If They Build It, Will They Understand It?

Exploring the Relationship between Student Code and Performance

Jean Salac & Diana Franklin
University of Chicago, Chicago, IL, USA
{salac,dmfranklin}@uchicago.edu

ABSTRACT

The computer science community has struggled to assess student learning via Scratch programming at the primary school level (ages 7-12). Prior work has relied most heavily on artifact (student code/projects) analysis, with some attempts at one-on-one interviews and written assessments. In this paper, we explore the relationship between artifact analysis and written assessments.

Through this study of a large-scale introductory computing implementation, we found that for students who had code in their projects, student performance on specific questions on the written assessments is only very weakly correlated to specific attributes of final projects typically used in artifact analysis as well as attributes we use to define candidate code ($r < 0.2$, $p < 0.05$). In particular, the correlation is not nearly strong enough to serve as a proxy for understanding.

KEYWORDS

primary education, assessment, artifact analysis, Scratch

ACM Reference Format:
Jean Salac & 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 (ITiCSE '20), June 15–19, 2020, Trondheim, Norway.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341525.3387379

1 INTRODUCTION

Many countries worldwide are implementing initiatives to integrate Computer Science (CS) and Computational Thinking (CT) instruction at the primary and secondary level, such as Israel, India, New Zealand, the United Kingdom, and the United States [7, 13]. Moving from the after-school, optional domain into the formal school classroom increases the pressure to develop accurate assessment techniques that match the pedagogical approaches and tools used for this age group.

A popular programming language and development environment used in primary schools is Scratch [8]. Three assessment techniques are common in this realm: analyzing programs students create (artifact analysis), giving written assessments, and interviewing students. Interviews are the most accurate measure because of the ability to ask follow-up questions, but they are prohibitively time-consuming. Analyzing programs is the fastest but may not be accurate, and there is a lack of validated written assessments tailored to the learning goals of specific curricula.

Many research studies have used artifact analysis to draw conclusions about student learning [4, 9, 10, 20, 27]. 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 [5]. Conversely, students may understand concepts that they do not choose to include in their code, leading to false negatives.

Our research 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?*

This paper presents key findings from summative assessments given at the conclusion of two modules in a curriculum taught in a large urban school district in the United States (US). In the next section, we present relevant literature on assessment of student learning in primary school and in §3, we present the theoretical frameworks grounding our study. In §4, we describe our methods and experimental design. We present our results in §5. We then provide a discussion and conclusions in §6. Finally, §7 describes limitations of this study.

2 RELATED WORK

There is a wealth of literature on automated assessment, including Scrape [27], Hairball [4], and Dr. Scratch [20]. Automated assessments have gotten more sophisticated over time, moving from counting instances of particular blocks [1, 27], to identifying correct and incorrect uses of code constructs [4], to analyzing higher order understanding [20, 22].

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 [5]. 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 [4]. 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

ITiCSE '20, June 15–19, 2020, Trondheim, Norway

ACM ISBN 978-1-4503-6874-2/20/06…$15.00
https://doi.org/10.1145/3341525.3387379

are necessary to find out whether students understand the concepts both included and not included in their code.

Traditional written assessments are frequently used to assess student learning in Scratch, both in the school [18] and the extracurricular setting [16]. 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 [17, 21]. 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 [26], and middle school students students [2].

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 [5]. 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 [5]. Interviews are also very time-consuming, making them unrealistic for teachers who are already very time-constrained.

This work builds upon previous research in assessing student learning by exploring the relationship between attributes in student artifacts and their performance on written assessment questions on events, sequences, and loops. Written assessments were selected over interviews because we were designing for the formal school setting, where interviews would be impractical for school teachers with limited time and resources.

3 THEORETICAL FRAMEWORK

To contextualize our exploration of the relationship between demonstrated coding ability and understanding, we present the Block model of program comprehension [24]. The Block model introduces a duality between a structural and functional understanding. A structural understanding is understanding *how the code works*, which encompasses the syntax and semantics of a programming language (text surface), and the data and control flow in a program (program execution). In contrast, a functional understanding is understanding *what the code does*, i.e. the more abstract purpose of the code.

