# What We Can Learn About Student Learning From Open-Ended Programming Projects in Middle School Computer Science

Shuchi Grover
ACTNext, ACT
Iowa City, IA 52244
shuchi.grover@act.org

Satabdi Basu
SRI International
Menlo Park, CA 94025
satabdi.basu@sri.com

Patricia Schank
Digital Promise
Redwood City, CA 94063
pschank@digitalpromise.org

## ABSTRACT

Block-based programming environments such as Scratch, App Inventor, and Alice are a key part of introductory K-12 computer science (CS) experiences. Free-choice, open-ended projects are encouraged to promote learner agency and leverage the affordances of these novice-programming environments that also support creative engagement in CS. This mixed methods research examines what we can learn about student learning from such programming artifacts. Using an extensive rubric created to evaluate these projects along several dimensions, we coded a sample of ~80 Scratch and App Inventor projects randomly selected from 20 middle school classrooms in a diverse urban school district in the US. We present key elements of our rubric, and report on noteworthy trends including the types of artifacts created and which key programming constructs are or are not commonly used. We also report on how factors such as students' gender, grade, and teachers' teaching experience influenced students' projects. We discuss differences between programming environments in terms of artifacts created, use of computing constructs, complexity of projects, and use of features of the environment for creativity, interactivity, and engagement. Our findings will help educators of introductory computing be more cognizant of how best to leverage the programming environments they are using, and what aspects they need to focus on as they attempt to address the learning needs of all in "CS For All."

**ACM Reference format:**

Shuchi Grover, Satabdi Basu, and Patricia Schank. 2018. What We Can Learn About Student Learning From Open-Ended Programming Projects in Middle School Computer Science. In *SIGCSE '18: 49th ACM Technical Symposium on Computer Science Education, Feb. 21–24, 2018, Baltimore, MD, USA.* ACM, NY, NY, USA, 6 pages. https://doi.org/10.1145/3159450.3159522

## 1 INTRODUCTION

Learning to program is central to most computer science (CS) curricula in secondary schools in the US and internationally.


*SIGCSE '18, February 21–24, 2018, Baltimore, MD, USA*
 ACM 978-1-4503-5103-4/18/02…$15.00
https://doi.org/10.1145/3159450.3159522

Block-based programming environments such as Scratch, Alice, and MIT App Inventor, among others, have become the platforms de jure for teaching computational thinking (CT) and introductory computing concepts related to programming and algorithmic thinking [3]. These environments help motivate beginners through ease-of-use and features that support the development of creative computing projects. Creation of open-ended, free-choice programming artifacts is being used extensively as culminative "performances" of student learning, but finished open-ended projects of choice don't always tell the whole story of student learning, for a few reasons: (1) students' projects may not showcase all that they have learned; (2) students may have received help and used code/constructs that they don't understand well [14]; and (3) finished projects don't show important aspects of learning that can be understood only from in-process evidence (e.g., how they debugged or approached the construction of a computational solution) [11]. Hence, recent research has called for such projects to be used in conjunction with various other forms of assessment to get a holistic picture of student learning [8]. That said, "meaningful" open-ended projects are important since children learn deeply when they create products that require understanding and application of knowledge [2]. Design activity involves stages of revisions as students define, create, assess, and redesign their products, and it often benefits from collaboration between students [10]. Thus, students' choice-driven final projects constitute an important, authentic form of assessment. Examining such artifacts bears the potential of providing valuable insight into the student learning experiences in introductory programming, and can also point the way toward areas for improvement in introductory programming curricula.

So what can we glean about student learning and introductory programming experiences from these artifacts? More specifically, what can we understand about students' use of algorithmic constructs and other affordances provided by the programming environment? What can we learn about CT practices such as modularity and abstraction? Are there differences by gender and ethnicity in the types of artifacts students create when given free choice? Do these artifacts vary based on the teachers' CS teaching experience, programming environment used, and whether students work in pairs or individually to create them? This research effort empirically investigates such questions by examining about 80 free-choice projects created by 6th, 7th, and 8th grade students as part of a middle school CS program recently launched in a large, diverse, urban school district in the US. This

paper describes an extensive rubric that was informed by past research, recent CS frameworks and standards, and by our preliminary analyses in this effort. The rubric qualitatively evaluates the projects along various criteria: design mechanics; computing constructs and programming features such as methods, variables, lists; and overall indicators such a type of artifact created, program bugginess and termination, and the complexity and creativity of the artifact created. We present descriptive statistics by grade, gender, and programming environment, as well as correlational analyses between students' project scores and variables such as teacher experience and paired versus individual projects. Our findings have implications for teaching, curriculum design, and assessment of introductory CS and programming in middle school CS, as described in the discussion section. We believe our findings provide valuable insights to educators around the country implementing CS in K-12 classrooms. Additionally, future efforts can apply program analysis techniques to automate detection of elements of our exhaustive rubric (https://goo.gl/SEpY51).

