastyni Loksa, and Amy J Ko. Explicit programming strategies. *Empirical Software Engineering*, 25(4):2416–2449, 2020.

[127] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. Computational thinking for youth in practice. *Acm Inroads*, 2(1):32–37, 2011.

[128] Michael J Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, et al. Principles of a debugging-first puzzle game for computing education. In *2014 IEEE symposium on visual languages and human-centric computing (VL/HCC)*, pages 57–64. IEEE, 2014.

[129] Michael J Lee and Amy J Ko. Comparing the effectiveness of online learning approaches on cs1 learning outcomes. In *Proceedings of the eleventh annual international conference on international computing education research*, pages 237–246. ACM, 2015.

[130] Nancy L Leech and Anthony J Onwuegbuzie. A call for greater use of nonparametric statistics. 2002.

[131] Colleen Lennon and Hal Burdick. The lexile framework as an approach for reading measurement and success. *electronic publication on www. lexile. com*, 2004.

[132] Jimmie Leppink, Fred Paas, Cees PM Van der Vleuten, Tamara Van Gog, and Jeroen JG Van Merriënboer. Development of an instrument for measuring different types of cognitive load. *Behavior research methods*, 45(4):1058–1072, 2013.

[133] Stanley Letovsky. Cognitive processes in program comprehension. *Journal of Systems and software*, 7(4):325–339, 1987.

[134] Gary Lewandowski, Dennis J Bouvier, Robert McCartney, Kate Sanders, and Beth Simon. Commonsense computing (episode 3): concurrency and concert tickets. In *Proceedings of the third international workshop on Computing education research*, pages 133–144. ACM, 2007.

[135] Colleen M Lewis and Niral Shah. Building upon and enriching grade four mathematics standards with programming curriculum. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 57–62. ACM, 2012.

[136] Marcia C Linn and Anne C Petersen. Emergence and characterization of sex differences in spatial ability: A meta-analysis. *Child development*, pages 1479–1498, 1985.

[137] Raymond Lister, Elizabeth S Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, et al. A multi-national study of reading and tracing skills in novice programmers. In *ACM SIGCSE Bulletin*, volume 36, pages 119–150. ACM, 2004.

[138] Raymond Lister, Colin Fidge, and Donna Teague. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin*, 41(3):161–165, 2009.

[139] Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L Whalley, and Christine Prasad. Not seeing the forest for the trees: novice programmers and the solo taxonomy. *ACM SIGCSE Bulletin*, 38(3):118–122, 2006.

[140] David C Littman, Jeannine Pinto, Stanley Letovsky, and Elliot Soloway. Mental models and software maintenance. *Journal of Systems and Software*, 7(4):341–355, 1987.

[141] Dastyni Loksa, Benjamin Xie, Harrison Kwik, and Amy J Ko. Investigating novices' in situ reflections on their programming process. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 149–155, 2020.

[142] Nicholas Lytle, Veronica Cateté, Danielle Boulden, Yihuan Dong, Jennifer Houchins, Alexandra Milliken, Amy Isvik, Dolly Bounajim, Eric Wiebe, and Tiffany Barnes. Use, modify, create: Comparing computational thinking lesson progressions for stem classes. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 395–401, 2019.

[143] Jessica Middlemis Maher, Jonathan C Markey, and Diane Ebert-May. The other half of the story: effect size analysis in quantitative research. *CBE—Life Sciences Education*, 12(3):345–351, 2013.

[144] John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. *Programming by choice: urban youth learning programming with scratch*, volume 40. ACM, 2008.

[145] Suzanne Liff Manz. A strategy for previewing textbooks: teaching readers to become thieves.(teaching ideas). *The Reading Teacher*, 55(5):434–436, 2002.

[146] Jane Margolis. *Stuck in the shallow end: Education, race, and computing*. MIT Press, 2010.

[147] Lauren E Margulieux. Spatial encoding strategy theory: The relationship between spatial skill and stem achievement. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 81–90. ACM, 2019.