When applied to Scratch programming pedagogical approaches, functional understanding might be expected out of students who remixed projects (because they know what the code does but not necessarily how or why it works that way), whereas structural understanding is often the goal for students building their own projects. In this study, we focus on structural understanding. 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 [25].

4 METHODS

4.1 Study Design

The study consisted of 296 students aged 9-10 from a large urban school district in the US. Student gender was split almost evenly. The self-identified participant ethnic breakdown was 32.91% Asian, 28.79% Hispanic/Latino, 9.49 % White, 8.29% Pacific Islander, and 6.33% Black. Over the course of approximately eight months, they were taught three units in a Constructionist-inspired introductory CT curriculum in Scratch, which was a modification of the Creative Computing Curriculum [6]. Unit 1 was an introduction to Scratch, Unit 2 covered events & sequence, and Unit 3 covered loops. Units 2 and 3 aimed to teach the beginner learning goals of sequence and repetition, respectively, from [23]. Upon completion of Units 2 and 3, students took a 20-30 minute pen-and-paper assessment, consisting of multiple-choice, fill-in-the-blank, and open-ended questions.

All four teachers in the study underwent the same professional development to teach this curriculum. Three teachers taught three classrooms, and one teacher taught five classrooms. Classroom size ranged from 13 to 31 students. Each lesson took 60-90 minutes and took place once every 1-2 weeks, depending on the classroom. The primary language of instruction was English.

In addition to their assessments, we also collected students' culminating projects for each unit. 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 Unit 2, and 275 students submitted projects for Unit 3.

4.2 Assessment Design

Our assessment design was guided by the Evidence-Centered Design Framework [19]. Domain analysis was informed by the CS K-12 Framework and the K-8 learning trajectories for elementary computing [23]. These overarching goals were narrowed in domain modeling to identify specific knowledge and skills desired. In this study, the artifacts students created are projects that students create on their own, not remixed from the Scratch 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.

The assessment questions were designed by a team of CS and education researchers and practitioners. For face validity, questions were then reviewed by a group of four practitioners. Cronbach's alpha (α) was also calculated for internal reliability between questions on the same concept to ensure that they produce similar scores.

Written results were analyzed to remove questions for which formatting led to spurious markings or open-ended question wording led to answers that did not provide insight into understanding. In this paper, we present a question each on sequence, events, and parallelism, as well as 4 questions on loops. One of the loops questions has 3 sub-questions for a total of 6 items (α=.8)

4.3 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 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, Unit 2 taught sequence and events and Unit 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 [12]. 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 1: Attributes from Artifact Analysis

5 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).

5.1 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 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 1. 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 1, 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

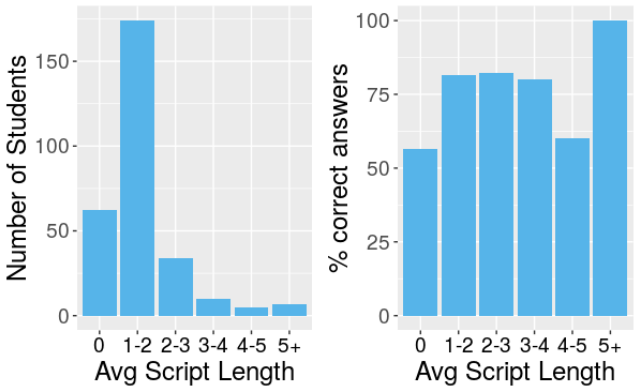

Figure 1: Q1 Student Count (left) & Correct Responses (right) for Average Script Length

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.

5.2 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 2).

5.3 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

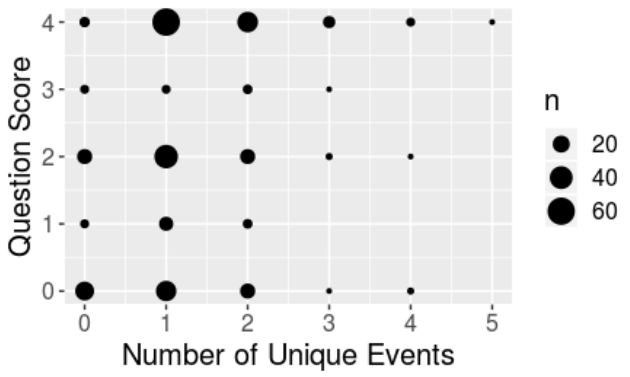

Figure 2: Q2 Events for 1 Script vs No. of Unique Events

the actions sequentially, whereas Giga's two scripts run in parallel (Figure 3). 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 4). 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 [14, 15].

