Investigating the Use of Planning Sheets in Young Learners' Open-Ended Scratch Projects

David Gonzalez-Maldonado
University of Chicago
Chicago, USA
dagm@uchicago.edu

Alex Pugnali
University of Maryland
College Park, USA
apugnali@umd.edu

Jennifer Tsan
University of Chicago
Chicago, USA
jennifertsan@uchicago.edu

Donna Eatinger
University of Chicago
Chicago, USA
dmeatinger@uchicago.edu

Diana Franklin
University of Chicago
Chicago, USA
dmfranklin@uchicago.edu

David Weintrop
University of Maryland
College Park, USA
weintrop@umd.edu

ABSTRACT

Open-ended tasks can be both beneficial and challenging to students learning to program. Such tasks allow students to be more creative and feel ownership over their work, but some students struggle with unstructured tasks and, without proper scaffolds, this can lead to negative learning experiences. Scratch is a widely used coding platform to teach computer science in classrooms and is designed to support learner creativity and expression. With its open-ended nature, Scratch can be used in various ways in the classroom to meet the needs of schools and districts. One challenge of using Scratch in classrooms is supporting learners in exploring their interests and fostering creativity while still meeting the instructional goals of a lesson and ensuring all students are engaged with, and understand, focal concepts and practices.

In this paper, we investigate the use of planning sheets to facilitate novice programmers designing and implementing Scratch programs based on open-ended prompts. To evaluate the planning sheets, we look at how closely students' implemented Scratch projects match their plans and whether the implemented Scratch projects met the technical requirements for the given lesson. We analyzed 303 Scratch projects from 155 middle grade students (ages 10-14) who were introduced to programming via the Scratch Encore Curriculum. Completed Scratch projects that used planning sheets (202) were qualitatively coded to evaluate how closely they matched the initial plan, and Scratch programs (303) were analyzed with an automated grader to check if technical project requirements were met. Our results reveal that students that used planning sheets met significantly more technical project requirements and had more complex structures than those that did not have planning sheets. Results differ based on teacher and type of planning sheet used (*physical* vs. *virtual*). This work suggests that planning sheets are a helpful tool for young learners when completing open-ended coding projects.

ICER 2022, August 7–11, 2022, Lugano and Virtual Event, Switzerland

ACM ISBN 978-1-4503-9194-8/22/08…$15.00
https://doi.org/10.1145/3501385.3543972

CCS CONCEPTS

• **Social and professional topics** → **K-12 education**; **Computing education**.

KEYWORDS

computer science education, planning, K-8

ACM Reference Format:
David Gonzalez-Maldonado, Alex Pugnali, Jennifer Tsan, Donna Eatinger, Diana Franklin, and David Weintrop. 2022. Investigating the Use of Planning Sheets in Young Learners' Open-Ended Scratch Projects. In *Proceedings of the 2022 ACM Conference on International Computing Education Research V.1 (ICER 2022), August 7–11, 2022, Lugano and Virtual Event, Switzerland.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3501385.3543972

1 INTRODUCTION

As the computer science education community continues to work towards the goal of CSforAll, it is important for us to research the most effective ways to scaffold students' learning, especially with younger learners. A common approach to introducing younger learners to computer science is through the use of block-based programming environments like Scratch [83], often by inviting them to author programs based on their own ideas and interests through the use of open-ended tasks [58]. While some students excel on such ill-defined tasks, there are some potential drawbacks to this approach. For example, some students may struggle with the open-ended, unconstrained nature of the task and have difficulty getting started on a project. Alternatively, students may envision programs beyond the capabilities of the tool or their abilities, or start out on a project that would require a significant amount of time and skill to implement. Further, such open-ended activities can be in tension with formal curricula with specific learning goals. While it is important to support and encourage individual creativity and agency as part of learning to program, it is equally important to provide scaffolds for struggling students, so they can have positive CS experiences. Drawing from other fields that have similar opportunities for open-ended activities, such as writing, this work explores how planning scaffolds may help address these challenges by supporting learners in thinking through what the program will do and how they will implement it.

In open-ended writing tasks, planning has been shown to be an important part of the pre-writing process and led to improved writing artifacts [7, 69]. In other subjects, such as reading [56], math

[33, 77], and science [34], planning scaffolds are often used as an instructional technique for young children. Across different subjects, planning scaffolds have shown to be effective for all students, and are especially helpful for neurodiverse learners [15, 34, 38] and English language learners [33, 35, 56]. CS education researchers have explored the use of planning scaffolds with high school [44] and undergraduate students [11, 70]; however, relatively little work has been done to investigate ways to scaffold younger learners' planning when learning to program.

Inspired by reading and writing graphic organizers, such as 5W (**W**ho, **W**hat, **W**here, **W**hen, **W**hy) Questions and Story Map [12, 33, 59], we developed a series of scaffolds in the form of planning sheets to help students articulate their vision for Scratch programs within a structure that targets an appropriate level of difficulty and relevant technical content. These types of graphic organizers are often used in elementary/primary (grades K-5, ages 5-11) classrooms, which allow us to harness teachers' and students' familiarity of using such scaffolds.

To understand the impact of using planning sheets, we incorporated them into the Scratch Encore Curriculum, a middle grade (ages 10-14, spanning the border between late primary and early secondary school) CS curriculum that uses the Use→Modify→Create pedagogical approach [41] to provide opportunities for both structured and open-ended programming challenges. In previous work [74], we investigated how students' used the planning sheets to structure their envisioned Scratch programs, finding high rates of planning sheet completion with all grade levels. However, differences occurred in planning sheet completion based on the format of the sheet (*physical* vs. *virtual*) and the teacher.

In this paper, we aim to extend this work by investigating how the student-authored planning sheets impacted the Scratch programs they wrote, specifically with respect to how closely the implemented projects adhere to the initial plans and to what degree the authored Scratch projects meet the assignment requirements. More concretely, this paper answers the following three research questions:

- *RQ1: After completing a planning sheet, how, and to what extent, do learners implement the Scratch projects they plan?*
- *RQ2: How, and to what extent, does using a planning sheet impact completeness of project requirements?*
- *RQ3: How do the factors of plan format, grade level, and teacher influence the implementation of plan and completion of project requirements?*

To better understand how planning can scaffold novice programmers as part of early programming instruction, we conducted a classroom-based study which included five teachers and 103 students who used planning sheets and one teacher with 52 students who did not use planning sheets prior to coding Scratch projects. Through a mixed-methods analysis of completed student planning sheets and Scratch projects, we investigated how often students implemented their planning aspects in their projects. Additionally, we used automated analysis to determine the percentage of requirements and extensions each Scratch project met, how complex the projects were (based on total blocks and unique blocks) and further analyzed the results based on whether the students used a planning sheet prior to coding. Finally, we investigated differences in **project implementation** and **requirements met** based on the format of the plan (physical vs. virtual), grade level, and teacher. This paper contributes to our knowledge of ways to support novices in learning to program through open-ended tasks and identifies the benefits of providing scaffolded planning materials to aid novices. In doing so, this work advances our understanding of planning as an instructional strategy to help young learners have positive early experiences in programming.

2 THEORETICAL ORIENTATION

In this work, we examine the role of external scaffolds as a means to help novices plan and implement Scratch programs, and in doing so, provide opportunities for early, positive experiences writing programs that include foundational computer science concepts and employ essential programming practices. To understand the role and importance of the planning sheets, we use two, complementary theoretical lenses. First, we use a distributed cognition lens to understand the ways that the planning sheets are supporting learners and the role the sheets play in structuring the cognitive exercise of authoring a program in response to an open-ended prompt [13, 29]. Second, the learning activities are a part of the Scratch Encore Curriculum, which was designed using a constructionist design approach.

2.1 Distributed Cognition

Distributed cognition posits that cognition is externalized and distributed through interactions of:

(1) An individual with other people
(2) An individual's tools and artifacts
(3) Earlier events with later events[29–31]

In this way, the theory challenges the notion that cognition resides solely inside an individual's head and instead provides a way to understand cognition that recognizes the essential roles that external artifacts (like tools and individuals) play in contributing to a cognitive task. Through examining the artifacts, effects of, and process of this distribution, we can gain insight into human cognition [30]. The individual, everyone, and everything the individual interacts with to complete a task, is part of a system of knowledge representations, both internal and external [32]. The external knowledge representation also helps individuals in "holding" knowledge that would normally be difficult to keep in an individual's mind. For example, in our context, students may struggle to remember: every sprite they plan to use, which events and actions they associate with each sprite, and what the project requirements are. The scaffolds we provide them, in this case, planning sheets, provide a means for learners to externally define these aspects of their project. By externally defining them, learners can refer to them and use them as an integral component of the cognitive task of authoring a program. In that way, the scaffolds also carry cognition; the way we designed the planning sheet and requirements list shape the way students view and complete the activities [53]. Because distributed cognition "offers access to the social, material, cultural, embodied, and mental richness of activity and learning,'; [13], researchers call on the learning community to reorient themselves to "facilitating individuals" responsive and novel uses of resources for creative and intelligent activity alone and in collaboration" [53].

In CS research, distributed cognition has been used in software engineering to analyze team interactions [66, 67] and how software engineering tools can support cognition [79]. Distributed cognition has also been used as a framework in the human-computer interaction field [25, 85], which includes an investigation into how students collaborate on and design tasks [78]. In computer science education research, distributed cognition is a fairly new theoretical framework [71]. Deitrick et al., used distributed cognition to analyze interactions in music programming tasks in a middle school classroom [13]. They analyzed student interactions with teachers, and tools, both *physical* and *virtual*. Knobelsdorf and Frede analyzed students' collaborative interactions while solving CS proofs through a distributed cognition lens [39]. To our knowledge, CS ed researchers have not yet analyzed material scaffolds, such as planning sheets, through a distributed cognition lens.

In our work, we draw on previous research in CS education and writing research to investigate how students externalize and distribute their ideas from the planning stage to the completed CS project. Using a distributed cognition theoretical lens helps justify our research focus on external resources (e.g., the planning sheets) as playing is a central role in the cognitive task of learning to program. Further, it helps structure how we attend to the specific roles the planning sheets play in the cognitive task and can help us identify the links between the knowledge stored on the planning sheets and how that knowledge is used for authoring the student Scratch projects.

2.2 Constructionism

The design of the Scratch Encore Curriculum was informed by the constructionist design approach [50, 52]. Constructionist design, grounded in Piaget's constructivist learning theory [55], emphasizes focusing on the powerful ideas of a discipline and foregrounds learning-by-doing, giving learners opportunities to construct personally meaningful artifacts [49]. Learning experiences designed through the constructionist lens prioritize self-directed learning and exploration, with the goal of giving the learner agency over their learning [37, 51]. Constructionism also emphasizes the idea of having the learners create artifacts that are public and shareable [37, 51]. As a result, the curriculum provides many opportunities for learners to modify existing and create new programs as a means for them to develop an understanding of the concept at hand while retaining agency to do so in a way that is consistent with their interests and identity. We will return to these ideas when we introduce the Scratch Encore Curriculum.

3 PRIOR WORK

In this section, we review three areas of relevant prior work: 1) planning in CS and outside CS, 2) students learning to program in Scratch, and 3) automated feedback for Scratch projects.

3.1 Teaching Planning

In this subsection, we cover relevant literature on teaching planning, starting with literature from CS education. Then, we cover literature on planning in subjects such as reading, writing, math, and science.