[148] Eva Marinus, Zoe Powell, Rosalind Thornton, Genevieve McArthur, and Stephen Crain. Unravelling the cognition of coding in 3-to-6-year olds: The development of an assessment tool and the relation between coding ability and cognitive compiling of syntax in natural language. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 133–141. ACM, 2018.

[149] Richard E Mayer. *Applying the science of learning*. Pearson/Allyn & Bacon Boston, MA, 2011.

[150] Allison L McGrath and Marie Tejero Hughes. Students with learning disabilities in inquiry-based science classrooms: A cross-case analysis. *Learning Disability Quarterly*, 41(3):131–143, 2018.

[151] Mary L McHugh. The chi-square test of independence. *Biochemia medica*, 23(2):143–149, 2013.

[152] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. Learning computer science concepts with scratch. *Computer Science Education*, 23(3):239–264, 2013.

[153] Katherine Michelmore and Susan Dynarski. The gap within the gap: Using longitudinal data to understand income differences in educational outcomes. *AERA Open*, 3(1):2332858417692958, 2017.

[154] Alexandra Milliken, Wengran Wang, Veronica Cateté, Sarah Martin, Neeloy Gomes, Yihuan Dong, Rachel Harred, Amy Isvik, Tiffany Barnes, Thomas Price, et al. Planit! a new integrated tool to help novices design for open-ended projects. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 232–238, 2021.

[155] Robert J Mislevy and Geneva D Haertel. Implications of evidence-centered design for educational testing. *Educational Measurement: Issues and Practice*, 25(4):6–20, 2006.

[156] Marjorie Montague. The effects of cognitive and metacognitive strategy instruction on the mathematical problem solving of middle school students with learning disabilities. *Journal of learning disabilities*, 25(4):230–248, 1992.

[157] Jesús Moreno-León, Marcos Román-González, Casper Harteveld, and Gregorio Robles. On the automatic assessment of computational thinking skills: A comparison with human experts. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, pages 2788–2795, New York, NY, USA, 2017. ACM.

[158] P. L. Morgan, G. Farkas, Y. Wang, M. M.. Hillemeier, and S. Maczuga. Executive function deficits in kindergarten predict repeated academic difficulties across elementary school. *Early childhood research quarterly*, 46.

[159] Kazunori Morikawa and Michael K McBeath. Lateral motion bias associated with reading direction. *Vision Research*, 32(6):1137–1141, 1992.

[160] Briana B. Morrison, Brian Dorn, and Mark Guzdial. Measuring cognitive load in introductory cs: Adaptation of an instrument. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, page 131–138, New York, NY, USA, 2014. Association for Computing Machinery.

[161] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, page 21–29, New York, NY, USA, 2015. Association for Computing Machinery.

[162] Duygu Mutlu-Bayraktar, Veysel Cosgun, and Tugba Altan. Cognitive load in multimedia learning environments: A systematic review. *Computers & Education*, 141:103618, 2019.

[163] Thomas L Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, et al. Exploring the role of visualization and engagement in computer science education. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 131–152. 2002.

[164] Isaac Nassi and Ben Shneiderman. Flowchart techniques for structured programming. *ACM Sigplan Notices*, 8(8):12–26, 1973.

[165] National Center for Education Statistics. The condition of education, 2019.

[166] Greg L Nelson, Benjamin Xie, and Amy J Ko. Comprehension first: evaluating a novel pedagogy and tutoring system for program tracing in cs1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 2–11, 2017.

[167] Vicente Lustosa Neto, Roberta Coelho, Larissa Leite, Dalton S Guerrero, and Andrea P Mendonça. Popt: a problem-oriented programming and testing approach for novice students. In *2013 35th international conference on software engineering (ICSE)*, pages 1099–1108. IEEE, 2013.

[168] Lijun Ni, Diane Schilder, Mark Sherman, and Fred Martin. Computing with a community focus: outcomes from an app inventor summer camp for middle school students. *Journal of Computing Sciences in Colleges*, 31(6):82–89, 2016.

