

Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic

Shuchi Grover
SRI International
333 Ravenswood Ave. Menlo
Park, CA 94025, USA
shuchi.grover@sri.com

Satabdi Basu
SRI International
333 Ravenswood Ave. Menlo
Park, CA 94025, USA
satabdi.basu@sri.com

ABSTRACT

Programming in block-based environments is a key element of introductory computer science (CS) curricula in K-12 settings. Past research conducted in the context of text-based programming points to several challenges related to novice learners' understanding of foundational programming constructs such as variables, loops, and expressions. This research aims to develop assessment items for measuring student understanding in introductory CS classrooms in middle school using a principled approach for assessment design. This paper describes the design of assessments items that were piloted with 100 6th, 7th, 8th graders who had completed an introductory programming course using Scratch. The results and follow-up cognitive thinkalouds indicate that students are generally unfamiliar with the use of variables, and harbor misconceptions about them. They also have trouble with other aspects of introductory programming such as how loops work, and how the Boolean operators work. These findings point to the need for pedagogy that combines popular constructionist activities with those that target conceptual learning, along with better professional development to support teachers' conceptual learning of these foundational constructs.

CCS Concepts

Social and professional topics~Computational thinking • Social and professional topics~Computer science education • Social and professional topics~Student assessment • Social and professional topics~K-12 education • Information systems~Data mining • Computing methodologies~Semi-supervised learning settings

1. INTRODUCTION

Programming or “coding” is a key element of introductory computer science curricula in K-12 classrooms in the US. In order for “CSForAll” to achieve the broader goal of preparing K-12 learners for future studies and careers in CS, students need to be engaged in early experiences with programming while also learning the requisite foundational concepts of computational problem solving. Block-based programming environments such as Scratch, Alice, Snap!, App Inventor, Blockly are popular vehicles for introductory programming. They provide a fun and engaging

introduction to concepts without having to deal with syntax that has historically plagued novice learners of text-based programming. CS education research over the last four decades has documented the types of issues that learners struggle with as they encounter programming concepts [e.g. 3,4,8,11,13,16-21]. Most of these studies were conducted in the context of text-based programming in CS1 undergraduate coursework. They discussed conceptual and cognitive difficulties in dealing with the process of constructing programs, and also challenges posed by specific programming constructs and control structures such as variables and other data structures, various types of looping structures, logical flow using conditionals, and Boolean logic. Although studies in the 1980s examined the many conceptual challenges for younger learners working in environments such as LOGO and BASIC [e.g. 16], few studies since then have looked at learner misconceptions of introductory computing and algorithmic concepts, especially in block-based programming environments, among tween and teen learners in middle school.

This paper reports on research to measure student understanding of key programming concepts such as variables, expressions using variables, loops, and Boolean logic. This is part of a larger effort to design curricular activities that help learners explore these concepts in engaging ways and have a stronger understanding of them before they need to use them in programming. Our interest in these concepts is inspired by our own experiences [8] and prior literature that points to problems novice programmers face in introductory programming. The following section presents a review of relevant literature. Then, we describe our process for creating assessments to capture student understanding, and the results of a pilot study with 100 middle school students. We delve into an analysis of what the results reveal about student misconceptions regarding the constructs of interest. While some misconceptions corroborate findings that have been documented in past studies (albeit being in the new context of middle school students using block-based programming), some are hitherto undocumented, to our knowledge. We end with a discussion on how our results informed revisions to the assessments, implications of the results for CS pedagogy in K-12 classrooms, and recommendations for practitioners and curriculum designers on pedagogies that may be employed to address misconceptions.

2. RELATED WORK

Programming is a complex cognitive activity [16]. There is extensive past literature in CS education research dedicated to the various challenges faced by novice programmers in their early encounters with algorithmic constructs and the complex process of problem solving involved in programming. Several past studies examine the cognitive demands of learning programming and the process of problem solving involved in programming [e.g.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SIGCSE '17, March 8–11, 2017, Seattle, WA, USA.

© 2017 ACM. ISBN 978-1-4503-4698-6/17/03...\$15.00.