## 2 RELATED WORK

Researchers in the past have examined the frequency of use of several CT concepts such as event handling, loops, conditional statements, data abstraction into variables, parallelism, Boolean logic, and random numbers in Scratch programs created by urban youth in after-school settings [15]. They found these numbers to be much higher for simple CT constructs and progressively lower for more complex concepts and constructs such as conditionals, Boolean logic, variables and random numbers. In another empirical study involving 5th graders, it was found that students used more Boolean expressions, conditions, and loops than variables and events in the game projects they programmed using Scratch than other types of artifacts [4]. Others have used computational techniques to examine a few thousand projects submitted to the informal Scratch online community to study frequency of construct use in addition to computational participation [6]. They found that levels of participation did not necessarily suggest depth of engagement with advanced computing concepts. Similarly, millions of apps built by students using MIT App Inventor have been examined computationally to study program complexity using frequency or presence/absence of blocks as measures of CT [22].

However, holistic code evaluation cannot rely solely on frequency counts of constructs, as several student-created programs in environments such as Scratch are inefficient, ill-constructed, fragmented and use many more blocks than are necessary [18,19]. Few research efforts, if any, have created and/or shared rubrics that go beyond counting frequency of code constructs and can be used for evaluating student projects along other dimensions. We base our work on a rubric designed and used by Grover [10] to analyze middle school students' final Scratch projects at the end of an introductory CS and programming course. That rubric had in turn been inspired by and adapted from [17] who used the rubric to grade game projects in Agentsheets, and separated scoring criteria into various type of programming and design elements.

## 3. METHODOLOGY

This research on examining student projects was guided by the following research questions: What do open-ended projects of choice tell us about student application of introductory programming concepts & CT practices? What can we learn about students' use of features afforded by block-based programming environments? How do these programming artifacts differ based on factors such as gender and grade of students, programming environment used, and teaching experience of CS teachers?

### 3.1 Data Measures

Data were collected from 20 middle schools (6th, 7th & 8th grade classrooms) in a large urban school district in Western US. (One teacher did not respond to the data request.) These CS classrooms generally reflected the diverse student population of the district—55% male, 45% female; 10% African Americans, 30% Asian Americans, 28% Latinos, 13% Whites, 14% ELL and 11% students with special needs. The district was in its 2nd year of middle school CS implementation, offering 2 levels of CS courses to accommodate year 1 and year 2 CS learners: a first level course using Scratch and a second level course using App Inventor (AI). These courses typically run for 9 or 12 weeks. AI was taught to those 7th & 8th grade students who had taken the year 1 (Scratch) CS course the previous year. Teachers were asked to randomly select 4 open-ended free-choice projects created by students in their classrooms, 2 by female students and 2 by male (so that we could run gender-wise analyses). Teachers who taught AI provided AI projects instead of Scratch projects. In some cases, those teachers who taught both courses provided a mix of AI and Scratch projects. The data we analyzed included the following:

- 60 Scratch and 20 AI projects with no personally identifiable information. (One AI project was dropped from the sample due to incompleteness resulting in **n=79**).
- Survey forms filled by teachers containing meta-data associated with each project
    - Individual/pair project
    - Grade, gender, ethnicity of student(s)
    - Teacher's years of experience as teacher & CS teacher
    - Amount of support provided on project (4-point scale)
    - Time given to complete the project.

Analyzing the metadata revealed that the grade-wise breakdown of our projects dataset was 44% from 6th, 39% from 7th, and 17% from 8th grade. Ethnicity of student creators was reported for 63 projects—49% Asian American, 22% Latina/o, 13% African American, 13% Caucasian and 3% Pacific Islander. Most students worked on their projects for 3-5 days, while some students were given more time to finish their projects. Also, 68% of projects were done individually (solo) and 32% in pairs.