[169] Suzan Nouwens, Margriet A Groen, and Ludo Verhoeven. How working memory relates to children's reading comprehension: the importance of domain-specificity in storage and processing. *Reading and writing*, 30(1):105–120, 2017.

[170] Joseph D Novak. Concept maps and vee diagrams: Two metacognitive tools to facilitate meaningful learning. *Instructional science*, 19(1):29–52, 1990.

[171] Peter Akinsola Okebukola. Attitude of teachers towards concept mapping and vee diagramming as metalearning tools in science and mathematics. *Educational Research*, 34(3):201–213, 1992.

[172] Fred Paas, Tamara Van Gog, and John Sweller. Cognitive load theory: New conceptualizations, specifications, and integrated research perspectives. *Educational psychology review*, 22(2):115–121, 2010.

[173] Fred Paas and Jeroen JG van Merriënboer. Cognitive-load theory: Methods to manage working memory load in the learning of complex tasks. *Current Directions in Psychological Science*, 29(4):394–398, 2020.

[174] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[175] Miranda C Parker, Yvonne S Kao, Dana Saito-Stehberger, Diana Franklin, Susan Krause, Debra Richardson, and Mark Warschauer. Development and preliminary validation of the assessment of computing for elementary students (aces). In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 10–16, 2021.

[176] Roy D Pea and D Midian Kurland. On the cognitive prerequisites of learning computer programming. 1983.

[177] Roy D Pea and D Midian Kurland. On the cognitive effects of learning computer programming. *New ideas in psychology*, 2(2):137–168, 1984.

[178] Janice L Pearce, Mario Nakazawa, and Scott Heggen. Improving problem decomposition ability in cs1 through explicit guided inquiry-based instruction. *J. Comput. Sci. Coll*, 31(2):135–144, 2015.

[179] Peng Peng and Douglas Fuchs. A meta-analysis of working memory deficits in children with learning difficulties: Is there a difference between verbal domain and numerical domain? *Journal of learning disabilities*, 49(1):3–20, 2016.

[180] Nancy Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology*, 19(3):295–341, 1987.

[181] Ken Perlin, Zhenyi He, and Karl Rosenberg. Chalktalk: A visualization and communication language–as a tool in the domain of computer science education. *arXiv preprint arXiv:1809.07166*, 2018.

[182] Marian Petre. Uml in practice. In *2013 35th international conference on software engineering (icse)*, pages 722–731. IEEE, 2013.

[183] Jean Piaget. Piaget's theory. In *Piaget and his school*, pages 11–23. Springer, 1976.

[184] Lori Pollock, Kathleen McCoy, Sandra Carberry, Namratha Hundigopal, and Xiaoxin You. Increasing high school girls' self confidence and awareness of cs through a positive summer experience. In *ACM SIGCSE Bulletin*, volume 36, pages 185–189. ACM, 2004.

[185] Michael Pressley. Metacognition and self-regulated comprehension. *What research has to say about reading instruction*, 3:291–309, 2002.

[186] Michael Pressley, John G Borkowski, and Wolfgang Schneider. Cognitive strategies: Good strategy users coordinate metacognition and knowledge. 2010.

[187] Yizhou Qian and James D Lehman. Correlates of success in introductory programming: A study with middle school students. *Journal of Education and Learning*, 5(2):73–83, 2016.

[188] Brian Rague. Measuring cs1 perceptions of parallelism. In *2011 Frontiers in Education Conference (FIE)*, pages S3E–1. IEEE, 2011.

[189] Vennila Ramalingam, Deborah LaBelle, and Susan Wiedenbeck. Self-efficacy and mental models in learning to program. In *ACM SIGCSE Bulletin*, volume 36, pages 171–175. ACM, 2004.

[190] Sean F Reardon. The widening income achievement gap. *Educational Leadership*, 70(8):10–16, 2013.

[191] Amy C Reichelt, R Fred Westbrook, and Margaret J Morris. Impact of diet on learning, memory and cognition. *Frontiers in behavioral neuroscience*, 11:96, 2017.