Many in the CS education community, including ourselves, have the goal of creating equitable CS learning experiences. One important step to take in reaching this goal is to consider and meet the needs of students who are neurodiverse and students who are English language learners (ELL). Planning is especially helpful for those who are neurodiverse [15, 34, 38] or ELL [33, 35, 56] but is also beneficial for all students. Although our paper does not focus on the above-mentioned populations of students (due to restrictions on the data we could collect), developing effective planning scaffolds will help us reach our goal.

Computer Science. In CS education, instructors and researchers have often focused on two ways of helping students plan: 1) various planning format/strategies, and 2) tools such as Intelligent Tutoring Systems (ITSs) to assist students in planning. Common planning formats include Unified Modeling Language (UML) diagrams, [2, 54, 70], flow charts [24, 46], and pseudocode [22, 57]. Storyboarding is also used in areas such as Human-Computer Interaction [72, 73]. Newer planning formats and strategies focus on supporting students in decomposing and chunking programming problems [11, 61]. Tools that have been developed to support planning include ITSs and software that provides automated feedback to users [2, 22, 26, 36, 47, 64]. Researchers have also developed tools using block-based programming environments [27, 81]. One such tool was a gallery of code examples integrated Snap! [81]. The researchers investigated how undergraduate students' use of code examples in an open-ended programming task correlated with their planning behavior. These researchers found that students with plans that were more complex (had more unique features) integrated more code examples into their projects.

We must also investigate how to best support young learners in planning their projects. Common CS planning formats such as UML diagrams are likely too complex for students aged 14 and younger. Of the work we reviewed, only three publications addressed teaching planning to K-12 students [44, 61, 74]. Much more recent work involved scaffolds developed for high school (secondary, ages 14-18) [44] and undergraduate [9] (post-secondary) students in game design. These scaffolds had students identify aspects of their projects such as the backstory, actors, important scenes, mechanics, player goal, and aesthetics. Work with middle grades (ages 10-14) include an exploratory analysis of young Scratch learners' use of planning scaffolds for their open-ended projects [74]. Similar to the works of Card et al. [9], and Milliken et al. [44], we also support students in identifying and recording visual (sprites/actors, setting) and functional (events, and actions/backstory) elements of their programs.

Outside of Computer Science. At the K-12 level, students often use planning and comprehension scaffolds in reading [12, 21, 56], writing [7, 14, 33], math [77, 80], and science [10, 34, 38, 60]. Planning and comprehension scaffolds take many forms, including graphic organizers [28]. Graphic organizers are visual displays that show the relationships between facts or concepts. They are often used in writing, reading, and science [15, 35, 56]. Types of graphic organizers include concept maps [34, 48], story-maps, and vee diagrams [4, 23]. Most relevant to our work are the 5Ws (Who, What, Where, When, Why) and story-mapping, which have been shown to improve student reading comprehension [12, 59] and writing [33, 75]. We drew upon these formats for many reasons:

1) they are used in K-12 schools in the U.S., and we know that they are appropriate for our age-group, 2) they are used in the U.S., the students are likely to be familiar with this process, and 3) the Create projects are open-ended and story-based, which makes the task somewhat similar to a writing task. Both formats scaffold students in identifying elements of their stories (characters/who, settings/where, actions/what, and other elements) and organizing the story plot.

3.2 Evaluating Learning in Scratch

With the rise in CS education in primary and secondary school [65, 76] many educators have a need for platforms with which to teach coding to students. As a free, online, block-based programming platform with millions of users that allows users to create open-ended and creative coding projects [42], Scratch has fulfilled that need for many educators. Learning in Scratch is often focused on how students gain knowledge of programming concepts by analyzing and evaluating the code that students create (e.g. [1, 84]). Observation and field notes [19, 43], peer evaluation [17, 82], and student interviews [3] are also used alongside student code as important sources when looking at how students gain programming knowledge. In response to this trend of evaluating learning in Scratch through student programs, Salac and Franklin explored the relationship between concepts implemented in code and students' responses in written assessments. This study found a weak correlation between coding concepts being present in students code, and students understanding it. It also suggests that using student programs may not be an effective measure of evaluating student learning and understanding when using Scratch [62].

3.3 Automatic Feedback for Scratch Projects

As the popularity of Scratch exploded in K-12 CS education, there have been several efforts aimed at analyzing student projects in order to provide meaningful feedback to both students and teachers. Moreno-León and Robles developed Dr. Scratch, a public facing web app that parses Scratch projects and generates a gamified report ranking how the project scored on several high level CS skills (Flow Control, Data representation, Parallelism, etc.) [45]. Similarly, Hairball developed by Boe et al is a "lint-like" tool for Scratch that is able to identify patterns such as infinite loops and unmatched broadcast/receive blocks in addition to providing a framework for extensible plugins [5]. Another tool worth mentioning is Scrape, originally developed by Burke and Kafai for use during a seven-week middle school Scratch workshop. Scrape is a visualization tool aimed at analysis of multiple Scratch projects [8]. The automated assessment tools developed by the Scratch Encore Curriculum build upon the work done by these projects, providing feedback on both an individual level to students and a classroom level to teachers via a web app, while also being extensible enough so that hundreds of projects can easily be evaluated at once.

4 CURRICULUM

Before detailing the methods for the study presented in this work, we first provide an overview of the curriculum and the planning sheets used to scaffold project requirements and story elements of Scratch projects.

This study took place in classrooms that used the Scratch Encore Curriculum. As a part of the curriculum, students are introduced to CS concepts (e.g. events, animation, conditional loops, etc.) and asked to program in the Scratch [58] environment. The Scratch Encore Curriculum includes 15 modules (one per CS concept), with each module taking between 3-5 lessons to complete. Lesson length varies from 60 - 120 minutes, which can be spilt into different class sessions if necessary.Each module employs the **Use→Modify→Create** [41] pedagogical approach where students are first introduced to the CS concept through a teacher-led mini-lesson or animated video, both of which engage students with real-life examples of the focal CS concept. Students then **use** the TIPP&SEE [63] strategy to observe an existing project through a "user" lens and make predictions and explore the code from a "programmer" lens. Using the same project, students will then **modify** the project by adding to or changing the code using the lesson's focal CS concept. Finally, students **create** open-ended Scratch projects that incorporate the focal CS concepts.

Students are supported throughout the create activity through scaffolds such as project idea prompts and planning sheets that are tailored to each module. In this work, we focus on the create projects for Module 2 (M2) and Module 3 (M3), where students learn about Events and Animation, respectively. Before each project, students complete a planning sheet. The planning sheet consists of two sections: 1) the project requirement task list, and 2) the 5W Questions. Due to COVID-19 virtual and hybrid learning, two versions of the planning sheets are available: paper *(physical)* for in-person and Google Form *(virtual)* for in-person and virtual learners.

4.1 Planning Sheets: Project Requirements Section

Included in the create planning sheets for all modules of the Scratch Encore Curriculum are lists of tasks necessary to complete the project, depicted on the top right side of Figure 1. Project requirements are based on the focal CS concept of the module. The project requirement task list reminds students to include aesthetic features (backdrops, sprites, and costumes) in their projects, and it identifies the coding requirements (blocks and scripts) needed to complete the project. For M2, the focal CS concept is user events, so the list includes a task requiring the use of different events in the project along with other coding and aesthetic tasks. M2 Project requirements are listed below:

- Choose a backdrop
- Include three sprites
- Use all three types of events (when green flag clicked, when sprite clicked, and when _ key pressed)
- Sprites say things and move
- At least one sprite changes size

For M3, the focus is on animating sprites both in place and across the screen, so the project requirement task list includes two tasks that check for animation. M3 requirements are listed below:

- Choose a backdrop
- Include three sprites
- At least two sprites animated in place (w/o movement)

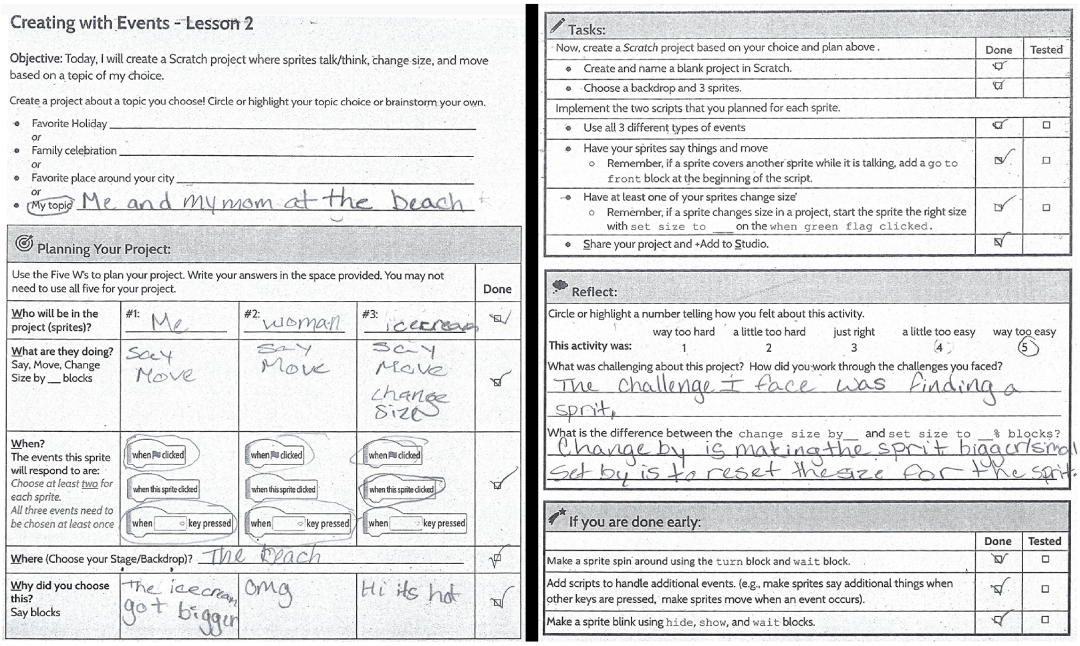


Figure 1: M2 Completed *Physical* Planning Sheet

- At least one sprite animated with movement

All planning sheets also include a task list of extension activities for students who have extra time after completing the project requirement tasks (e.g. For M2: making sprites spin/blink and including additional events; for M3: animating additional sprites and experimenting with different types of animation).

4.2 Planning Sheets: 5W Questions Section

Planning sheets also include a section designed to scaffold the storytelling nature of Scratch projects. This section includes elements of Story Map and 5W Questions graphic organizers that are often used for both reading comprehension and pre-writing [12, 33, 59] activities in elementary classrooms. The 5W questions and Story Map graphic organizers compare with our Create Planning Sheets for Scratch, as shown in Table 1.

5Ws	Story Map	Planning Sheets for Scratch Coding
Who?	Characters	Sprites
Where:	Setting	Backdrop
What?	Plot	Action blocks
When?	Plot (Beg/Mid/End) & Setting	User Events
Why?	Plot	Why did you choose this? Why is the sprite doing this?

Table 1: Comparing Graphic Organizers

We theorized that learners would be better supported learning something new (coding in Scratch) while using a familiar and relevant scaffold.