DOI: <http://dx.doi.org/10.1145/3017680.3017723>

12,16,21]. These papers focus not so much on learners' issues with specific constructs as they do with the problems associated with the broader understanding of programming—assembling the constructs sensibly to create a working program.

Others, however, have focused on issues pertaining to specific programming constructs, and are thus more relevant to this research. For example, several problems pertaining to the use of variables, expressions, and loops have been reported [3], specifically, flawed ideas about variable assignment, variables being able to assume multiple values at the same time, distinguishing between what goes inside a loop and what precedes or follows a loop, and that an expression involving the control variable of a loop can have different values in each cycle of the loop. Extensive research on misconceptions related to variables [20] highlighted the difficulties learners have with initialization of variables since it is hard for students to make assumptions about the initial state of a system. Other literature reported students' difficulties with understanding how and when to terminate loops [4,18]. Recent research with Scratch also reports that students struggle with loops, especially 'repeat-until' loops that have terminating conditions involving variables [8]. Novices also have problems with analyzing and designing mathematical and logical expressions, naming variables and assigning suitable data types and structures to these variables and expressions [5]. Further, students tend to perform worse on the logic-based questions on the Advanced Placement CS Exam than on any other type of question on the exam [1]. Problems pertaining to the understanding of conditionals are often attributed to logical operators in IF statements [4]. The Boolean AND/OR operators are often mistakenly interpreted as they are in the English language. Specifically, students tend to misinterpret the OR operator as true when one of the operands is true, but not both [9].

Research involving teens using Scratch in informal settings reported infrequent use of logic and variables in student's programs compared to constructs such as conditionals that are used five times more frequently [11]. This suggests that without teaching and guidance, teens find it harder to create programs requiring the use of variables and logic—concepts that we expect our middle school students to learn. These findings echo those in [13] with middle school students' use of Scratch. We are therefore motivated in our current research to focus on these concepts that have been shown to be problematic for novices in various settings, namely variables, expressions (arithmetic and logical), loops and Boolean logic, and see how we can measure understanding of these concepts in the context of middle school students who are learning introductory programming and algorithmic thinking in block-based programming environments such as Scratch.

3. METHODOLOGY

This research is informed by the broader research question: How can learning outcomes for computing constructs such as variables, expressions (arithmetic and logical), and loops, be organized into a structured assessment framework and measured with technical quality? A subgoal addressed in this paper : What do assessments aimed to measure student understanding of computing constructs such as variables, expressions and loops, tell us about student understanding and misconceptions related to these concepts in the context of block-based programming in middle school CS?

3.1 Design of an Assessment Instrument

In order to create measures of student understanding of VEL concepts, we were guided by Evidence-Centered Design (ECD), a

principled framework for assessment design [14]. ECD enables assessment developers to work collaboratively with domain and assessment experts to build artifacts that support task development with warrantable claims. The ECD process forces thinking about the claims we want to make about student learning and the evidence we must gather to support the claims, followed by designing tasks that elicit such evidence. This methodology has been used in recent CS education efforts for the development of assessments for the Exploring Computer Science curriculum [6].

Based on the phases outlined by ECD, we started the process with a domain analysis that clarified learning goals for middle school CS and especially the goals concerning VEL concepts. These learning goals were influenced by earlier work on the FACT curriculum [7] that was a precursor to this project, as well as the new K-12 CS framework (k12cs.org) that outlines the need for learners to understand and use the VEL concepts in grades 6-8. A key outcome for the domain analysis was to clarify the included aspects and boundaries of the proposed VEL constructs.

All constructs begin with “Students will learn...”