[192] Alexander Repenning and Andri Ioannidou. Broadening participation through scalable game design. In *ACM SIGCSE Bulletin*, volume 40, pages 305–309. ACM, 2008.

[193] Kathryn M Rich, T Andrew Binkowski, Carla Strickland, and Diana Franklin. Decomposition: A k-8 computational thinking learning trajectory. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 124–132, 2018.

[194] Kathryn M Rich, Diana Franklin, Carla Strickland, Andy Isaacs, and Donna Eatinger. A learning trajectory for variables based in computational thinking literature: Using levels of thinking to develop instruction. *Computer Science Education*, pages 1–22, 2020.

[195] Kathryn M Rich, Carla Strickland, T Andrew Binkowski, and Diana Franklin. A k-8 debugging learning trajectory derived from research literature. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 745–751, 2019.

[196] Kathryn M Rich, Carla Strickland, T Andrew Binkowski, Cheryl Moran, and Diana Franklin. K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 182–190. ACM, 2017.

[197] Kathryn M Rich, Carla Strickland, T Andrew Binkowski, Cheryl Moran, and Diana Franklin. K–8 learning trajectories derived from research literature: sequence, repetition, conditionals. *ACM Inroads*, 9(1):46–55, 2018.

[198] John TE Richardson. Eta squared and partial eta squared as measures of effect size in educational research. *Educational Research Review*, 6(2):135–147, 2011.

[199] Luca Rinaldi, Samuel Di Luca, Avishai Henik, and Luisa Girelli. Reading direction shifts visuospatial attention: An interactive account of attentional biases. *Acta psychologica*, 151:98–105, 2014.

[200] Karen L Rizzo and Jonte C Taylor. Effects of inquiry-based instruction on science achievement for students with disabilities: An analysis of the literature. *Journal of Science Education for Students with Disabilities*, 19(1):2, 2016.

[201] Kathryn L Roberts, Rebecca R Norman, Nell K Duke, Paul Morsink, Nicole M Martin, and Jennifer A Knight. Diagrams, timelines, and tables—oh, my! fostering graphical literacy. *The Reading Teacher*, 67(1):12–24, 2013.

[202] Anthony V Robins, Lauren Margulieux, and Briana B Morrison. Cognitive sciences for computing education. 2019.

[203] Kellie Rolstad, Kate Mahoney, and Gene V Glass. The big picture in bilingual education: A meta-analysis corrected for gersten's coding error. *Journal of Educational Research & Policy Studies*, 8(2):1–15, 2008.

[204] Carly Rosenzweig, Jennifer Krawec, and Marjorie Montague. Metacognitive strategy use of eighth-grade students with and without learning disabilities during mathematical problem solving: A think-aloud analysis. *Journal of learning disabilities*, 44(6):508–520, 2011.

[205] Jean Salac and Diana Franklin. If they build it, will they understand it? exploring the relationship between student code and performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 473–479, 2020.

[206] Jean Salac, Qi Jin, Zipporah Klain, Saranya Turimella, Max White, and Diana Franklin. Patterns in elementary-age student responses to personalized & generic code comprehension questions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 514–520, 2020.

[207] Jean Salac, Cathy Thomas, Chloe Butler, and Diana Franklin. Supporting diverse learners in k-8 computational thinking with tipp&see. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 246–252, 2021.

[208] Jean Salac, Cathy Thomas, Chloe Butler, Ashley Sanchez, and Diana Franklin. Tipp&see: A learning strategy to guide students through use-modify scratch activities. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 79–85, 2020.

[209] Jean Salac, Cathy Thomas, Bryan Twarek, William Marsland, and Diana Franklin. Comprehending code: Understanding the relationship between reading and math proficiency, and 4th-grade cs learning outcomes. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 268–274, 2020.

[210] Jean Salac, Max White, Ashley Wang, and Diana Franklin. An analysis through an equity lens of the implementation of computer science in k-8 classrooms in a large, urban school district. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 1150–1156, 2019.