As Figure 1 illustrates, the *physical* create planning sheet includes a question about the *topic choice or theme* of the project, and the *5W questions* arranged in a grid for students to fill in. The grid provides cells for students to answer questions about the Sprites (characters) they plan to use (**W**ho?), actions the sprites will perform (**W**hat?), timing of the actions (**W**hen?), setting of the project (**W**here?), and reason(s) they chose the project theme, actions, events and/or sprites (**W**hy?). Sample answer (with Scratch related blocks) selections are provided for each of the questions to guide students in understanding the expectations. A checkbox is also provided to help students keep track of the tasks completed.

Figure 2 depicts a completed *Google Form (virtual)* version of the create planning form for M2. While most elements of the form are the same as the *physical* version, we also made changes based on teacher feedback (outlined in blue). For example, the "Why" question was located at the end of the *physical* planning sheet. Teachers stated that this was confusing because students did not understand what we wanted them to explain. To address this concern, we moved the "Why" question directly after the Project Choice in the *virtual* plan. Additionally, the *virtual* version asks students to identify the actions that will be completed for **each** sprite and event (also outlined in blue), whereas the *physical* version does not link actions to specific events.

5 METHODS

5.1 Study Context and Participants

In this IRB-approved study, we collected project planning sheets and Scratch projects for M2 and M3 from middle grade students

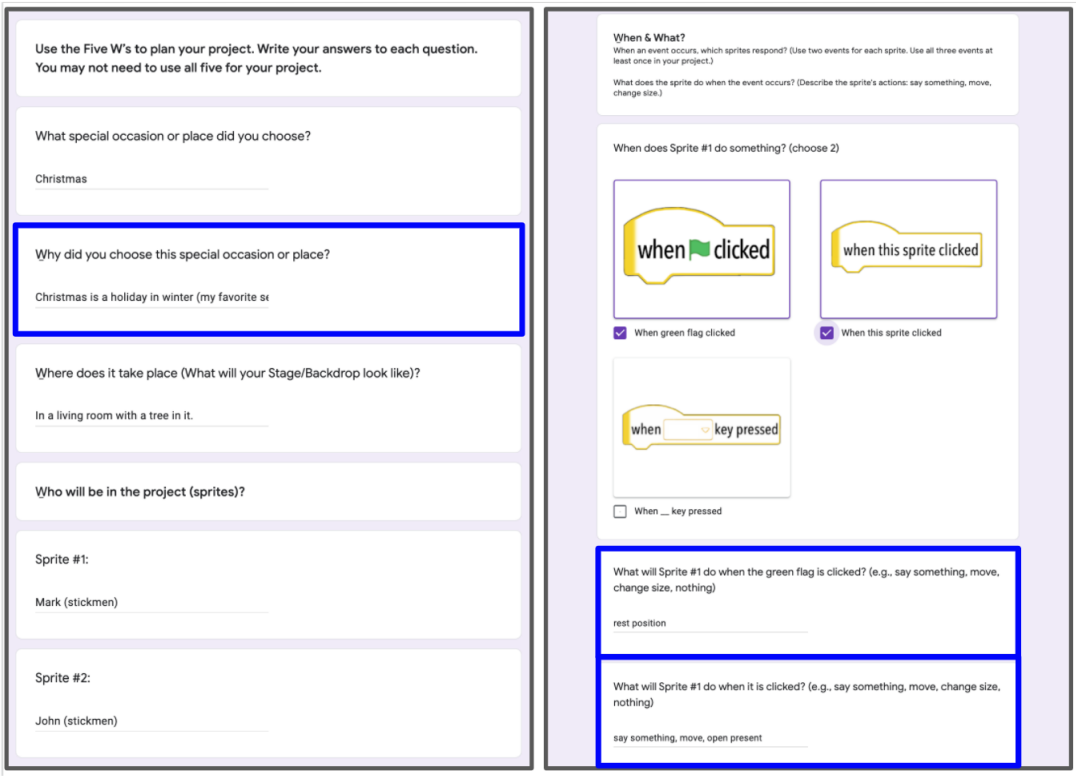


Figure 2: M2 Completed *Virtual* Planning Sheet

(ages 10 - 14) in a Midwestern, metropolitan school district. Our district collaborators invited Technology, Media, and Science teachers from their district to attend a professional development (PD) workshop for the Scratch Encore Curriculum conducted in-person in Year 1 (Y1: 2019-2020) and virtually in Year 2 (Y2: 2020-2021). Teachers were recruited for our study during the PD. The teachers and classrooms included in this paper are a subset of those participating in recurring multi-year studies with the large, metropolitan partner district.

As Table 2 depicts, in Y1 of the study we collected and analyzed *physical*, student planning sheets and Scratch projects from three teachers, three classes, and 58 students in grades five and seven. In Y2, we collected and analyzed *virtual*, student planning sheets and Scratch projects from three teachers, four classes, and 45 students in grades 5, 6, 7 and 8 (ages 10 – 14). Teacher experience with the Scratch Encore Curriculum ranged from 1 to 3 years for both Y1 and Y2. One teacher (teacher A) participated in the both years of this study. Table 2 also illustrates the breakdown of school demographic data.

Additionally, in Y2, we collected Scratch Project links from students who did **not** use a planning sheet (teacher F*), 52 projects for M2 and 49 projects for M3. During teacher interviews, teacher F* stated that this class did not use the planning sheet and that only the project requirements were posted on Google Classroom.

5.2 Data Collection and Analysis

In a previous study, we qualitatively coded each cell under the sections of the student planning sheets (Figure 3 and Figure 2) for completion to capture student thinking across the 5W's (Who, What When, Where and Why), marking both whether the cell was filled in as well as the contents of student responses [74]. The dataset consisted of 203 planning sheets for 202 Scratch projects from 103 students across 7 classrooms.

In this paper, we explore the relationship between student *Create* project planning sheets for M2 and M3, the implementation of student plan aspects within Scratch projects, and the project requirements that they met. We analyzed the original dataset with an additional 101 projects from 52 students who did not use the planning sheets, totaling 303 projects from 155 students across 8 classrooms. We used a mixed methods analysis. Our findings and analysis of the data will include two main categories: **project implementation** and **requirements met**.

For **project implementation**, we qualitatively coded the students' Scratch projects to quantify the extent to which they implemented their plans. The four researchers who had previously coded the content of the planning sheets reviewed each project and attempted to match the sprites present with those described in the "Who" portion of the planning materials; in some cases, such as when the names of the sprites had been changed to match those on the planning sheets, this process was trivial. However, in scenarios where the sprites could not be clearly distinguished by their

Year-Teacher	# of Students	Grade Level	Age Range	Asian	Black	HSP	White	Other
1-A	23	5	10-11	2.4%	60.7%	23%	9.6%	4.3%
1-B	18	5	10-11	0%	88.9%	5.6%	2.5%	3%
1-C	17	7	12-13	0%	5.9%	92.3%	0.6%	1.4%
2-A	14	5	10-11	0%	88.9%	5.6%	2.5%	3%
2-D	13	8	13-14	9.7%	1.2%	45.8%	37.5%	6%
2-E	18	6&7	11-13	76.1%	1.2%	16.3%	5.2%	<2%
2-F*	52	8	13-14	0.1%	0.1%	99.3%	0.1%	<1%

Table 2: Classroom and School Demographics (* denotes class where no plans were used)

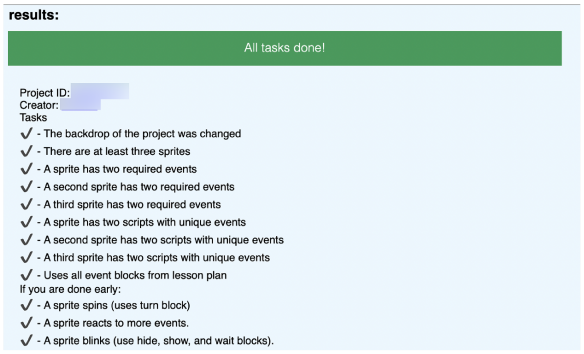


Figure 3: M2 Auto Grader Results

name or appearance, the researchers would compare the "What" section of the planning sheets and the implemented behavior of each sprite and attempt to use that as a basis to match the sprites. In scenarios where an objective mapping between implemented sprites and planned characters was impossible (either because the project implementation varied widely with the plan or because the project was incomplete) the researchers defaulted to sorting the sprites alphabetically.

The open-ended nature of the projects coupled with the fact that programming is an inherently creative process made the development of a coding scheme that reliably captured all the different ways in which a student might implement a planned behavior a surprisingly difficult task and required several iterations. We individually coded 20% of the documents for each pairing (e.g., M2Y1, M3Y1). We calculated the Interrater Reliability (IRR) using Cohen's Kappa (κ = 0.822-0.918, indicating almost perfect agreement [40]). The team discussed inconsistencies to reach 100% agreement. The remaining planning sheets were coded individually.

We coded each sprite, action, event, etc. that were planned as:

- "1" - *fully implemented* if the element found in the project matched the plan (e.g., (Who?) planned sprite was "Mom" and project contained a sprite named "Mom"),
- "P" - *partially implemented* part of the planned element existed in the project (e.g., (What?) planned actions were "walk" and "talk" and project included a say block but no movement blocks), or
- "0" - *not implemented* planned element was not included in the project (e.g., (When?) planned event "When Sprite Click", but the event was not in the project).

In our analysis, considered a planned item as "implemented" if they fell under the "1 - fully implemented" or "P - partially implemented" code. We did so because the "P" occurred infrequently (5.3%) and we aimed to account for student attempts at implementing their plans. 78.07% of the entries were fully implemented and 16.63% of the entries were not implemented.

For **requirements met**, student projects were run through our automatic assessment tool, which we will refer to as the autograder, that checked how many of the project requirements the project satisfied (output shown in Figure 3). The autograders work by first building an Abstract Syntax Tree (AST) out of the Scratch projects and then parsing it while looking for specific patterns. For instance, to check whether a sprite is "animated while moving" the autograder would traverse the AST and award points if it is able to pattern match on a loop of any kind that contains a move block. Both students and teachers had access to the autograder and were encouraged to use it to check their progress.

6 RESULTS

In this section, we begin by exploring the correlation between student plans and implementations. We then dive into the relationship between plans (or the lack thereof) and project requirement completion, including a case study as well as quantitative results on an individual level (across classrooms) and a class-level comparison. Finally, we explore the degree to which various characteristics (teacher, grade level, *virtual* vs. *physical*) affected the **project implementation complexity** and **requirements met**.

6.1 RQ1: How, and to what extent, do Students Implement their Plans?

We begin by breaking down the planning sheet into sections (aspects) and looking at the percentage of students who implement that section's plan.

Finding 1: Students implemented their planned Sprites (92%), Backgrounds (81.77%) and Project Choice (79.59%) more than the other planned sections in their Scratch Projects (e.g., Actions, Events).

A Kruskal-Wallis H test was conducted to determine if there were differences in student implementation of their plans in each section.

