Patterns in Elementary-Age Student Responses to Personalized & Generic Code Comprehension Questions

Jean Salac, Qi Jin, Zipporah Klain, Saranya Turimella, Max White, & Diana Franklin
University of Chicago
Chicago, Illinois
{salac,qijin,zklain,sturimella,mnwhite,dmfranklin}@uchicago.edu

ABSTRACT

The CS community has struggled to assess student learning at the K-8 level, with techniques ranging from one-on-one interviews to written assessments. While scalable, automated techniques exist for analyzing student code, a scalable method for assessing student comprehension of their own code has remained elusive. This study is a first step in bridging the gap between the knowledge gained from interviews and the time efficiency and scalability of written assessments and automated analysis. The goal of this study is to understand how student answers on various types of questions differ depending on whether they are being asked about their own code or generic code. We find that while there were no statistically-significant differences in overall scores, questions about generic and personalized code of comparable complexity are far from equivalent. Our qualitative analyses revealed interesting patterns in student responses, inviting further research into this assessment technique. In particular, students answered differently from students with generic code when presented with individual blocks from their code taken out of context and placed into different code snippets, and students answered in a way that demonstrates a functional, instead of structural, understanding on Explain in Plain English (EiPE) questions.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Student assessment**; *Computational thinking*;

KEYWORDS

K-8 education, assessment, code comprehension, computational thinking, Scratch

ACM Reference format:
Jean Salac, Qi Jin, Zipporah Klain, Saranya Turimella, Max White, & Diana Franklin. 2020. Patterns in Elementary-Age Student Responses to Personalized & Generic Code Comprehension Questions. In *Proceedings of The 51st ACM Technical Symposium on Computer Science Education, Portland, OR, USA, March 11–14, 2020 (SIGCSE '20),* ACM, New York, NY, USA. 7 pages. https://doi.org/10.1145/3328778.3366833

SIGCSE '20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366833

1 INTRODUCTION

The CS4All movement advocates computer science (CS) instruction for all students, and several large, urban school districts (including Chicago, New York, and San Francisco) have started district-wide efforts to get CS into elementary schools. Moving from the after-school, informal domain into the formal school classroom increases the pressure on developing accurate assessment techniques that match the pedagogical approaches and tools used for this age group.

A popular programming language and development environment used in elementary school is Scratch [6]. Three assessment techniques are common in this realm: analyzing programs that students create (artifact analysis), giving written assessments, and interviewing students. Each has its major advantages and drawbacks. For example, artifact analysis has both false negatives and false positives. That is, students include code in their programs that they do not understand [3], leading to false positives. At the same time, students may understand concepts that they do not choose to include in their culminating project. On the other hand, interviews are the most accurate measure because of the ability to ask follow-up questions, but they are prohibitively time consuming.

In this study, we explore the space with a novel approach—creating a personalized assessment that asks students questions involving their own code. The ultimate goal is to create written assessments that allow students to demonstrate their understanding of code within their artifacts and beyond. However, this study is just one step towards this larger goal. It seeks to answer the following question: *How does integrating a student's code from their artifacts affect the understanding they demonstrate on written assessment questions?*

This paper presents key findings from different types of assessment questions using generic code or code drawn from student projects, including the following patterns:

- students answer differently when their code blocks are taking out of context and placed int a different script in multiple-choice questions,
- results were mixed when asking multiple-choice or fill-in-the-blank questions about partial or full scripts when kept in context, and
- student answer in a way that demonstrates functional understanding but not structural understanding when asked EiPE questions about their own code.

In the next section, we present relevant literature on assessment of student learning in elementary school. We then describe our personalized assessment generator in Section 4.1. In Section 4, we describe our methods. We present our results in Section 5. We then provide conclusions in Section 6.

2 BACKGROUND & RELATED WORK

In this section, we provide background on three methods of assessment: automated assessment, interviews, and written assessments.

There is a wealth of literature on automated assessment, including Scrape[19], Hairball[2], and Dr. Scratch[14]. Automated assessments have gotten more sophisticated over time, moving from counting instances of particular blocks [19, 1], to identifying correct and incorrect uses of code constructs[2], to analyzing higher order understanding[14, 16].

However, any technique focused on artifact analysis assumes that students understand the code they use in their projects. This is not necessarily true [3]; students can use code in their projects that they don't truly understand, either by copying exact code they were taught or remixing from the Scratch community. In addition, a student may understand a concept even though they did not choose to use it in their open-ended artifact. Written assessments or interviews are necessary to find out whether students understand the concepts both included and not included in their code.

Written assessments are used to assess student learning in Scratch, both in formal [8, 12] and informal [7, 4, 10] settings. In addition, Marinus et al. developed an assessment around Cubetto, a simplified version of the turtle LOGO programming task developed by Seymour Papert [11, 15].