- *how simple loops work (fixed number of repetitions)*
- *algorithmic flow of control—how sequence, repetition and selection works; how instructions are executed in sequence even when there are loops, except that the set of instructions within a loop are repeated*
- *what data is, and how it is used in a program*
- *how data types define the set of values a variable can have, and the set of operators that can be used*
- *how to create, use, assign values to, and update variables*
- *how variable values change within loops*
- *what initialization is and why it is important*
- *using expressions to make new variables from existing ones*
- *about Boolean variables, operators & expressions*
- *the idea of controlling loops and conditionals using Boolean conditions (may/may not involve variables and expressions)*
- *to identify and articulate patterns in real-world phenomena and problems, and abstract them into structural components of a program (pre-loop actions, repeating logic in a loop, post-loop actions)*
- *how variables are an abstraction or representation of data in the program and the real world*

In the next step, we developed a “design pattern” or assessment argument for VEL concepts. This design pattern emphasizes the definition of the focal knowledge, skills, and abilities (FKSAs) that are associated with VEL, the task situations in which students will be observed, and then the observations that would relate to inferences about students' learning. Note that while learning goals are teacher- & curriculum-facing, FKSAs are assessment-facing. The following are some key FKSAs for our assessment items.

1. Ability to describe what a given loop is doing
2. Ability to describe the sequence that is executed in a given program when the program contains things inside a loop as well as outside of the loop.
3. Knowledge that a loop involves a repeating pattern, that will terminate under a specified condition or after a certain number of repetitions
4. Ability to identify the repeating pattern within a loop
5. Ability to describe the structural components of a pattern (not in a programming context).
6. Ability to identify a pattern from a real-world phenomenon
7. Ability to describe how a conditional pathway would operate
8. Ability to create variables, assign values and update variables
9. Ability to describe how a variable changes values in a loop


10. Ability to determine what variables are required in a program to achieve the goals of the computational solution.
11. Ability to evaluate a Boolean expression
12. Ability to use Boolean operators in a programming context
13. Ability to create a Boolean expression for a given condition
14. Ability to identify sub-parts of a computational solution
15. Ability to create a Boolean test to control a loop given specifications
16. Ability to describe how the Boolean tests interacts with the loop execution


In the next step, we defined a conceptual assessment framework, which articulated a blueprint (specification) for the VEL assessment. We defined key features of the task environment; and plans for the measurement model—a summative assessment that would be administered as a pencil-paper test and take no longer one 50-minute class period. We first designed about twice as many items as were needed for a pilot assessment. These were reviewed by experts in the field and reduced to set of 10 assessment items to be piloted. Table 1 describes how each of the 10 items mapped to specific FKSA(s) that were in turn aligned to the learning goals. Since the FKSA(s) encompassed real-world and programming contexts, some of the items were in the context of Scratch programming while others assessed broader algorithmic thinking and problem solving skills needed to code in Scratch.

Table 1. Mapping between assessment items and FKSA(s) assessed

Assessment questions	Target FKSA(s)
Item 1	FKSA 7
Item 2,3	FKSA(s) 1, 2
Item 4	FKSA(s) 1, 2, 8, 9
Item 5	FKSA 11 (item with images)
Item 6	FKSA 11 (item with words/text)
Item 7	FKSA(s) 4, 5
Item 8	FKSA(s) 5, 6
Item 9	FKSA 12
Item 10	FKSA(s) 12, 13, 14

We present 4 items (see Figures 1-4) in more detail here as space constraints allow us to discuss students' responses to only a few specific items.





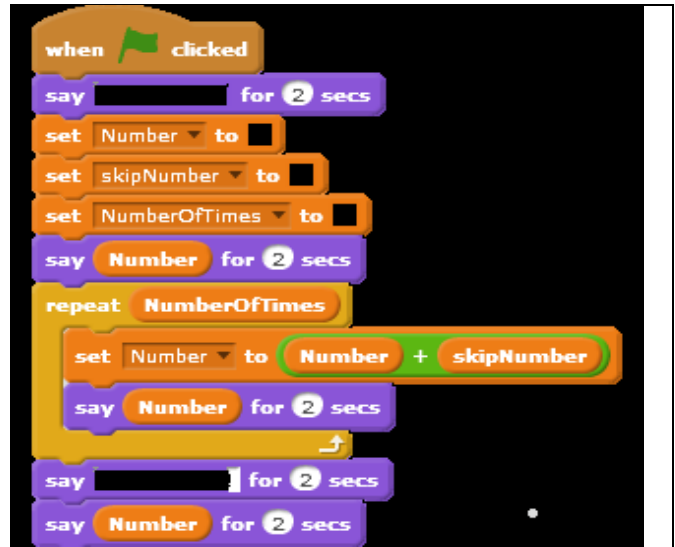
For the program on the left, write down, in order, what the fox says after the green flag is clicked.