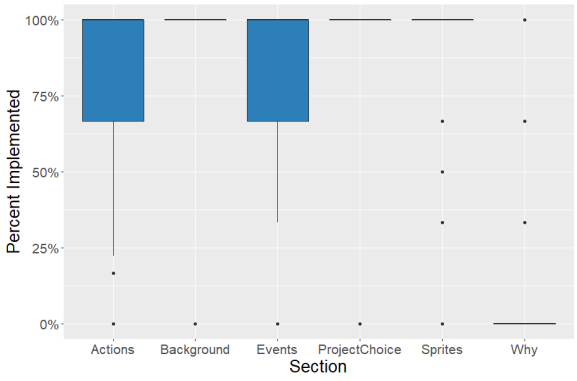


Figure 4: Project Implementation of Plan aspects

Distributions of project implementation were not similar for all sections (Figure 4). *The mean ranks of project implementation were statistically significantly different between sections, $\chi^2(3) = 480.639$, $p < 0.001$.*

Pairwise comparisons were performed using Dunn's [16] procedure with a Bonferroni correction for multiple comparisons. Adjusted p-values are presented. *This post hoc analysis revealed statistically significant differences in project implementation between: the "Why" and all other sections ($p < 0.001$ in all cases); the "Actions" section and the "Sprites", "Background", and "Project Choice" sections ($p < 0.001$ in all cases); and the "Events" section and the "Sprites", "Background", and "Project Choice" ($p = 0.003$ in all cases).*

Figure 4 depicts how different sections of the planning sheet were implemented in projects. For the "Sprites", "Background", and "Project Choice" sections, almost all (87%-90%) of the project implementations were at 100%. For the "Why" section, almost all (87%) the project implementations were at 0%. Actions and Events are more spread out, and fewer students implemented the Actions (68.99%) and Events (65.02%) overall. A possible reason that the Sprites, Backgrounds and Project Choice were implemented more often is that backgrounds and Sprites are fairly easy to include in Scratch projects, whereas, coding different Events and Actions may prove more challenging. Additionally, some students may have planned actions that were later found to be challenging to code (e.g., "kicking a soccer ball" which might include attending to costumes, motion and timing). We hypothesize that the "Why" aspect may not have been implemented due to student confusion about how and where to integrate this information in their Scratch stories.

6.2 RQ2: What is the Relationship between Planning Sheets and Project Requirements Completion?

For RQ2, we begin by exploring the relationship between completed planning sheets and project requirements on a case study basis and quantitative individual student basis. We then explore differences between completed projects (requirements met and project complexity) on a classroom basis, comparing classrooms that used planning sheets and those that did not.

Finding 2: Plans expressed on planning sheets do not always satisfy project requirements.

To answer RQ2, we first present a case study analyzing two students' completed planning sheets (Figure 1 and Figure 5) and the resulting Scratch projects (Figure 6 and Figure 7). The individual projects were selected as they are representative of two particularly interesting types of projects encountered by the authors: projects that are technically correct but aesthetically simple, and projects that fail to meet all of the requirements yet demonstrate substantial creative effort. In both instances, the students made use of the planning sheets and implemented all aspects of their plans. In one case, however, the student completed all of the project requirements (Figure 3), while in the other case the student completed only 4 out of 9 technical requirements (Figure 8).

Focusing first on the planning sheets, a significant distinction between the high-scoring and low-scoring projects is the level of detail of the planned actions. The high-scoring student followed the instructions of the planning sheet completely: they first listed out which action blocks each character would use, selected two events for each character, specified a backdrop, and then wrote out the dialog. The low-scoring student on the other hand did not fully follow the directions of the planning sheets: they did not list specific action blocks that would be used, they only used the "green flag" event for all sprites, and they left the dialog portion blank. There are several potential reasons for this. It is possible that the planning sheet proved to be too vague, leading to confusion regarding what is expected in each section. The fact that the switch from *physical* planning sheets to *virtual* planning sheets that make more explicit what type of input is expected in each section led to an overall increase in project completion rates would certainly support this. Similarly, since the authors of the high- and low-scoring projects belonged to different classrooms, it is possible the manner in which their teachers introduced the planning sheets and illustrated what was expected in each section could also lead to different project completion rates. This might also indicate that the high-scoring student was using the planning sheet as a cognitive artifact that assisted them in offloading the need to hold the entirety of their plan in memory, while the low-scoring student treated the planning sheets as an assignment to be completed rather than a design tool.

The most striking difference between the two projects from an aesthetic perspective is the use of custom-made sprites in the low-scoring project (Figure 7): it is evident that in the planning stage of the project, the student had a very specific list of characters (three celebrity chefs) that went beyond what is immediately available by default in Scratch. By contrast, the high-scoring project only utilizes assets that are immediately available within the Scratch editor. This implies that in the implementation phase, the student of the low-scoring project spent more time on finding their sprites than that of the other student. While using custom sprites allow for students to express their ideas, students may also need more help in managing their time.

Turning to the projects requirements part of the planning sheets, it is also interesting to note that in both instances the students appear to be aware of the extent to which their projects were satisfying the requirements. The author of the high-scoring project accurately indicated that they had completed all requirements and extensions, while the author of the low-scoring project seemed to recognize that their project was incomplete.

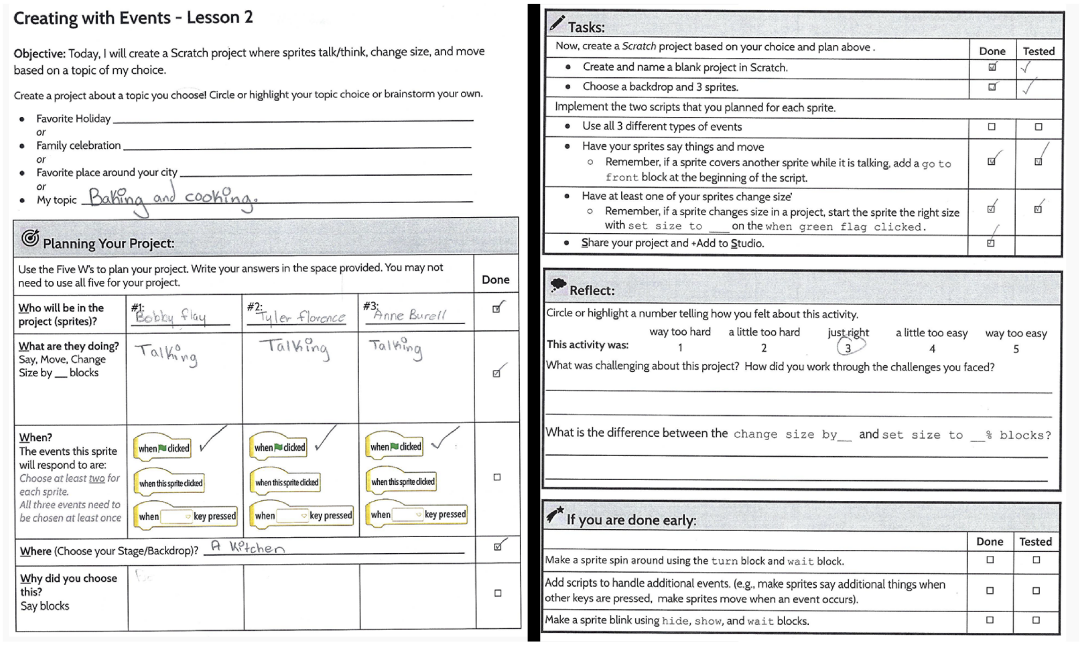


Figure 5: Planning sheets for a project that completely implemented the plan yet met few of the requirements

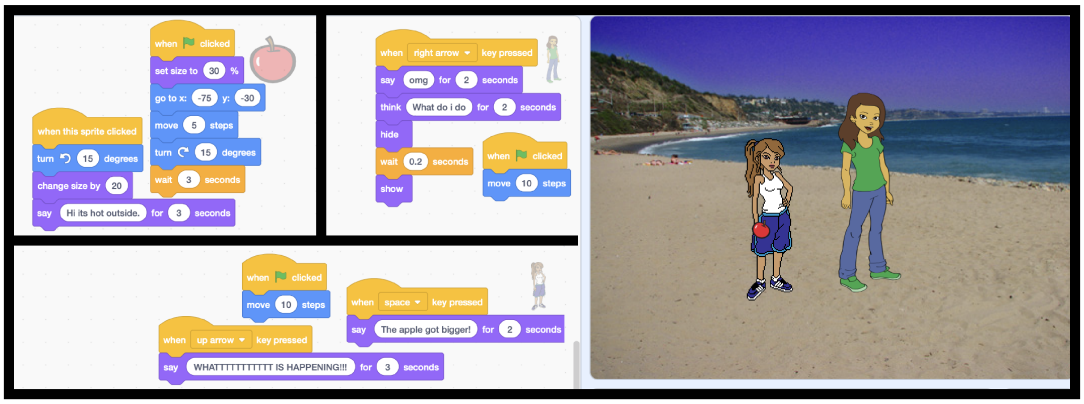


Figure 6: Implementation for High-Scoring Project

The low-scoring project is fairly representative of others encountered by the authors where students found/made custom assets leading to aesthetically complex projects without meeting the content requirements for the assignment. Additionally, we encountered several projects that satisfied the requirements with minimal aesthetic changes. This tension between allowing students enough creative control to make projects that are meaningful and intellectually stimulating while also providing sufficient constraints to ensure that the content of the curriculum is practiced lies at the core of the Scratch Encore Curriculum.

Having shown a detailed description of what students' implementation of their plans looked like for the Scratch Encore Curriculum, we now look across our full set of data to understand the students' use of their plans in their projects.

Finding 3: There was no correlation between the extent to which students implemented their plans and how well the resulting programs met the project requirements. Having investigated how, and to what extent, students implemented their plans, we sought to investigate the relationship between students meeting their requirements and implementing their plans. Figure 9 displays the scatterplot of the student project implementation (x-axis, in percentages) and project requirements and extensions met (y-axis, in percentages). As shown in the figure, the students' rate of project implementation of plan did not correlate with the students' rate of meeting the project requirements or extensions.

Finding 4: Students taught by the teacher that did not use planning documents completed significantly less of the project requirements.