[211] Philip Sands. Addressing cognitive load in the computer science classroom. *ACM Inroads*, 10(1):44–51, 2019.

[212] Fredrick A Schrank, Scott L Decker, and John M Garruto. *Essentials of WJ IV cognitive abilities assessment*. John Wiley & Sons, 2016.

[213] Fredrick Allen Schrank, Kevin S McGrew, Nancy Mather, Barbara J Wendling, and Erica M LaForte. *Woodcock-Johnson IV tests of cognitive abilities*. Riverside, 2014.

[214] Ann C Schulte, Joseph J Stevens, Stephen N Elliott, Gerald Tindal, and Joseph FT Nese. Achievement gaps for students with disabilities: Stable, widening, or narrowing on a state-wide reading comprehension test? *Journal of Educational Psychology*, 108(7):925, 2016.

[215] Carsten Schulte. Block model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the Fourth international Workshop on Computing Education Research*, pages 149–160. ACM, 2008.

[216] Thomas E Scruggs, Margo A Mastropieri, Sheri L Berkeley, and Lisa Marshak. Mnemonic strategies: Evidence-based practice and practice-based evidence. *Intervention in School and Clinic*, 46(2):79–86, 2010.

[217] Sue Sentance, Jane Waite, and Maria Kallia. Teachers' experiences of using primm to teach programming in school. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 476–482. ACM, 2019.

[218] Tina Seufert. The interplay between self-regulation in learning and cognitive load. *Educational Research Review*, 24:116–129, 2018.

[219] Mikyung Shin and Diane Pedrotty Bryant. A synthesis of mathematical and cognitive performances of students with mathematics learning disabilities. *Journal of learning disabilities*, 48(1):96–112, 2015.

[220] Ben Shneiderman and Richard Mayer. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer & Information Sciences*, 8(3):219–238, 1979.

[221] Juha Sorva. Notional machines and introductory programming education. *ACM Transactions on Computing Education*, 13:8:1–8:31, 06 2013.

[222] Juha Sorva. Notional machines and introductory programming education. *ACM Trans. Comput. Educ.*, 13(2), July 2013.

[223] M-A Storey. Theories, methods and tools in program comprehension: Past, present and future. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 181–191. IEEE, 2005.

[224] Michael Striewe and Michael Goedicke. Automated assessment of uml activity diagrams. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 336–336, 2014.

[225] Addison YS Su, Chester SJ Huang, Stephen JH Yang, Ting-Jou Ding, and YZ Hsieh. Effects of annotations and homework on learning achievement: An empirical study of scratch programming pedagogy. *Journal of Educational Technology & Society*, 18(4):331–343, 2015.

[226] Sangho Suh, Martinet Lee, Gracie Xia, et al. Coding strip: A pedagogical tool for teaching and learning programming concepts through comics. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–10. IEEE, 2020.

[227] Keith S Taber. The use of cronbach's alpha when developing and reporting research instruments in science education. *Research in Science Education*, 48(6):1273–1296, 2018.

[228] Harriet G Taylor and Luegina C Mounfield. Exploration of the relationship between prior computing experience and gender on success in college computer science. *Journal of educational computing research*, 11(4):291–306, 1994.

[229] Allison Elliott Tew and Mark Guzdial. Developing a validated assessment of fundamental cs1 concepts. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 97–101. ACM, 2010.

[230] Texas Education Agency. Staar performance standards, 2020.

[231] Texas Education Code. Texas essential knowledge and skills for mathematics, 2012.

[232] Cathy Thomas, Diana Franklin, and Jean Salac. Teacher perspectives of year-long professional development in inclusive elementary computer science, 2019.

[233] Cathy Newman Thomas, Delinda Van Garderen, Amy Scheuermann, and Eun Ju Lee. Applying a universal design for learning framework to mediate the language demands of mathematics. *Reading & Writing Quarterly*, 31(3):207–234, 2015.