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 well as how students were employing computational thinking practices while developing their projects[3].

While interviews can provide a more complete and process-oriented picture of student learning, interviews are limited by what students can remember about their projects and the project(s) selected for discussion [3]. 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 written assessments that use code snippets from student artifacts to understand how that affects the way in which students answer the questions.

3 THEORETICAL FRAMEWORK

To contextualize our study, we use Schulte's Block model, a model for program comprehension [18]. It comprises of a duality between "structure" (how the code works) and "function" (the purpose of the code). In the context of Scratch programming, it would be reasonable to expect a student who remixes projects to have a functional understanding of the code—they may know what the code does, but not how or why. However, structural understanding is frequently the goal when students are asked to build their own projects, which is the case with the curriculum in this study.

In this study, we experiment with integrating student code into different kinds of questions to see what patterns emerge. For example, Explain in Plain English (EiPE) questions present an interesting conundrum. To answer an EiPE question about code they did not write, students must arrive at a functional understanding of the code by leveraging a structural understanding, i.e. interpreting the syntax, semantics, and control flow of the code. However, this is not the case when asking students about their own code as they presumably would have run it and seen its results. Grounded in the Block model, we investigate the different kinds of understanding that students demonstrate when presented with code from their own projects in various ways, as opposed to generic code snippets.

4 METHODS

4.1 Personalized Assessment Worksheets for Scratch (PAWS)

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

4.2 Study Design

The study consists of two years of students from a large urban school district, with revisions to assessment question formats occurring between the two years. In the first year, our study consisted of 316 4th grade students. Student gender was split almost evenly. The participant ethnic breakdown was 32.91% Asian, 28.79% Hispanic/Latino, 9.49 % White, 8.29% Pacific Islander, and 6.33% Black. In the second year, our study consisted of 329 3rd, 4th, and 5th grade students.

In both years, students completed three modules in an introductory computational thinking (CT) curriculum in Scratch over the course of a year—the first module was an introduction to Scratch, the second covered events & sequence, and the third covered countable loops. The Constructionist-inspired [9] curriculum was a modification of the Creative Computing Curriculum [5]. For all students in the study, this curriculum was their first formal in-school computing experience, though they may have had informal out-of-school exposure.

4.3 Assessment Design

Upon completion of Modules 2 and 3, students took a pen-and-paper assessment, consisting of multiple-choice, fill-in-the-blank and open-ended questions. Following the Evidence-Centered Design framework [13], assessments were designed based on domain analysis informed by the CS K-12 framework and K-8 learning trajectories [17]. Overarching learning goals were narrowed in domain modeling to identify specific knowledge and skills desired.

The assessment questions were designed by a team of CS and education researchers and practitioners. For face validity, questions were then reviewed by a larger group of practitioners and reading comprehension experts. They were also piloted in Year 1 of our study and improved upon in Year 2. Assessment questions were analyzed by undergraduate researchers.

4.4 Data Analysis

Analysis was performed separately on each question, including *only* students with candidate code for that question. Due to absent

students or blank responses, there was a slight (< 10%) imbalance in the number of generic and personalized questions available for analysis. To account for the imbalance, the Type 3 Sum of Squares was used.

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

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

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

5 RESULTS

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

5.1 Blocks Integrated in Generic Code

This question (B1) asked students to circle the `say` block that ran last in a script with alternating `say` and `wait` blocks (Figure 1). Candidate code was any `say` block. Candidate `say` blocks replaced generic `say` blocks in the script.

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

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

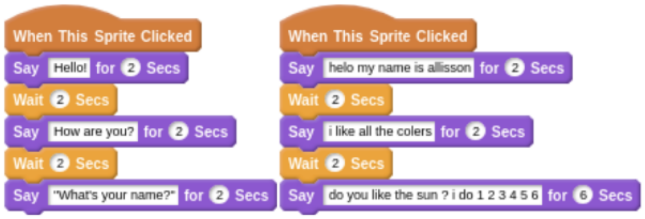

Figure 1: Generic (left) and Personalized (right) Scripts in B1

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

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

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

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

In Year 1's personalized assessment, one of the generic `when sprite clicked` scripts was retained, and a candidate script could be swapped with any of the other three scripts. However, this could lead to one, two, or three `when sprite clicked` events. To hold the number of each event type constant, in Year 2, candidate code only swapped out a script with same event block (though a different parameter was allowed for when key pressed).