Figure 1: Assessment Item #2a

3.2 Data Measures

Two middle school CS teachers helped pilot the assessments in mid-May (at the end of the school year) with a total of 100 6th, 7th, and 8th grade students and provided us anonymous assessments completed by their students. The students had completed an introductory programming and CS course using Scratch that was unrelated to the current project. The goal of the pilot was to try out the assessment in a real CS classroom setting and get a sense for the perceived difficulty of items, whether students had trouble interpreting the questions, and what typical

responses looked like. The teachers were also interested in potentially using our assessments in the future.



(a) Write down, in order, what will appear on the screen in the fox's speech box, after the green flag is clicked.

(b) Does the value of **Number** change in the loop?
☐ Yes ☐ No
 If Yes, explain how it changes.

(c) Does the value of **NumberOfTimes** change in the loop?
☐ Yes ☐ No
 If Yes, explain how it changes.

Figure 2: Assessment Item #4

Logical Expression	Words	
(Starts with a D) AND (ends with an E)	<input type="checkbox"/> DANCE	<input type="checkbox"/> SOCCER
	<input type="checkbox"/> DELICIOUS	<input type="checkbox"/> SHARE
(Starts with a D) AND does NOT (end with an E)	<input type="checkbox"/> DANCE	<input type="checkbox"/> SOCCER
	<input type="checkbox"/> DELICIOUS	<input type="checkbox"/> SHARE
(Starts with a D) OR (ends with an E)	<input type="checkbox"/> DANCE	<input type="checkbox"/> SOCCER
	<input type="checkbox"/> DELICIOUS	<input type="checkbox"/> SHARE
(Starts with a D) OR does NOT (end with an E)	<input type="checkbox"/> DANCE	<input type="checkbox"/> SOCCER
	<input type="checkbox"/> DELICIOUS	<input type="checkbox"/> SHARE

Figure 3. Assessment Item #6

The teachers completed a two-question survey for each question:

1. In general, how well do you think your students will do on this question? (3-point Likert scale: A **majority** of my students will get this item **incorrect**; About **half of my students** will get this **correct**; A **majority** of my students will get this item **correct**)
2. Below is a list of the learning goals addressed in this question. Please indicate how well these learning goals were covered for this class (3-point Likert scale: **Not covered** (i.e., spent no instructional time on this goal); **Somewhat covered** (i.e., spent some instructional time on this goal); **Fully covered** (i.e., spent a lot of instructional time on this goal)).

In order to see if students' responses generalized to other settings, and also better understand some of the incorrect responses (that

were identical for many students), we then examined students' responses and misconceptions through cognitive think-alouds with three students from a minority ethnic group in a summer camp. We selected students with similar demographic profiles to the students with whom we had piloted the assessments earlier.

A VELA's Variety Store
333 Garden Way
Plantville, CA 94032
408-555-5858

April 1, 2015
Transaction: 23893

B 12:20
Register: 3

Chips	\$3.27
Sandals	\$6.45
Book	\$4.72
Stickers	\$3.65
Card	\$3.89

SubTotal

\$21.98

SalesTax

\$1.76

Total

D \$23.74

Cash

\$30.00

Change

\$6.26

E Thank you for your business!

The below pictures shows the different sections of a shopping receipt. Use this picture to answer the questions below:

- Which sections are identical on every receipt? _____
- Which sections are different on the receipts? _____
- Which sections depend on what input is taken from the customer? _____
- If you had to create a program to print the receipt, what section of the receipt would you print using a loop? _____

Figure 4. Assessment Item #8

4. ANALYSIS & RESULTS

We analyzed the completed assessments from the pilot, teacher feedback on the assessment items, and students' responses during the cognitive think-alouds as they worked on the assessment items. The 100 completed assessments were coded according to a rubric. For multiple-choice items, a code was assigned for each possible response. For open-ended items responses, codes were generated for correct answers as well as for expected erroneous responses. Additional codes were added when scoring to capture some of the frequent incorrect responses to each item. An initial test set of assessments were graded independently by two researchers, resulting in refinement of the initial rubric. Once they consensus was reached, the remaining assessments were divided between the researchers for grading purposes. In this paper, we will limit our analysis to assessment items 2a, 4, 6b, and 8 described in Section 3.