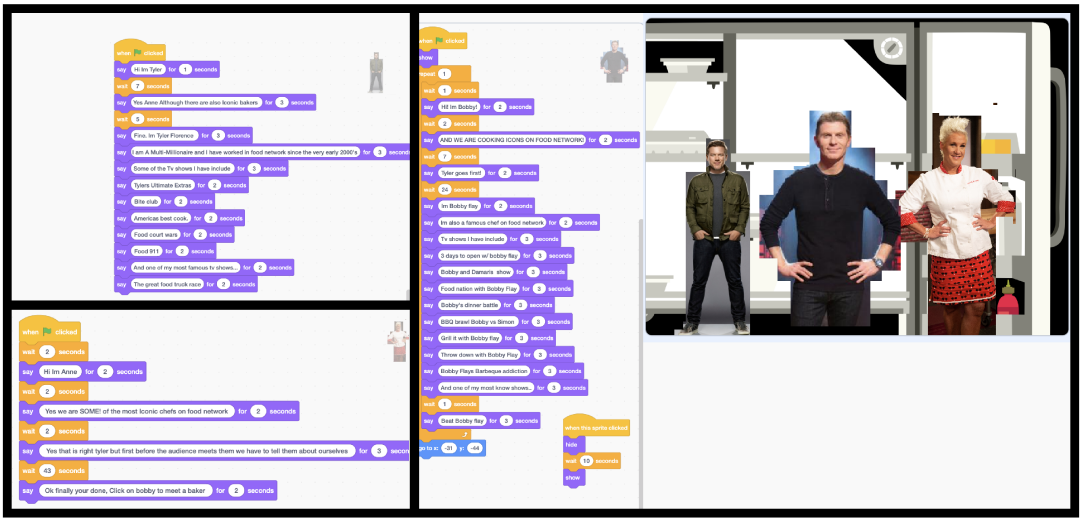


Figure 7: Implementation for Low-Scoring Project

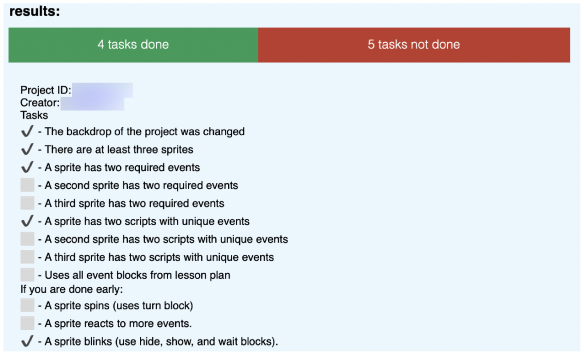


Figure 8: Auto Grader Feedback for Low-Scoring Project

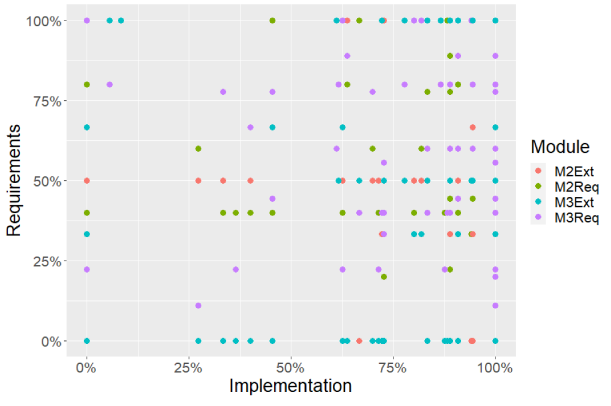


Figure 9: Scatterplot of Students' Requirements Met and Project Implementation

We conducted Mann-Whitney U tests to determine if there were differences in the students' completion of the project requirements and extensions based on whether they used a planning sheet. *There were statistically significant differences in all four cases (M2 Req: W = 916, p < 0.001 ; M2 Ext: W = 1089.5, p < 0.001; M3 Req: W = 1084.5, p < 0.001; M3 Ext: W = 2947, p = 0.014).* The differences are shown in Figure 10, which shows the percentage of no-plan and plan requirements and extensions completed. In the case of M2Req, M2Ext, and M3Req, the projects with plans had higher percentages of completion. And, while completion of extensions may seem relevant, the purpose of exploring this becomes clear when we see that, for M3Ext, the projects without plans had higher percentages of completion. This suggests that for M3Ext, the students without a plan worked on the extension tasks before completing the requirements. Extensions are designed to be creative ideas that may not correspond as closely with the material just taught, so the intent is for students to only pursue them once they have completed required elements.

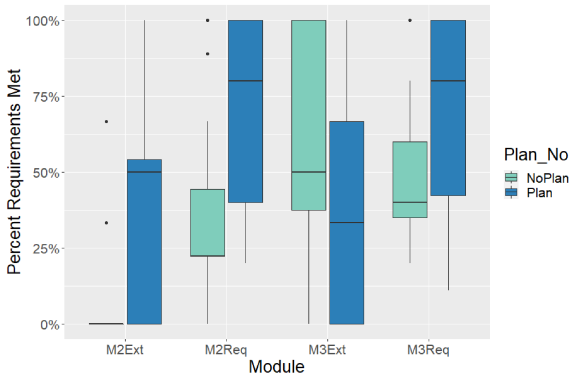
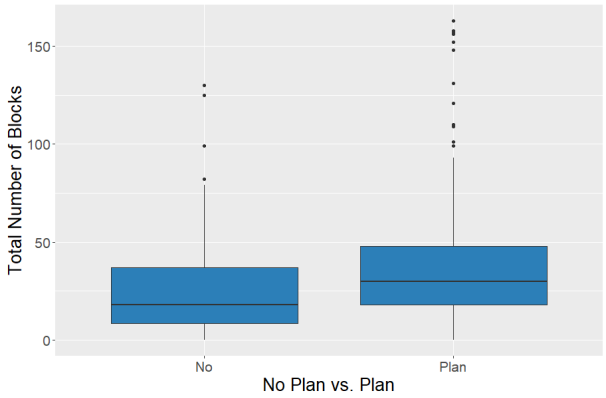
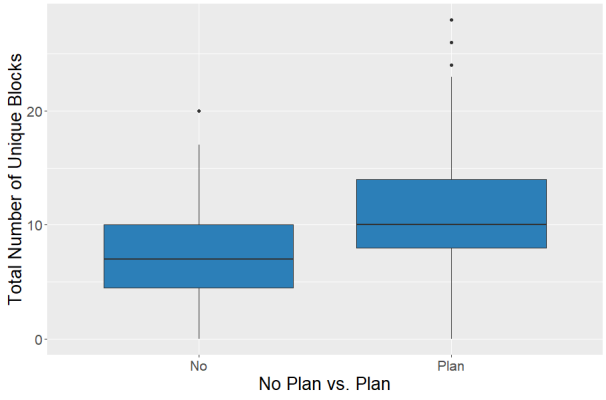


Figure 10: Completing Required Tasks with/without a Plan

Finding 5: Projects with a plan were more complex, using both more Scratch blocks and more types of Scratch blocks, than projects without a plan.



(a) Total Number of Blocks without and with Plans



(b) Total Number of Unique Blocks without and with Plans

Figure 11: Project Complexity of Projects without and with Plans

We ran Mann-Whitney U tests to determine whether the projects with a plan were more complex using two criteria: number of blocks (Figure 11a) and number of unique blocks (Figure 11b). *The median number of total blocks was statistically significantly higher for the plan (30 blocks) than the no plan (18 blocks) projects, U = 11711.500, z = 4.249, p < 0.001. Additionally, the median number of unique blocks was statistically significantly higher for programs based on a plan (10 blocks) than those without a plan (7 blocks), U = 12771.500, z = 5.897, p < 0.001.* This result suggests that the plans scaffolded students to use more blocks and more types of blocks. One potential explanation stems from the distributed cognition lens, which suggests that since the plan was able to "remember" the learners' intentions, the learners could focus on implementation while not needing to keep track of everything they needed to add to their projects.

6.3 RQ3: Other Factors Influencing Project Implementation and Requirements Met

Now we present the results of our final research question where we compare students' project implementation and requirements met based on the format of the planning sheets, grade-level, and teacher.

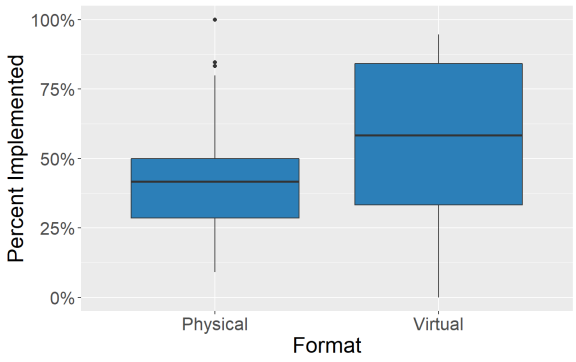


Figure 12: Project Implementation By Planning Sheet Format: Overall

Finding 6: Student project implementation and requirements met differed between planning sheet formats.

Project Implementation. A Mann-Whitney U test was run to determine if there were differences in project implementation percentage overall between *physical* and *virtual* planning sheets. Distributions of the implementation percentage for *physical* and *virtual* formats were not similar (Figure 12). *Project implementation percentage for physical (41.7%) were statistically significantly lower than for virtual (58.3%), U = 6845.5, z = 4.358, p < .001.*

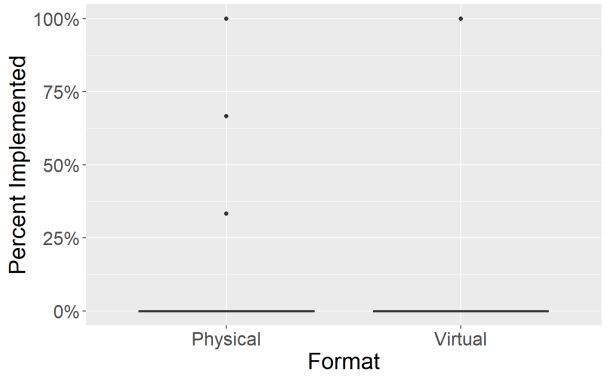
We also investigated the differences in project implementation percentage for each section between planning formats. We conducted a Mann-Whitney U test for each section. *We found that the differences between the formats were statistically significant for the "Why" and "Event" sections.*

For the "Why" section (Figure 13a), the median project implementation percentage for *physical* sheet were statistically significantly higher than for *virtual* version, U = 3739.5, z = -3.165, p = 0.002.

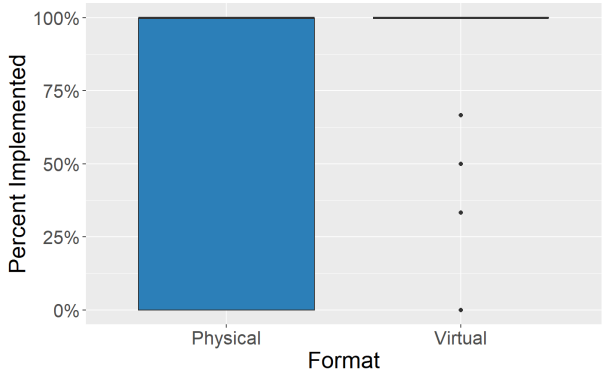
For the "Event" section, distributions of the project implementation percentage for planning formats were not similar (Figure 13b), with the project implementation percentage for *virtual* sheets being statistically significant higher (U = 6189.5, z = 4.423, p < 0.001).

As discussed in the case study, ambiguities in the *physical* versions of the planning sheets may have led to some students underutilizing the scaffolds. We believe that our shift to *virtual* planning sheets made necessary by COVID-19 pandemic inadvertently addressed this issue. Our edits to the planning sheets (explained in Section 4) made some prompts less open-ended while still allowing students to make choices. This includes the modification of the action section from asking "What is [the sprite] doing?" to asking the students to choose actions for each event for each sprite (e.g., "What does Sprite X do when the green flag is clicked?" and "What does Sprite X do when it is clicked"). This likely scaffolded the students to think a step further and made the implementation part of their projects easier.