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

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

From Year 1 to Year 2, there was also an improvement in the proportion of students circling all the correct answers, as opposed to only a subset. This may be attributed to the stricter candidate

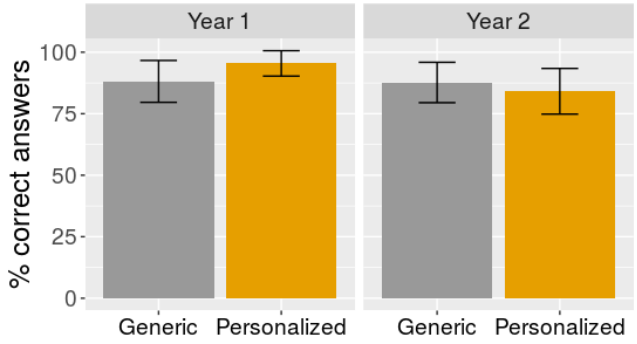

Figure 2: MF2 Repeat Iteration Count Results

code selection criteria and/or more direct instruction of parallelism in Year 2.

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

Table 1: MF1 Qualitative Results

5.2.2 MF2: Repeat Iteration Count. Students were shown a repeat block and asked how many times the loop would repeat. Candidate code was defined as a repeat block with fewer than 3 blocks inside it. If a student had a repeat block with more blocks, PAWS included only the first three blocks.

Overall, students performed well on this question, with at least 85% of students answering correctly in both years. There was no statistically-significant difference between control and treatment (Year 1: F(1,123)=2.27, p=.13; Year 2: F(1,126)=.33, p=.56; see Figure 2).

5.2.3 MF3: Unrolling a Loop. In Year 1, students were shown two blocks inside a `repeat 4` loop and given choices of those blocks repeated 1, 2, 3, and 4 times. Students were asked to choose the unrolled code that did the same thing as the loop. Candidate code consisted of a repeat block with 2 blocks that was repeated at most 4 times. If a student had a repeat block with more than 2 blocks and/or the repeat block was repeated more than 4 times, PAWS included only the first 2 blocks and changed the number of times the repeat block ran to 4.

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

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

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

5.3 Open-Response Questions

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

5.3.1 OR1: Explain a Sequence. In Year 2, OR1 was an open-response question in which students are provided a single script and asked to explain what that script does. OR1 was design to allow students to demonstrate (1) their ability to *articulate* an instruction's functionality and their understanding of the *order* of instructions, and (2) their ability to identify the *event* that causes the action. A candidate code snippet was any script, excluding code constructs beyond the scope of their curriculum (e.g. forever loops, conditionals, and variables), limiting script length by allowing a maximum of 3 action blocks (the length of the generic script), and excluding blocks that had ambiguous sequential execution (e.g. the `play sound` block).

Our Year 2 revision transformed a fully open-ended question (Figure 3) into more targeted questions that asked students to identify the event and sequence of the script (Figure 4). The number of lines changed based on the number of action blocks in their script, as we allowed scripts with anywhere from 1-3 action blocks.

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

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

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

In addition, 5.05% of students with personalized assessments, in contrast with 2.11% of students with generic assessments, had

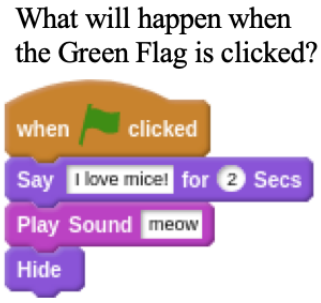

Figure 3: Year 1 Fully Open-Ended OR1

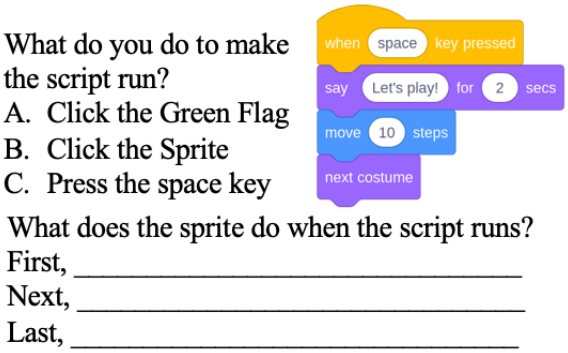

Figure 4: Year 2 Scaffolded OR1

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

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

5.3.2 OR2: Explain a Loop. For OR2, students were shown a loop and asked to explain what the loop would do in their own words.

In Year 1, candidate code was defined as either a countable or forever loop. Answers were given between 0-10 points depending on accuracy and completeness. As in OR1, the Year 2 question was revised to provide more scaffolding for student responses. Instead of a blank empty box, students were given lines preceded by "First", "Next", and "Last", similar to OR1 (Figure **??**), and there was a separate blank for the number of times the loop would repeat. Year 2's OR2 was graded in the same way as Year 2's OR1, with an additional point for identifying the number of loop iterations. In addition, forever loops were excluded as they were not explicitly covered in the curriculum; thus, we could not expect students to understand them.