Student responses for Item2a were coded as follows:

- Correct:** Let's start!, Hello, Goodbye, Hello, Goodbye, Hello, Goodbye, Finished!
- Missed non-loop:** Alternated Hello and Goodbye correctly, but missed either the beginning *Let's start!* or the ending *Finished!* or both
- No Repetition:** Let's start!, Hello, Goodbye, Finished!
- Grouped:** Grouped the Hello's together and grouped the Goodbye's together [Let's start!, Hello, Hello, Hello, Goodbye, Goodbye, Goodbye, Finished!]
- Other:** Other incorrect response
- Missing:** Missing response

Table 2 reports student responses for Item 2a in terms of percentage of student responses corresponding to each code. We see that a majority of the students answered this item correctly. However, we notice a new misconception here—when there are multiple actions inside a loop, instead of executing a sequence of actions in a loop and then repeating the entire sequence, some students tend to repeat each action separately before

repeating the subsequent action(s), thus grouping the actions in the loop. Some student responses that were classified as 'Other' also demonstrated the grouping error, although along with other errors. It is noteworthy that students who demonstrated the grouping error in Item 2a also demonstrated the error in other assessments items that are not described in this paper. During the think-aloud study, we verified this misconception for 1 of the 3 participating students, who explained that the loop construct works by grouping the actions and repeating each separately.

Table 2. Coded student responses for Item 2a

Correct	Missing non-loop	Grouped	No Repetition	Other	Missing
70%	4%	8%	2.5%	10.5%	5%

Most students, even those who understood simple loops and answered Item 2a correctly, struggled with loops that involved variables. In Item 4, the number of times the loop must repeat is specified using a pre-assigned variable. The expression inside the loop involves two other variables that are initialized outside the loop. Responses for Item 4a were coded as follows:

- Correct:** Let's count!, 0, 2, 4, 6, 8, 10, Last Number!, 10
- Extra Count:** All correct but last number is 12 [Let's count!, 0, 2, 4, 6, 8, 10, Last Number!, 12]
- Missing Count:** [Let's count!, 0, 2, 4, 6, 8, Last Number!, 10] OR [Let's count!, 0, 2, 4, 6, 8, Last Number!, 8]
- Missing 0:** [Let's count!, 2, 4, 6, 8, 10, Last Number!, 10] OR [Let's count!, 2, 4, 6, 8, 10, Last Number!, 12]
- Other:** Other incorrect response
- Missing:** Missing response

Table 3. Coded student responses for Items 4a, 4b, and 4c

	Correct	Extra count	Missing count	Missing 0	Other	Missing
Q4a	11.7%	7.8%	3.9%	1.3%	63.6%	11.7%
	Marked yes	Marked no	Marked nothing	Correct explanation	Incorrect explanation	Missing explanation
Q4b	62.3%	23.4%	14.3%	33.8%	28.6%	37.6%
	Marked yes	Marked no	Marked nothing	Included an explanation	Did not include an explanation	
Q4c	18.2%	67.5%	14.3%	20.8%	79.2%	

Items 4b and 4c were coded based on students' responses to the multiple choice questions, and whether students indicated that the variable 'Number' increases by 2 in each cycle of the loop. Table 3 reports how students fared on Items 4a,b,c. We see that very few students could correctly write the output of the given program segment. Some students had missing or extra numbers at the beginning or end of the loop, while several students had combinations of these errors, or other responses like "Number, Number, Number, Number, Number" and "0,0,0,0,0,2,4,6,8,10". We hypothesized that students' difficulties with this item stemmed from a lack of understanding of variables and probable non-exposure to loop constructs with variables. For items 4b and 4c, we found that only about two-thirds of the students understood that the value of 'Number' changes, while the value of 'NumberOfTimes' does not.