Requirements Met. We conducted Mann-Whitney U tests to determine if there were differences in the students' completion of the project requirements and extensions based on the different planning formats. *There were statistically significant differences in three of the cases (M2 Ext: W = 3445, p < 0.001; M3 Req: W = 3140.5, p = 0.01; M3 Ext: W = 1084, p < 0.001).* The differences in M2 requirements



(a) Why Section



(b) Events Section

Figure 13: Project Implementation by Planning Sheet Format

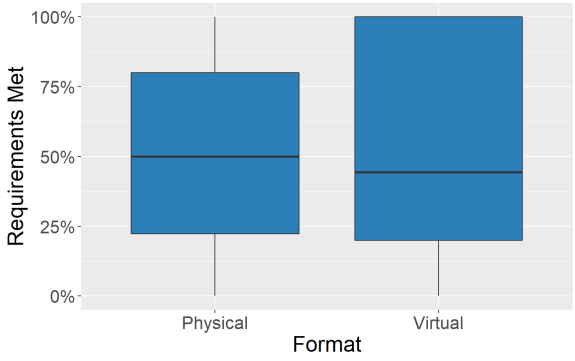
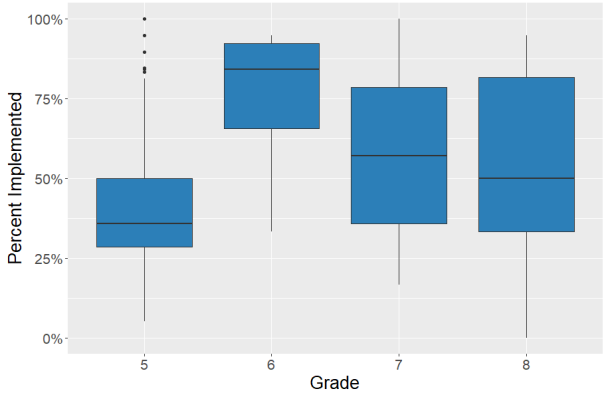
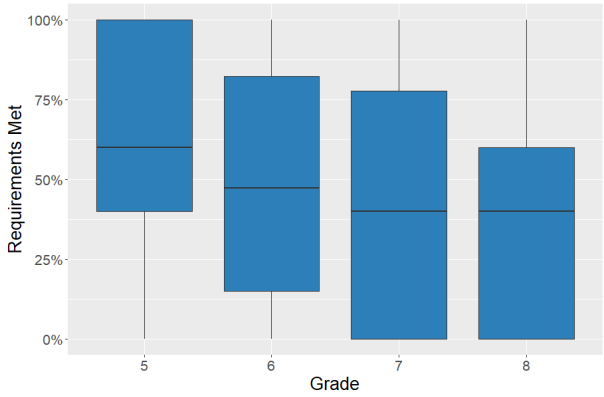


Figure 14: Requirements Met by Planning Sheet Format

met did not differ between the planning formats (W = 2723, p = 0.48). The differences are shown in Figure 14, which shows the percentage of requirements and extensions completed for the projects with each planning sheet format. Overall, the projects completed using the *physical* sheets had higher percentages of completion than the projects with *virtual* sheets. However, the results across the projects and requirements/extensions were not consistent. For M2Ext and



(a) Project implementation of Plan by Grade Level



(b) Requirements met Aspects by Grade Level

Figure 15: Grade Level Differences

M3Req, the *physical* planned projects (M2Ext median = 50%; M3Req median = 77.78%) were higher than the *virtual* planned projects (M2Ext median = 0%; M3Req median = 60%). The M3Ext completion for the *virtual* (median = 50%) plan projects were much higher than the *physical* plan (median = 0%) projects.

Finding 7: Student project implementation and requirements met differed across grade levels.

Project Implementation. Student plans were fully implemented approximately 75% of the time, regardless of grade level. A Kruskal-Wallis H test was conducted to determine if there were differences in student project implementation of their plans based on grade level. Distributions of project implementation were not similar for all grades (Figure 15a). *The mean ranks of project implementation were statistically significantly different between sections,* $\chi^2(3) = 28.879$, $p < 0.001$. Pairwise comparisons were performed using Dunn's [16] procedure with a Bonferroni correction for multiple comparisons. Adjusted p-values are presented. *This post hoc analysis revealed statistically significant differences in project implementation between grades 5 and 6 ($p < 0.001$), and grades 5 and 7 ($p < 0.001$).* No other statistically significant differences were found. The 6th grade students implemented their projects closest to their plans, while the 5th grade students implemented their projects furthest from their plans.

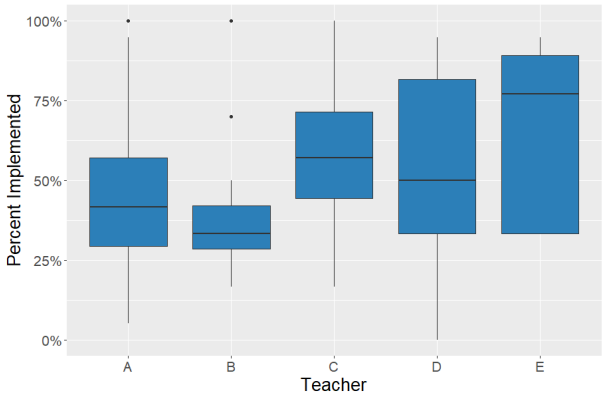
Requirements Met. A Kruskal-Wallis H test was conducted to determine if there were differences in student requirements met based on grade level. Distributions of requirements met were not similar for all grades (Figure 15b). *The mean ranks of requirements met were statistically significantly different between sections, $\chi^2(3) = 36.953$, $p < 0.001$.* Pairwise comparisons were performed using Dunn's [16] procedure with a Bonferroni correction for multiple comparisons. Adjusted p-values are presented. *This post hoc analysis revealed statistically significant differences in requirements met between grades 5 and 7 ($p < 0.001$), and grades 5 and 8 ($p < 0.001$).* No other statistically significant differences were found. The 5th grade students met the most requirements and extensions, while the 7th and 8th grade students met the least. The results on grade-level differences further supports our findings from earlier in this paper on the mismatch between the percentage of student project implementation and requirements met (Finding 6).

Finding 8: Student project implementation and requirements met differed by teacher.

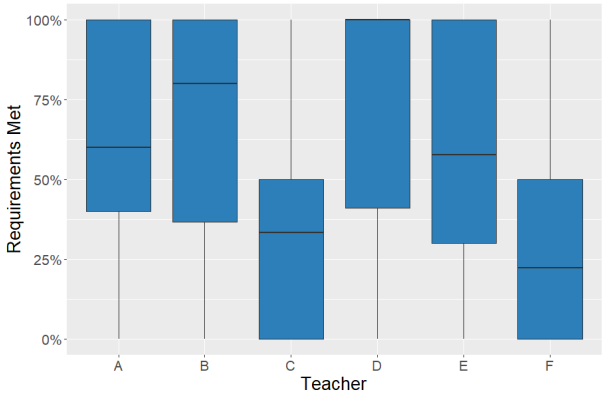
Project Implementation. A Kruskal-Wallis H test was conducted to determine if there were differences in student implementation of their plans based on teacher. Distributions of project implementation were not similar for all grades (Figure 16a). *The mean ranks of project implementation were statistically significantly different between sections, $\chi^2(3) = 33.082$, $p < 0.001$.* Pairwise comparisons were performed using Dunn's [16] procedure with a Bonferroni correction for multiple comparisons. Adjusted p-values are presented. *This post hoc analysis revealed statistically significant differences in project implementation between teachers B and D, C, and E ($p < 0.001$ for B-C, B-E; $p = 0.018$ for B-D), and teachers A and E ($p = 0.005$).* No other statistically significant differences were found. Teacher B's students had the lowest percentage of the project implementation matching the plan (median = 33.33%). Teacher E's students had the highest percentage of the projects matching the plans (median = 77%).

Requirements Met. A Kruskal-Wallis H test was conducted to determine if there were differences in student requirements met based on teacher. Distributions of requirements met were not similar for all teachers (Figure 16b). *The mean ranks of requirements met were statistically significantly different between sections, $\chi^2(3) = 88.168$, $p < 0.001$.* Pairwise comparisons were performed using Dunn's [16] procedure with a Bonferroni correction for multiple comparisons. Adjusted p-values are presented. *This post hoc analysis revealed statistically significant differences in requirements met between teachers C and A, B, D, and E ($p < 0.001$ for all), and teachers F and A, B, D, and E ($p < 0.001$ for all).* No other statistically significant differences were found. Teachers F and C's students had the lowest percentage of the requirements met matching the plan (medians = 22.22%, 33.33%). Teacher D's students had the highest percentage of the projects matching the plans (median = 100%).

It is interesting to note that similar to our findings about differences in project implementation and requirements met by grade level, the differences by teacher were not the same for the project implementation as they were for the project requirements met. This is another piece of evidence showing a lack of correlation between levels of project implementation and requirements met, a trend we will return to in the discussion.



(a) Project Implementation by Teacher



(b) Requirements Met by Teacher

Figure 16: Teacher Differences

7 DISCUSSION

Having presented the data and individual results related to each research question, we now take a cross-cutting look at the entire set of results and how it relates to the original research questions. In this study, we investigated how young programmers used their planning sheets in their project implementation and how the sheets related to the students meeting the project requirements. Through the lens of distributed cognition, we hypothesize that the act of writing down and organizing their thoughts in a scaffolded sheet helped the students create their Scratch projects. In what ways, however, did it help them? To plan, implement, both, or neither?

RQ1: After completing a planning sheet, how, and to what extent, do learners implement the Scratch projects they plan?

Our results show that students are faithful to their plans of Sprites, Backgrounds, and Project Choice. What is the distinction between these sections and the others (Events / Actions)? The straightforward answer is that students fairly faithfully implemented the aesthetic choices much more than the coding aspects. This could be because during the planning stage, students do not realize the complexity in programming some actions (e.g. kicking a ball), so they have high aspirations but must adjust due to programming difficulty. In addition, some mismanage their time, focusing on aesthetics rather than functionality (as seen in the case study student).

This raises two potential challenges that students face in all open-ended projects, well beyond computer science: planning projects of appropriate difficulty and managing their time in order to complete required technical elements.

RQ2: How, and to what extent, does using a planning sheet impact completeness of project requirements?

Exploring the relationship between planning sheets and corresponding Scratch projects requirements brought up several interesting issues. Due to the lack of correlation between the extent to which students implemented their plan and the extent to which they completed requirements, one might conclude that the planning documents were not useful. On the other hand, the data was clear that being in a classroom with the planning documents helped students meet project requirements as well as create more complex projects. Put another way, while, on an individualized level, a student's **project implementation** (i.e., how often their final programs included the features they had planned to include) does not correlate with **requirements met** (i.e., how well they accomplished the specific objects of the assignment), at the classroom level the act of planning consistently led to projects that better implemented the technical requirements as well as projects that were more complex (as measured by number of and distinct types of blocks used). Since only two classrooms (one teacher) did not to use planning documents, we do not know whether it is the planning documents themselves or the teacher's instruction that makes the difference.

To help resolve this conundrum, we use the existence of the planning sheet to disambiguate between student *plans* versus student *implementations*. Our case study showed an example of two students who fully implemented their plans, but one student's plan did not fulfill the requirements (that were specified on the planning sheet). Coupled with the finding that there was no correlation between the extent to which students implemented their plans and how well the resulting programs met the project requirements, this points to the possibility that shortcoming of the *plans*, not shortcomings of the *implementations*, is a critical factor in not meeting requirements.

This tension is at the heart of constructionist design. Our goal in Create projects is to give learners agency in customizing their projects and making them personally meaningful *while still using specific computing concepts*. In the 5W Questions portion of the planning sheet, we included specific directions to ensure that students' plans would meet the requirements. For example, the instructions in the physical version of M2 planning sheet for the "When" question included "Choose at least two [Events] for each sprite. All three events need to be chosen at least once." The students were to circle their chosen events for each sprite. However, we also find that some students still make plans that do not satisfy all of the requirements. Perhaps some requirements are too detailed for some students, or they do not pay close attention to the "small print". This could be further supported by a plan checker that points out when aspects of their plan do not satisfy the requirements. However, the planning documents led to both more complex projects and a higher requirement completion rate, so they have been helpful in obtaining our desired balance between technical coverage and creativity.

RQ3: How do the factors of plan format, grade level, and teacher influence the implementation plan and completion of project requirements?