5.4 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 6.

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

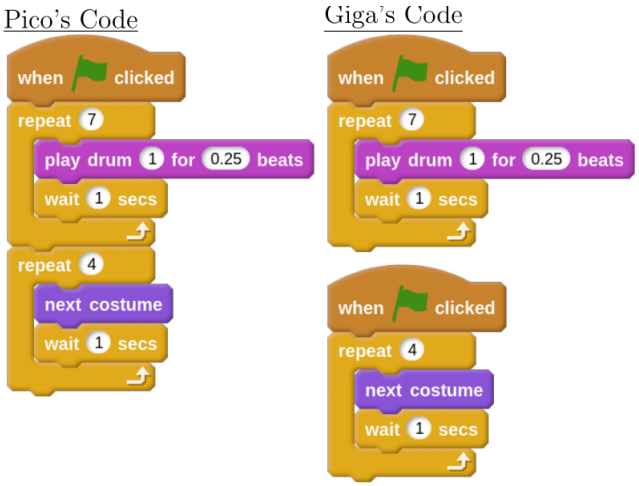

Figure 3: Q3 Sequential (left) and Parallel (right) Scripts

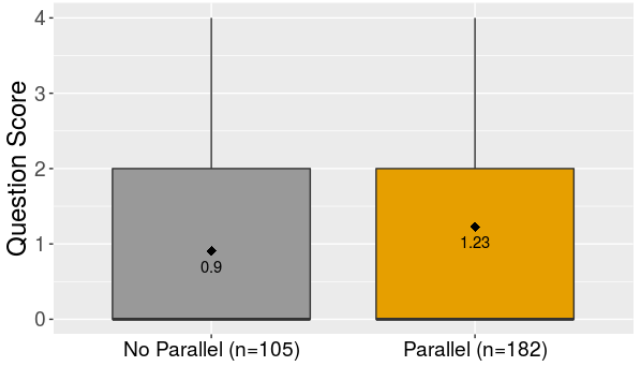

Figure 4: Q3 Events for Multiple Scripts

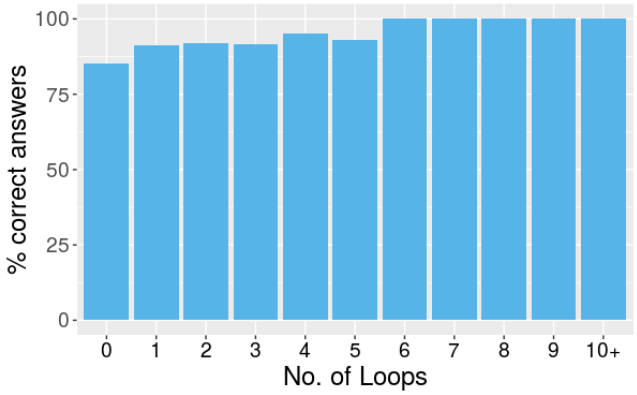

Figure 5: Q4 Correct Responses vs No. of Loops

below the cut-off (Figure 5). 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.