Our hypotheses were verified during the cognitive think-alouds. None of the three students had seen a repeat block in Scratch without a number. Further, **students harbored the misconception that a variable is a letter that is used as a short form for an unknown number** – an idea that comes from middle school mathematics classes. Together, this led students to believe that "repeat(NumberOfTimes)" was a new command. One student

conjectured it was a command for multiplication by 5 (the value of *NumberOfTimes*), while another thought it would print each number five times as follows: (0,0,0,0,0,2,2,2,2,2,4,4,4,4,4,.....) After being told that *NumberOfTimes* was indeed a variable, the students could correctly predict the program output, though they **continued to take issue with the length of the variable name**. Due to similar misconceptions, one of the students was of the opinion that the value of *Number* does not change while that of *NumberOfTimes* does. The “say (*Number*)” block is present before and within the loop, but “set (*NumberOfTimes*)” block is used outside the loop while “repeat (*NumberOfTimes*)” is used in the loop. In addition, students also grappled with the concept of a variable whose value changed inside a ‘repeat’ control structure. **Students articulated that a loop repeats the same set of actions and expected loops to produce the exact same output in every iteration.** This misconception was also manifested in student responses to Item 8d where students were asked to identify which section of a receipt could be generated using loops. While students could generally answer other subparts of Item 8 correctly, less than 4% of the students in the pilot study got Item 8d correct. During the think-alouds, students opined that none of the sections of the receipt could be generated using loops since loops are used to generate the same output in every iteration.

Item 6 assessed students’ understanding of Boolean logic and required predicting outputs of expressions with Boolean operators. 6b assessed the Boolean OR operator, specifically. Only about half the students answered this item correctly (Table 4). Also, we observed in this item and other similar items involving Boolean operations with picture images that some students mistakenly think that **the OR operator evaluates to true when one of its operands is true, but not both. This is similar to the use of the word ‘or’ in the English language**, which is effectively closer to the way the ‘exclusive OR’ or ‘XOR’ operator works.

Table 4. Coded student responses for Item 6b

Correct (OR)	AND	XOR	Other	Missing
46.7%	6.5%	6.5%	39%	1.3%

Student performance on the assessment items generally matched teachers’ perceptions about how students would fare on them. The teachers believed that majority of their students would get Items 2 and 6 correct since they had spent some instructional time on helping students understand what a given loop does, the sequence performed by a program containing things inside as well as outside a loop, and how to evaluate a Boolean expression. However, the teachers acknowledged that they had not spent any instructional time on describing how a variable changes values within a loop, identifying repeating patterns within a loop, describing the structural components of a pattern, or identifying a pattern from a real-world phenomenon, and hence anticipated that a majority of their students would get Items 4 and 8 incorrect.

5. DISCUSSION

Based on student responses on the pilot study as well as in the think-alouds, we revised our assessment items, especially those that were open to misinterpretation. We describe only some key improvements here. We discarded an item where evaluating expressions involving Boolean operators depended on characteristics of objects that could be interpreted variably. For Q2 we reworded the item so that the actions inside and outside the loop said different things, while retaining the nub of the question, as it appears to be valuable in revealing a misconception of how loops work. We have renamed the variable called ‘Number’ in Q4 to ‘Counter’ and replaced SkipNumber with the number 2. We

have revised Q8a,b,c to provide the 5 options (A/B/C/D/E) along with a “Mark ALL that apply”. We have removed Q8d. We have revised Q10 (not discussed here) to include a sub-question that requires a terminating condition to be constructed using a “repeat-until” so that we can address additional FKSAs (e.g. FKSA 3, 15, 16)). We have also created a new item to map to FKSA 9 that was not addressed by the earlier set of assessment items.