Our results also suggest that some changes we made while shifting from the physical planning sheets to the virtual planning sheets may have inadvertently addressed some problems with helping students create plans that more closely met requirements. In particular, our changes made more explicit what type of answers were expected in each section. For example, the virtual version of the M2 planning sheet had students select their event choices using a checkbox rather than a textbox. This implies that there may be worthwhile gains in further improving the planning document and/or checking plans prior to implementation.

Our results also show that teacher-level differences played a large role in their students' outcomes, which is consistent with previous literature [20, 74]. While we do not currently have the data to fully understand the ways in which teacher differences affected their students' completed projects (such as recordings of the classes), we encountered evidence of teacher feedback on some physical planning sheets that we believe to be relevant. In the case of one instructor, this was merely a check mark at the top of the page. In other cases, it appeared that the teacher wrote specific suggestions. Figure 17 shows that at some point the students submitted their planning sheets for review. The teacher from the paper on the left correctly identified that the student had skipped the *why* section of the planning sheet yet provided no further instructions/suggestions on how to improve the project. On the right side of Figure 17, we see writing in a different color and handwriting saying "Change costume to sprites to animate" and "add text to tell viewer what to do." That the virtual planning sheets led to a significant increase in project completion was somewhat surprising to the authors, as there was concern that teachers would no longer be able to spot-check and provide feedback in the same immediate way as they were able to on the physical versions.

7.1 A Holistic View of Designing for Computing Education

Another contribution of this work is to broaden the conversation around the design of introductory programming environments. Historically, a significant portion of the design-related discussion related to introducing novices to programming has focused on the design of the technical programming environment (e.g., the design of programming environments [42], the role of block-based programming [83], features of programming languages [68]) while less emphasis has attended to scaffolds that live outside the programming environment itself. While there is a growing body of research evaluating various aspects of and pedagogical strategies (e.g., [41], TIPP&SEE [63]) and introductory curricula [6, 18], this work adds yet another consideration to designing for novices - that of scaffolds that live alongside programming environments in support of the act of programming. The distributed cognition lens is particularly useful for this larger framing, as it now extends the cognitive system beyond the individual and the programming environment to include a fuller set of scaffolds that can contribute to the cognitive task of writing a program. In doing so, this work seeks to open the door to further scholarship investigating the role

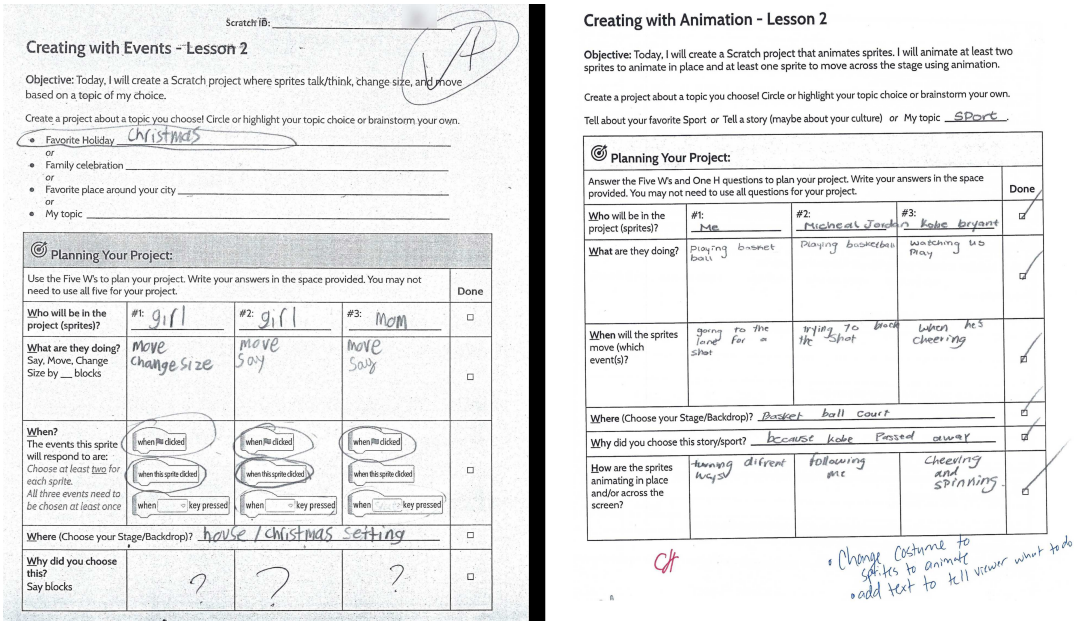


Figure 17: Feedback from two different teachers.

that designed artifacts beyond the programming environment itself play in helping novices learn to program.

7.2 Relevance and Future Work

By analyzing the impact of planning sheets on completed programming project, we believe we have identified compelling evidence that adequate planning scaffolds can play a beneficial role in early CS education: scaffolds such as our planning sheets allow students creative control and ownership over their own ideas while at the same time directing towards projects that reinforce the technical concepts they are learning.

We have also identified several areas for future work on the topic. The discrepancies in student performance for virtual v.s. physical planning documents was a particularly interesting finding for the authors, however, since the switch from virtual to physical was driven primarily by the urgent need of curriculum materials that could be taught during the emergency period of remote learning instigated by the COVID-19 pandemic, a more rigorous effort to study the specific affordances being exploited by high performing students in each condition is required.

Similarly, the discovery of student projects that demonstrate high levels of technical and creative competence yet meet only a portion of the techincal requirements suggessts that the planning scaffolds provided might need further refinements: are low scoring students unaware of what the technical requirements are? Of how to implement? Or perhaps the current scaffolds are inadvertently driving some students to plan projects that are too complex to be implemented within the constraints of the Scratch environment.

7.3 Limitations

While we think the study design and data collected positioned us well to make claims about the role of planning sheets in supporting novice programmers, it is not without its limitations. For example, the number of students and teachers who participated in the study was relatively small. This may introduce some confounds, especially when the full set of participants was subdivided (such as the plan/no-plan analysis). Additionally, one of the five teachers participated in the study both years and was therefore more familiar with the curriculum than some other teachers. However, this reflects the conditions of a real school environment, where teachers will have varying levels of experience and skills. Second, as with all qualitative studies, there is a potential of researcher bias. We worked together to minimize those biases through discussion. Finally, some teachers were teaching Module 3 around February/March 2020 and the COVID-19 pandemic may have affected the students' work. The pandemic and the switch to digital planning sheets likely also affected their work: some students might have experienced substantially different work environments during the periods of remote learning, the amount/quality of teacher feedback provided could also have been different to what students before the pandemic received, etc.

Another set of potential limitations is in how teacher decisions shaped the structure of the data we analyzed. For example, the division of classrooms into plan/no plan groups was incidental rather than an intervention: one of the teachers who participated in the study decided to have their students start programming without using the planning sheets and using only the project requirements as a guide. As such, it is impossible to conclusively separate the effects of planning with confounding variables such as teacher effects.

On a more technical level, while the automated assessment tools were designed to be fairly robust, it is possible for a student project to contain code that satisfies the spirit of a requirement, yet is accomplished in an unconventional way so the autograder fails

to recognize the behavior as correct or valid. Cases such as these are extremely rare in student code, as the overwhelming majority of projects follow the conventions taught in the curriculum, specifically those of the module being tested. Furthermore, the auto graders were custom made for each specific module and student projects were spot checked and compared with the autograder output as part of this analysis.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1738758.

REFERENCES

[1] Efthimia Aivaloglou and Felienne Hermans. 2016. How kids code and how we know: An exploratory study on the Scratch repository. In *Proceedings of the 2016 ACM conference on international computing education research*. 53–61.
[2] Sohail Alhazmi, Charles Thevathayan, and Margaret Hamilton. 2021. Learning UML sequence diagrams with a new constructivist pedagogical tool: SD4ED. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 893–899.
[3] Yasemin Allsop. 2019. Assessing computational thinking process using a multiple evaluation approach. *International journal of child-computer interaction* 19 (2019), 30–55.
[4] Marino C Alvarez and Victoria J Risko. 2007. The use of vee diagrams with third graders as a metacognitive tool for learning science concepts. (2007).
[5] Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Franklin Diana. 2015. Hairball: lint-inspired static analysis of scratch projects. In *Proceedings of the Workshop in Primary and Secondary Computing Education*. 132–133.
[6] K Brennan, M Chung, and J Hawson. [n. d.]. Creative computing: A design-based introduction to computational thinking. *https://creativecomputing.gse.harvard.edu/guide/* ([n. d.]).
[7] Marjorie Brown. 2011. Effects of Graphic Organizers on Student Achievement in the Writing Process. *Online Submission* (2011).
[8] Quinn Burke and Yasmin B Kafai. 2012. The writers' workshop for youth programmers: digital storytelling with scratch in middle school classrooms. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. 433–438.
[9] Alexander Card, Wengran Wang, Chris Martens, and Thomas Price. 2021. Scaffolding Game Design: Towards Tool Support for Planning Open-Ended Projects in an Introductory Game Design Class. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–5.
[10] Christina R Carnahan, Pamela Williamson, Nicole Birri, Christopher Swoboda, and Kate K Snyder. 2016. Increasing comprehension of expository science text for students with autism spectrum disorder. *Focus on Autism and Other Developmental Disabilities* 31, 3 (2016), 208–220.
[11] Umberto Costantini, Violetta Lonati, and Anna Morpurgo. 2020. How plans occur in novices' programs: A method to evaluate program-writing skills. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 852–858.
[12] Tim Crabtree, Sheila R Alber-Morgan, and Moira Konrad. 2010. The effects of self-monitoring of story elements on the reading comprehension of high school seniors with learning disabilities. *Education and Treatment of Children* (2010), 187–203.
[13] Elise Deitrick, R Benjamin Shapiro, Matthew P Ahrens, Rebecca Fiebrink, Paul D Lehrman, and Saad Farooq. 2015. Using distributed cognition theory to analyze collaborative computer science learning. In *Proceedings of the eleventh annual international conference on international computing education research*. 51–60.
[14] Laura Nicole Delrose. 2011. Investigating the use of graphic organizers for writing. (2011).
[15] Douglas D Dexter and Charles A Hughes. 2011. Graphic organizers and students with learning disabilities: A meta-analysis. *Learning Disability Quarterly* 34, 1 (2011), 51–72.
[16] Olive Jean Dunn. 1964. Multiple comparisons using rank sums. *Technometrics* 6, 3 (1964), 241–252.
[17] Hatice Yildiz Durak and Tolga Guyer. 2019. Programming with Scratch in primary school, indicators related to effectiveness of education process and analysis of these indicators in terms of various variables. *Gifted Education International* 35, 3 (2019), 237–258.
[18] Diana Franklin, Phillip Conrad, Gerardo Aldana, and Sarah Hough. 2011. Animal tlatoque: attracting middle school students to computing through culturally-relevant themes. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. 453–458.
[19] Diana Franklin, Phillip Conrad, Bryce Boe, Katy Nilsen, Charlotte Hill, Michelle Len, Greg Dreschler, Gerardo Aldana, Paulo Almeida-Tanaka, Brynn Kiefer, et al. 2013. Assessment of computer science learning in a scratch-based outreach program. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 371–376.
[20] Diana Franklin, Jean Salac, Zachary Crenshaw, Saranya Turimella, Zipporah Klain, Marco Anaya, and Cathy Thomas. 2020. Exploring student behavior using the TIPP&SEE learning strategy. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. 91–101.
[21] Linda B Gambrell and Ann Dromsky. 2000. Fostering reading comprehension. *Beginning reading and writing* (2000), 143–153.
[22] Stuart Garner. 2007. A program design tool to help novices learn programming. *ICT: Providing choices for learners and learning* (2007), 321–324.
[23] D Bob Gowin. 1981. *Educating*. Cornell University Press.
[24] Dee Gudmundsen, Lisa Olivieri, and Namita Sarawagi. 2011. Using visual logic©: three different approaches in different courses-general education, CS0, and CS1. *J. Comput. Sci. Coll* 26, 6 (2011), 23–29.
[25] James Hollan, Edwin Hutchins, and David Kirsh. 2000. Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Transactions on Computer-Human Interaction (TOCHI)* 7, 2 (2000), 174–196.
[26] Danial Hooshyar, Rodina Binti Ahmad, Moslem Yousefi, Farrah Dina Yusop, and S-J Horng. 2015. A flowchart-based intelligent tutoring system for improving problem-solving skills of novice programmers. *Journal of Computer Assisted Learning* 31, 4 (2015), 345–361.
[27] Minjie Hu, Michael Winikoff, and Stephen Cranefield. 2012. Teaching novice programming using goals and plans in a visual notation. In *Proceedings of the Fourteenth Australasian Computing Education Conference-Volume 123*. 43–52.
[28] Charles A Hughes, Paula Maccini, and Joseph Calvin Gagnon. 2003. Interventions that positively impact the performance of students with learning disabilities in secondary general education classrooms. (2003).
[29] Edwin Hutchins. 1995. *Cognition in the Wild*. MIT press.
[30] Edwin Hutchins. 2000. Distributed cognition. *International Encyclopedia of the Social and Behavioral Sciences. Elsevier Science* 138 (2000).
[31] Edwin Hutchins. 2006. The distributed cognition perspective on human interaction. *Roots of human sociality: Culture, cognition and interaction* 1 (2006), 375.
[32] Edwin Hutchins and Tove Klausen. 1996. Distributed cognition in an airline cockpit. *Cognition and communication at work* (1996), 15–34.
[33] Salem Saleh Khalaf Ibnian. 2010. The Effect of Using the Story-Mapping Technique on Developing Tenth Grade Students' Short Story Writing Skills in EFL. *English Language Teaching* 3, 4 (2010), 181–194.
[34] Elizabeth M Jackson and Mary Frances Hanline. 2020. Using a concept map with RECALL to Increase the comprehension of science texts for children with autism. *Focus on Autism and Other Developmental Disabilities* 35, 2 (2020), 90–100.
[35] Xiangying Jiang and William Grabe. 2007. Graphic organizers in reading instruction: Research findings and issues. (2007).
[36] Wei Jin, Albert Corbett, Will Lloyd, Lewis Baumstark, and Christine Rolka. 2014. Evaluation of guided-planning and assisted-coding with task relevant dynamic hinting. In *International Conference on Intelligent Tutoring Systems*. Springer, 318–328.
[37] Yasmin B Kafai and Mitchel Resnick. 2012. *Constructionism in practice: Designing, thinking, and learning in a digital world*. Routledge.
[38] Victoria F Knight, Fred Spooner, Diane M Browder, Bethany R Smith, and Charles L Wood. 2013. Using systematic instruction and graphic organizers to teach science concepts to students with autism spectrum disorders and intellectual disability. *Focus on autism and other developmental disabilities* 28, 2 (2013), 115–126.
[39] Maria Knobelsdorf and Christiane Frede. 2016. Analyzing student practices in theory of computation in light of distributed cognition theory. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. 73–81.
[40] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.
[41] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *Acm Inroads* 2, 1 (2011), 32–37.
[42] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.
[43] John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by choice: urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*. 367–371.
[44] Alexandra Milliken, Wengran Wang, Veronica Cateté, Sarah Martin, Neeloy Gomes, Yihuan Dong, Rachel Harred, Amy Isvik, Tiffany Barnes, Thomas Price, et al. 2021. 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*. 232–238.
[45] Jesús Moreno-León and Gregorio Robles. 2015. Dr. Scratch: a Web Tool to Automatically Evaluate Scratch Projects. In *Proceedings of the Workshop in Primary*