5.5 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

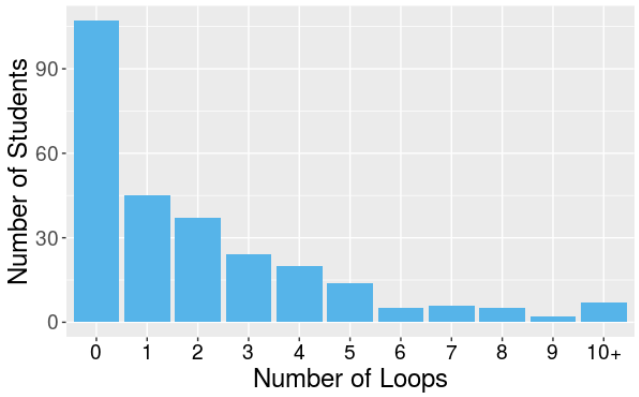

Figure 6: Student Count for No. of Loops

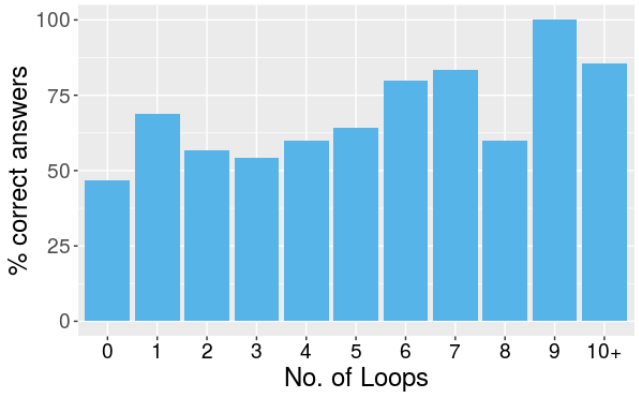

Figure 7: Q5 Correct Responses vs No. of Loops

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 7). 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.

5.6 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). 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.

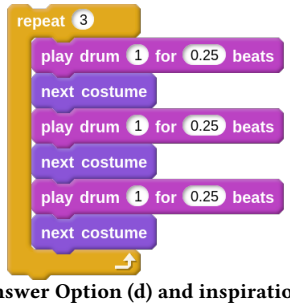

Figure 8: Q5 Answer Option (d) and inspiration for question.

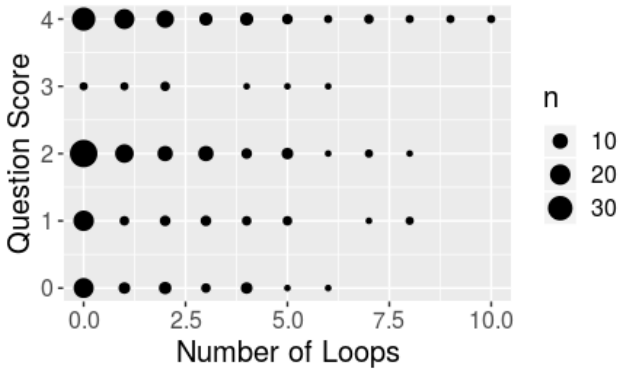

Figure 9: Q6 Repeat Blocks & Loops vs No. of Loops

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 9).

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.

5.7 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 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 10) 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.

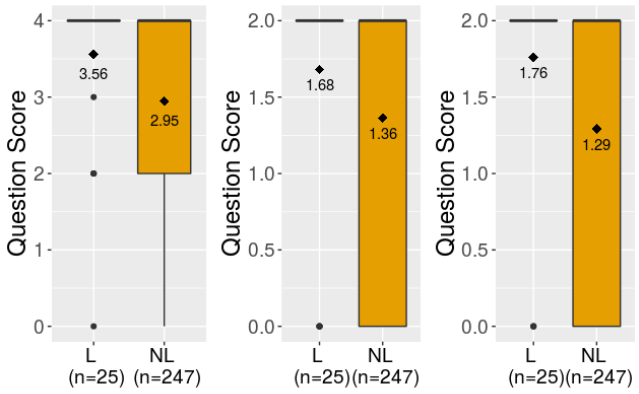

Figure 10: Q7a-c (left to right) vs Presence of Loop in Sequence (Loop/No Loop)

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.

6 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	Tot	Uni	Cat	Tot	Len	Tot	Tot	Uni
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 2: Correlations between Question Score and Project Attributes. 'Tot', 'Uni', 'Cat', & 'Len' are short for 'Total', 'Unique', 'Categories', & 'Length', respectively. Dash ('-'), 'VW', & 'W' indicate no, very weak, and weak correlations, respectively. Correlation intervals are in Section 4.3

Concept-related code constructs had either very weak or no correlations with performance on the written assessments for every question (shown in blue on Table 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 (shown in yellow on Table 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 [5].

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 [3]) and some not (like a piece of knowledge approach [11]). 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 [23], 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 primary 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.