Revisions notwithstanding, piloting these assessments aligned to the learning goals that map to the K-12 CS framework revealed that there are several aspects related to developing and understanding of VEL concepts in the context of block-based environments that need to be consciously taught as part of K-12 CS. Ours was a pilot study in classrooms with a group of well-meaning teachers who have only recently started teaching CS in schools that have taken the bold step to adopt CS. The thrust of this discussion is not to take issue with the particular teachers or curriculum in these classrooms, but to highlight that students harbor misconceptions related to relevant CS concepts that they bring into CS courses and these will impact their CS learning unless consciously addressed. It is also obvious that though block-based programming environments such as Scratch have made it easier for novices to construct programs, several misconceptions reported in earlier literature still exist *as do others that do not appear to have been reported on before*. Students don’t have a deep understanding of how loops work, what variables are, and what they do in a programming context. Meaningful use of variables appears to be rare in middle school students’ programs and curricula, and students do not encounter loops with variables and expressions. It is our belief that these topics ought to be covered in order for students to understand the idea of data abstraction, and these concepts need to be addressed more deeply rather than have students move through the grades with misconceptions and partial understanding of these concepts.

5.1 Implications for Pedagogy

Several prior studies on addressing students’ misconceptions have focused on the development of a mental model of how the program is executed by the computer through tracing and/or visualizations. Though block-based languages such as Scratch have features such as a variable inspector, learners still struggle with understanding this key concept. We also believe that since students don’t have to deal with variables data types in block-based environments, they end up with an incomplete understanding of expressions and operators (e.g. the fact that arithmetic operators make sense only with numbers even though the variable may be a non-number; or that using “join” with 2 numbers is really concatenating 2 strings (of numbers), are just a couple of the many issues related to this). The downside of open-ended constructionist pedagogies underlying Scratch is that they require focused effort from the teacher or curricular activities to address these foundational concepts in introductory programming. CS teacher PD needs to address this, and curriculum designers must take on the onus of including activities and pedagogies aimed at better conceptual learning. This resonates with the view that K-12 CS curricula must balance constructionism with other pedagogical approaches that foster deeper learning of problem solving and computing concepts [8,15].

Activities that require students to describe what is happening in each iteration of the loop (including tracing variable values) will help them understand how sequences of actions inside loops are repeated and how to use variables and expressions in the context of loops. Students must understand the concept of “variation”, that a variable can take on different values at various points in the

execution of the program (but can only hold one value at a time); that the appropriate range is determined by the context of the program and the variable “type” (numbers, strings, Boolean); that the operators on the variables vary by data type; and how the use of Boolean operators in CS is different from the everyday use of the terms. Apt naming of variables needs to be encouraged as a means to build a better understanding of variables [19]. A meaningful variable name that conveys the variable's function or role in the program has been shown to simplify the programmer's task [21]. Lastly, regularly measure student understanding of these constructs and concepts through formative assessments.

6. CONCLUSION

Middle school students are beginning to get an introduction to CS and programming. This paper and research invokes decades of prior research on the difficulties students have in understanding foundational ideas necessary for conceptualizing and semantically composing a computational solution. Building on this prior literature, this research highlights two critical aspects that K-12 CS educators and curriculum designers will benefit from keeping in mind as they teach or design curricula. The first is that even though it is syntactically easier to put together programs in block-based programs, the conceptual difficulties in understanding and using key building blocks of programs such as variables and loops still persists, despite efforts by designers of environments like Scratch to make the environments work with variables, to use control structures such as loops, create Boolean or arithmetic expressions, or use Boolean logic to make a true/false determination of a condition. All these affordances have only helped learners with the syntactic aspects of programming, and not the semantic/conceptual (nor strategic) aspects of programming. In order for learners to understand these concepts, additional effort and pedagogic strategies are needed. Secondly, more than ever, this points to a need for formative and summative assessments designed to measure student understanding, including misconceptions, and also to refine pedagogy and curricula. This has been pointed out as an imperative to scaling up CS in K-12 [2, 8, 22]. Our current research involves developing pedagogical strategies that address these issues and recommendations, as well as formative and summative assessments. Future work entails testing the revised assessments with a group of about 300 students and conducting more robust analyses of item difficulty and reliability. For CSForAll to succeed, we need to measure conceptual understanding through various types of assessments including the kinds described here and also address key barriers to understanding foundational programming concepts such as variables and loops and related constructs such as expressions and Boolean logic.