and Secondary Computing Education. 132–133.
[46] Isaac Nassi and Ben Shneiderman. 1973. Flowchart techniques for structured programming. *ACM Sigplan Notices* 8, 8 (1973), 12–26.
[47] Mark J Nelson and Michael Mateas. 2008. An interactive game-design assistant. In *Proceedings of the 13th international conference on Intelligent user interfaces*. 90–98.
[48] Joseph D Novak. 1990. Concept maps and Vee diagrams: Two metacognitive tools to facilitate meaningful learning. *Instructional science* 19, 1 (1990), 29–52.
[49] Seymour Papert. 1980. " Mindstorms" Children. *Computers and powerful ideas* (1980).
[50] S. Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc.
[51] Seymour Papert. 1993. *The children's machine: Rethinking school in the age of the computer*. ERIC.
[52] Seymour Papert and Idit Harel. 1991. Situating constructionism. *Constructionism* 36, 2 (1991), 1–11.
[53] Roy D Pea. 1993. Practices of distributed intelligence and designs for education. *Distributed cognitions: Psychological and educational considerations* 11 (1993), 47–87.
[54] Marian Petre. 2013. UML in practice. In *2013 35th international conference on software engineering (icse)*. IEEE, 722–731.
[55] Jean Piaget and Margaret Trans Cook. 1952. The origins of intelligence in children. (1952).
[56] Sam D Praveen and Premalatha Rajan. 2013. Using Graphic Organizers to Improve Reading Comprehension Skills for the Middle School ESL Students. *English language teaching* 6, 2 (2013), 155–170.
[57] Haider Ali Ramadhan. 2000. Programming by discovery. *Journal of Computer Assisted Learning* 16, 1 (2000), 83–93.
[58] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.
[59] D Ray Reutzel. 1985. Story maps improve comprehension. *The Reading Teacher* 38, 4 (1985), 400–404.
[60] Veronica Roberts and Richard Joiner. 2007. Investigating the efficacy of concept mapping with pupils with autistic spectrum disorder. *British Journal of Special Education* 34, 3 (2007), 127–135.
[61] Jean Salac. 2020. Diagramming as a Strategy for Primary/Elementary-Age Program Comprehension. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. 322–323.
[62] Jean Salac and Diana Franklin. 2020. 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*. 473–479.
[63] Jean Salac, Cathy Thomas, Chloe Butler, Ashley Sanchez, and Diana Franklin. 2020. 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*. 79–85.
[64] Joachim Schramm, Sven Strickroth, Nguyen-Thinh Le, and Niels Pinkwart. 2012. Teaching UML skills to novice programmers using a sample solution based intelligent tutoring system. In *Twenty-Fifth International FLAIRS Conference*.
[65] Sue Sentance, Erik Barendsen, and Carsten Schulte. 2018. *Computer Science Education: Perspectives on Teaching and Learning in School*. Bloomsbury Publishing.
[66] Helen Sharp, Rosalba Giuffrida, and Grigori Melnik. 2012. Information flow within a dispersed agile team: a distributed cognition perspective. In *International Conference on Agile Software Development*. Springer, 62–76.
[67] Helen Sharp, Hugh Robinson, Judith Segal, and Dominic Furniss. 2006. The Role of Story Cards and the Wall in XP teams: a distributed cognition perspective. In *AGILE 2006 (AGILE'06)*. IEEE, 11–pp.
[68] Andreas Stefik and Susanna Siebert. 2013. An empirical investigation into programming language syntax. *ACM Transactions on Computing Education (TOCE)* 13, 4 (2013), 1–40.
[69] Nicole Strangman, T Hall, and A Meyer. 2003. Graphic organizers and implications for universal design for learning: Curriculum enhancement report. *National Center on Accessing the General Curriculum* (2003).
[70] Michael Striewe and Michael Goedicke. 2014. Automated assessment of UML activity diagrams. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. 336–336.
[71] Josh Tenenberg and Maria Knobelsdorf. 2014. Out of our minds: a review of sociocultural cognition theory. *Computer Science Education* 24, 1 (2014), 1–24.
[72] Jakita O Thomas. 2018. The Computational Algorithmic Thinking (CAT) Capability Flow: An Approach to Articulating CAT Capabilities over Time in African-American Middle-school Girls. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 149–154.
[73] Khai N Truong, Gillian R Hayes, and Gregory D Abowd. 2006. Storyboarding: an empirical determination of best practices and effective guidelines. In *Proceedings of the 6th conference on Designing Interactive systems*. 12–21.
[74] Jennifer Tsan, Donna Eatinger, Alex Pugnali, David Gonzalez-Maldonado, Diana Franklin, and David Weintrop. 2022. Scaffolding Young Learners' Open-Ended Programming Projects with Planning Sheets. In *Proceedings of the 2022 ACM Conference on Innovation and Technology in Computer Science Education*. in press.
[75] Kayo Tsuji. 2017. Implementation of the Writing Activity Focusing on 5W1H Questions: An Approach to Improving Student Writing Performance. *LET Journal of Central Japan* 28 (2017), 1–12.
[76] Jan Vahrenhold, Quintin Cutts, and Katrina Falkner. 2019. *Schools (K–12)*. Cambridge University Press, 547–583. https://doi.org/10.1017/9781108654555.019
[77] Delinda van Garderen and Amy M Scheuermann. 2015. Diagramming word problems: A strategic approach for instruction. *Intervention in School and Clinic* 50, 5 (2015), 282–290.
[78] Christina Vasiliou, Andri Ioannou, Agni Stylianou-Georgiou, and Panayiotis Zaphiris. 2017. A glance into social and evolutionary aspects of an artifact ecology for collaborative learning through the lens of distributed cognition. *International Journal of Human–Computer Interaction* 33, 8 (2017), 642–654.
[79] Andrew Walenstein. 2002. *Cognitive support in software engineering tools: A distributed cognition framework*. Ph.D. Dissertation. Citeseer.
[80] David W Walker and James A Poteet. 1990. A Comparison of Two Methods of Teaching Mathematics Story Problem-Solving with Learning Disabled Students.. In *National Forum of Special Education Journal*, Vol. 1. ERIC, 44–51.
[81] Wengran Wang, Audrey Le Meur, Mahesh Bobbadi, Bita Akram, Tiffany Barnes, Chris Martens, and Thomas Price. 2022. Exploring Design Choices to Support Novices' Example Use During Creative Open-Ended Programming. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*. 619–625.
[82] Xiao-Ming Wang, Gwo-Jen Hwang, Zi-Yun Liang, and Hsiu-Ying Wang. 2017. Enhancing students' computer programming performances, critical thinking awareness and attitudes towards programming: An online peer-assessment attempt. *Journal of Educational Technology & Society* 20, 4 (2017), 58–68.
[83] David Weintrop. 2019. Block-based programming in computer science education. *Commun. ACM* 62, 8 (2019), 22–25.
[84] David Weintrop, Alexandria K Hansen, Danielle B Harlow, and Diana Franklin. 2018. Starting from Scratch: Outcomes of early computer science learning experiences and implications for what comes next. In *Proceedings of the 2018 ACM conference on international computing education research*. 142–150.
[85] Peter C Wright, Robert E Fields, and Michael D Harrison. 2000. Analyzing human-computer interaction as distributed cognition: the resources model. *Human-Computer Interaction* 15, 1 (2000), 1–41.