[234] Michele Tine. Working memory differences between children living in rural and urban poverty. *Journal of Cognition and Development*, 15(4):599–613, 2014.

[235] Denise Tolhurst, Bob Baker, John Hamer, Ilona Box, Raymond Lister, Quintin Cutts, Marian Petre, Michael de Raadt, Anthony Robins, Sally Fincher, et al. Do map drawing styles of novice programmers predict success in programming?: a multi-national, multi-institutional study. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pages 213–222. Australian Computer Society, Inc., 2006.

[236] Keith Topping and Nancy Ferguson. Effective literacy teaching behaviours. *Journal of Research in Reading*, 28(2):125–143, 2005.

[237] Scott A Turner, Manuel A Pérez-Quinones, and Stephen H Edwards. minimuml: A minimalist approach to uml diagramming for early computer science education. *Journal on Educational Resources in Computing (JERIC)*, 5(4):1–es, 2005.

[238] Timothy Urness and Eric D Manley. Generating interest in computer science through middle-school android summer camps. *Journal of Computing Sciences in Colleges*, 28(5):211–217, 2013.

[239] Delinda van Garderen and Amy M Scheuermann. Diagramming word problems: A strategic approach for instruction. *Intervention in School and Clinic*, 50(5):282–290, 2015.

[240] Sharon Vaughn and Meaghan Edmonds. Reading comprehension for older readers. *Intervention in school and clinic*, 41(3):131–137, 2006.

[241] Rebecca Vivian and Katrina Falkner. Identifying teachers' technological pedagogical content knowledge for computer science in the primary years. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 147–155, 2019.

[242] Lev Vygotsky. Interaction between learning and development. *Readings on the development of children*, 23(3):34–41, 1978.

[243] David W Walker and James A Poteet. A comparison of two methods of teaching mathematics story problem-solving with learning disabled students. In *National Forum of Special Education Journal*, volume 1, pages 44–51. ERIC, 1990.

[244] Jennifer Wang, Hai Hong, Jason Ravitz, and Marielena Ivory. Gender differences in factors influencing pursuit of computer science and related fields. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 117–122. ACM, 2015.

[245] David C Webb, Alexander Repenning, and Kyu Han Koh. Toward an emergent theory of broadening participation in computer science education. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 173–178. ACM, 2012.

[246] Xin Wei, Keith B Lenz, and Jose Blackorby. Math growth trajectories of students with disabilities: Disability category, gender, racial, and socioeconomic status differences from ages 7 to 17. *Remedial and Special Education*, 34(3):154–165, 2013.

[247] Miriam Westervelt. Schoolyard inquiry for english language learners. *The Science Teacher*, 74(3):47, 2007.

[248] Susan Wiedenbeck, Deborah Labelle, and Vennila NR Kain. Factors affecting course outcomes in introductory programming. In *PPIG*, page 11. Citeseer, 2004.

[249] Joanna P Williams. Instruction in reading comprehension for primary-grade students: A focus on text structure. *The Journal of Special Education*, 39(1):6–18, 2005.

[250] Brenda Cantwell Wilson and Sharon Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. In *ACM SIGCSE Bulletin*, volume 33, pages 184–188. ACM, 2001.

[251] Jacob O Wobbrock, Leah Findlater, Darren Gergle, and James J Higgins. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 143–146, 2011.

[252] Ursula Wolz, Christopher Hallberg, and Brett Taylor. Scrape: A tool for visualizing the code of scratch programs. In *Poster presented at the 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX*, 2011.

[253] Karl L Wuensch. Nonparametric effect size estimators, 2015.

[254] Benjamin Xie, Dastyni Loksa, Greg L Nelson, Matthew J Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J Ko. A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2-3):205–253, 2019.

[255] Kun Yuan, Jeffrey Steedle, Richard Shavelson, Alicia Alonzo, and Marily Oppezzo. Working memory, fluid intelligence, and science learning. *Educational Research Review*, 1(2):83–98, 2006.