7. ACKNOWLEDGMENTS

We acknowledge grant support from NSF (Award #1543062). Thanks to Daisy Rutstein for key contributions in this work, and to the teachers and students who participated in this pilot study.

8. REFERENCES

- [1] Almstrum, V. L. 1999. The propositional logic test as a diagnostic tool for misconceptions about logical operations. *Journal of Computers in Mathematics and Science Teaching*, 18, 205-224.
- [2] Cooper, S., Grover, S., Guzdial, M., & Simon, B. (2014). A Future for Computing Education Research. *Communications of the ACM*. 57 (11), 34-36.
- [3] duBoulay, B. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1), 57-73.
- [4] Ebrahimi, A. 1994. Novice programmer errors: Language constructs and plan composition. *International Journal of Human-Computer Studies*, 41(4), 457-480.
- [5] Gobil, A. R. M., Shukor, Z., & Mohtar, I. A. 2009. Novice difficulties in selection structure. In *International Conference on Electrical Engineering and Informatics*, 2, 351–356. New Jersey, USA: IEEE Computer Society.
- [6] Goode, J., Chapman, G., Margolis, J. 2012. Beyond Curriculum: The Exploring Computer Science Program. *ACM Inroads*. 3(2).
- [7] Grover, S., Cooper, S., & Pea, R. 2014. Assessing computational learning in K-12. In *Proceedings of the conference on Innovation & technology in computer science education*. ACM.
- [8] Grover, S., Pea, R., Cooper, S. 2015. Designing for Deeper Learning in a Blended Computer Science Course for Middle School Students. *Computer Sc. Education*, 25(2), 199-237
- [9] Herman, G. L., Loui, M. C., Kaczmarczyk, L., & Zilles, C. 2012. Describing the what and why of students' difficulties in boolean logic. *ACM Transactions on Computing Education*, 12(1), 3.
- [10] Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. 2005. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, 37(3), 14-18.
- [11] Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M. and Rusk, N. 2008. Programming by choice: Urban youth learning programming with Scratch. *ACM SIGCSE Bulletin*, 40, 1.
- [12] Mayer, R.E. (1989). The psychology of how novices learn computer programming. In E. Soloway & J.C. Spohrer (Eds.), *Studying the novice programmer* (pp. 129–159).
- [13] Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. 2011. Habits of programming in Scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 168-172). ACM.
- [14] Mislevy, R., Steinberg, L., & Almond, R. 2003. Focus article: On the structure of educational assessments. *Measurement: Interdisciplinary research and perspectives*, 1(1), 3-62.
- [15] Passey, D. (2016). Computer science (CS) in the compulsory education curriculum: Implications for future research. *Education and Information Technologies*, 1
- [16] Pea, R., & Kurland, D. 1984. On the cognitive effects of learning computer programming. *New Ideas In Psychology*, 2, 137–168.
- [17] Postner, L. “Computer science education research on programming: What we know and how we know it”, Technical Report. Online. Internet. [August, 2001]. Available WWW: <http://depts.washington.edu/pett/papers/>
- [18] Robins, A., Rountree, J., & Rountree, N. 2003. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137-172.
- [19] Samurcay, R. 1989. The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. In E. Soloway and J. C. Spohrer, editors, *Studying the Novice Programmer*, po. 161–178. Lawrence Erlbaum Associates, NJ.
- [20] Shneiderman, B., & Mayer, R. (1979). Syntactic/semantic interactions in programmer behavior: A model and experimental results. *Intl. Journal of Comp & Info. Sciences*, 8(3), 219-238.
- [21] Spohrer, J. C., & Soloway, E. 1986. Novice mistakes: Are the folk wisdoms correct?. *Communications of the ACM*, 29(7).
- [22] Yadav, A., Burkhart, D., Moix, D., Snow, E., Bandaru, P., & Clayborn, L. 2015. Sowing the Seeds: A Landscape Study on Assessment in Secondary Computer Science Education. *Comp. Sci. Teachers Assn.*, NY, NY.