Most students performed well on this question, with an average score of 8.37 out of 10 possible points in Y1 and 5.97 out of 7 points in Y2. As shown in Figure 5, there was no statistically-significant difference in performance between students who received a generic

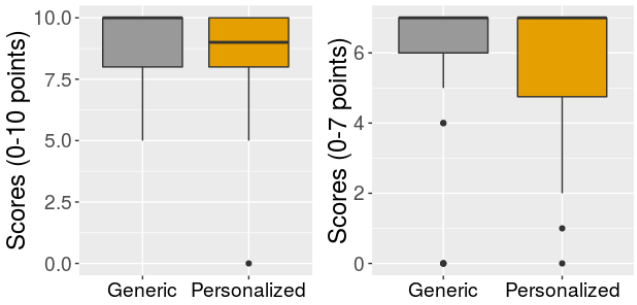

Figure 5: OR2 Free-Response on Loops Results

question and students who received a personalized question (Y1: F(1,156)=3.30, p=.071, Y2: F(1,75)=.69, p=.50).

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

Students who received personalized questions were more likely to write responses that were not precise enough to assess their structural understanding (how the code works). A greater proportion of them left out parameters (8.33% personalized vs 7.69% generic). They were also more leave out block descriptions entirely (16.67% personalized vs 0% generic; p <.05). As in OR1, some students with personalized questions brought in context from their projects in their responses, demonstrating functional instead of structural understanding (Figure 6).

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

Nonetheless, there was one trend from OR2 that differed from OR1. In OR1, a greater proportion of students with *personalized* code described a block incorrectly. In contrast, a greater proportion of students with *generic* code described a block incorrectly in OR2 (12.82% vs. 2.78%; p=.20). Closer inspection of responses revealed that for this question, more of the personalized code included blocks that were taught in the curriculum, whether in the events & sequence module or in the loops module. By this point in the curriculum, students have been exposed to more blocks. Students with personalized code actually used the blocks they were being asked about. At a minimum, these students could be expected to have a functional understanding of the loop shown in their question and thus, would be less likely to provide a description that was completely wrong. On the other hand, students may not have actually used the blocks in the generic script, even though they were covered in the curriculum.

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

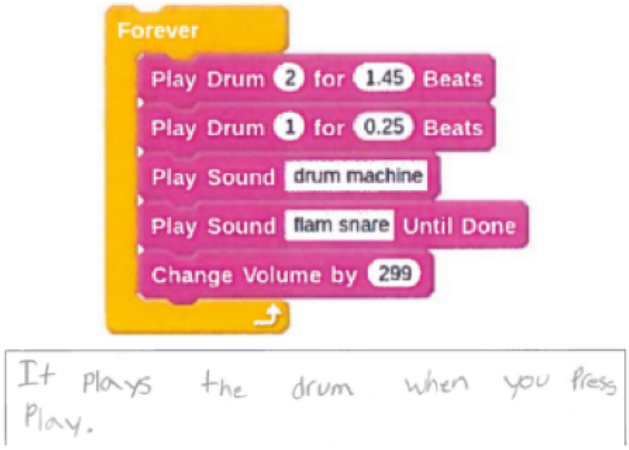

Figure 6: OR2 Personalized Script described with Project Context

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

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

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

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

5.3.4 Open-Response Discussion. Overall, we found that when students explain code snippets, they are less precise when given their own code but are more likely to answer with correct statements.

Students who received personalized code may remember their intentions in coding that script and the project in general, allowing them to better understand what the sprite would do when the script was run. However, this familiarity may put them more in a functional frame of mind, causing them to skim over the details in their answers. In addition, thinking of their specific project may have thwarted the question which asks when, in general, students should use loops in projects.

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

6 CONCLUSION

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

We identified the following patterns:

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

When students with personalized and generic questions answered differently, they demonstrated a functional understanding of the code in the question, instead of the intended structural understanding. As they built the code snippets they were being asked about, they were likely to remember their goals while creating their projects; interviews about student artifacts also face a similar challenge [3]. Further research is merited to explore how to ask questions that are not overly complicated but cause students to demonstrate structural understanding of their code.

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

7 LIMITATIONS

The technique used in this study is product-oriented, limiting what we could ask students. We can only postulate why different kinds of understanding were elicited, not fully explained them. More process-oriented future work, including thinkalouds as students approach these questions, would be necessary to better understand the reasons behind these differences.

8 ACKNOWLEDGEMENTS

This project was funded by 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] 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.

[3] 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.

[4] Quinn Burke and Yasmin B Kafai. "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*. ACM. 2012, pp. 433–438.

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

[6] 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.

[7] 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.

[8] Michal Gordon, Assaf Marron, and Orni Meerbaum-Salant. "Spaghetti for the main course?: observations on the naturalness of scenario-based programming". In: *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*. ACM. 2012, pp. 198–203.

[9] Idit Ed Harel and Seymour Ed Papert. *Constructionism*. Ablex Publishing, 1991.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.