7 LIMITATIONS

Students were drawn from 14 classrooms at 4 schools taught by 4 different teachers. While teachers all used the same curriculum and received the same training, instructional time varied by classroom, which could have implications for correlations with student artifact attributes. We note that in prior studies, even when artifact analysis has been used to study understanding, there is no established or validated knowledge on how much time students should be given to create artifacts such that their work can demonstrate understanding.

8 ACKNOWLEDGEMENTS

This project was funded by the US National Science Foundation (NSF) Grant No. 1660871 and DGE-1746045.

REFERENCES

[1] 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. ACM. 2012, pp. 643–648.

[2] Min-W. Wiebe E. Mott B. Boyer K. E. Lester J. Akram B. "Assessing Middle School Students' Computational Thinking Through Programming Trajectory Analysis". In: *Proceedings of the 50th ACM technical symposium on Computer science education*. ACM. 2019, p. 1269.

[3] Michael T Battista. "Conceptualizations and issues related to learning progressions, learning trajectories, and levels of sophistication". In: *The Mathematics Enthusiast* 8.3 (2011), pp. 507–570.

[4] Bryce Boe et al. "Hairball: Lint-inspired Static Analysis of Scratch Projects". In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE '13. Denver, Colorado, USA: ACM, 2013, pp. 215–220. ISBN: 978-1-4503-1868-6. DOI: 10.1145/2445196.2445265. URL: http://doi.acm.org/10.1145/2445196.2445265.

[5] 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*. Vol. 1. 2012, p. 25.

[6] Creative Computing. *An introductory computing curriculum using Scratch*.

[7] *CS for ALL*. URL: https://www.csforall.org/.

[8] Louise P Flannery et al. "Designing ScratchJr: support for early childhood learning through computer programming". In: *Proceedings of the 12th International Conference on Interaction Design and Children*. ACM. 2013, pp. 1–10.

[9] Diana Franklin 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*. ACM. 2013, pp. 371–376.

[10] Diana Franklin et al. "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*. ACM. 2017, pp. 231–236.

[11] David Hammer and TIFFANY-ROSE SIKORSKI. "Implications of complexity for research on learning progressions". In: *Science Education* 99.3 (2015), pp. 424–431.

[12] Dennis E Hinkle, William Wiersma, Stephen G Jurs, et al. "Applied statistics for the behavioral sciences". In: (1988).

[13] Peter Hubwieser et al. "A global snapshot of computer science education in K-12 schools". In: *Proceedings of the 2015 ITiCSE on working group reports*. ACM. 2015, pp. 65–83.

[14] Yifat Ben-David Kolikant. "Gardeners and cinema tickets: High school students' preconceptions of concurrency". In: *Computer Science Education* 11.3 (2001), pp. 221–245.

[15] Gary Lewandowski et al. "Commonsense computing (episode 3): concurrency and concert tickets". In: *Proceedings of the third international workshop on Computing education research*. ACM. 2007, pp. 133–144.

[16] 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*. ACM. 2012, pp. 57–62.

[17] Eva Marinus et al. "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*. ACM. 2018, pp. 133–141.

[18] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. "Learning computer science concepts with scratch". In: *Computer Science Education* 23.3 (2013), pp. 239–264.

[19] Robert J Mislevy and Geneva D Haertel. "Implications of evidence-centered design for educational testing". In: *Educational Measurement: Issues and Practice* 25.4 (2006), pp. 6–20.

[20] Jesús Moreno-León et al. "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. Denver, Colorado, USA: ACM, 2017, pp. 2788–2795. ISBN: 978-1-4503-4656-6. DOI: 10.1145/3027063.3053216. URL: http://doi.acm.org/10.1145/3027063.3053216.

[21] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[22] Alexander Repenning and Andri Ioannidou. "Broadening participation through scalable game design". In: *ACM SIGCSE Bulletin*. Vol. 40. 1. ACM. 2008, pp. 305–309.

[23] Kathryn M Rich et al. "K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals". In: *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ACM. 2017, pp. 182–190.

[24] 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*. ACM. 2008, pp. 149–160.

[25] Juha Sorva. "Notional Machines and Introductory Programming Education". In: *ACM Transactions on Computing Education* 13 (June 2013), 8:1–8:31. DOI: 10.1145/2483710.2483713.

[26] 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*. ACM. 2010, pp. 97–101.

[27] 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.