### 3.2 Rubric-based Coding and Analyses

Student projects were analyzed in depth using elaborate project rubrics (one for Scratch projects and a similar one for AI projects) that were designed to provide a fine-grained view of students' projects across various facets. The rubrics were first adapted from past scholarly work [10]. We then included relevant elements guided by newly defined national K-12 CS framework (k12cs.org) and CS standards [5]. The rubrics were designed to evaluate student work across five dimensions: general factors, design mechanics, user experience, basic coding constructs, and advanced coding constructs (Table 1). They were also designed with an eye toward providing commentary that would help teachers with feedback they could use to focus on particular aspects of programming in these programming environments and instill in students better CT practices and habits of programming. For each criterion within a dimension, rubric-based

coding/scoring was based on one of the following: (1) determinations (yes/no) about the existence of a given criterion (e.g., Does the program terminate at some point? Are instructions provided for interactive elements? Do variables have meaningful names?), (2) count of the number of times a criterion (e.g., coding constructs) was properly used (some with ceilings/caps to account for the fact that code segments are often repetitive), (3) score along a 3-point scale (e.g., many bugs, few bugs, no bugs), and (4) coding for type of project (game or story or app), types of variables used, etc. In projects that included several sprites with identical or near-identical code, we took into account code associated with only one sprite. We did not examine program termination for AI projects since those programs stop when the user explicitly terminates the application. Three scorers coded a total of 60 Scratch projects. They first chose 12 projects and graded that set to refine the rubric and test inter-rater reliability. After scores were compared and discussed, the rubric was refined, and scorers distributed the remaining set of projects between themselves for grading purposes. While grading the remaining projects, scorers met periodically to discuss questions about scoring
of particular projects against the rubric, which was further refined as a result (and previously graded projects were re-scored, if necessary). A similar process was adopted for scoring the AI projects; two (of the three scorers) scored 19 AI projects. Note that although we coded for code complexity and creativity (using program novelty and engagement as proxies for creativity), we felt that these were too subjective. Consequently, our results do not report on these aspects.

**Table 1. Rubric dimensions with examples of criteria within rubric dimensions for Scratch and App Inventor projects.**

| Rubric Dimension | Examples of Criteria within Dimensions for Scratch projects | Examples of Criteria within Dimensions for App Inventor projects |
|---|---|---|
| General factors | Judgments of engagement, novelty, complexity, bugginess; termination at some point. | Judgments of program engagement, novelty, complexity, bugginess; Meaningful labeling of UI components associated with code |
| Mechanics of design | Number of unique sprites, collision detection, keyboard input, scene change; existence of user-controlled navigation, random motion of sprites. | Use of levels and modes; Collision and edge detection; Use of keyboard inputs for user interactivity; Enabling and disabling components; Drawing using the pen; Number of screens with code; Existence of user-controlled navigation and random motion of sprites; Use of files and storage; Use of sensors and text-to-speech features; Use of connectivity features. |
| User experience | Use of sound, user-created media, use of special effects, getting user input, providing use instructions for interactive elements. | Use of sound, camera, user-created media; Use of special effects; Getting user input; Getting variable values to display; Number of unique UI components, drawing and animation components, invisible components, unique layouts used; Providing user instructions for interactive elements. |
| Basic coding and constructs | Initialization (of variables and of state), variables defined and used (through set/change); Use of simple loops (forever, repeat), use of operators (arithmetic, Boolean, relational), conditionals (IF-THEN), broadcast, methods defined. | Global and local variables defined and used; Setting and getting variable values; Initialization of program state; Use of simple loops; Use of operators (arithmetic, Boolean, relational); Use of conditionals (if/then, if/then/else, else-if); Use of the random number generator. |
| Advanced coding constructs | Use of lists, clones, nested conditionals, variable or advanced loops (Repeat Until loops) or Waits that required a Boolean condition, methods called more than once. | Use of lists, nested conditionals, 'for' and 'while' loops, nested Boolean expressions, procedures defined and used, advanced math functions; Validation of user input (if needed). |

**Table 2. Types of computational artifacts created by grade, gender, and programming environment**

| | | Scratch (n=60) | | | App Inventor (n=19) | | |
|---|---|---|---|---|---|---|---|
| | | Games | Stories | Apps | Games | Stories | Apps |
| Analysis across all students | | 60% | 30% | 10% | 58% | 21% | 21% |
| Analysis by grade | 6th grade | 71% | 26% | 3% | n/a | | |
| | 7th grade | 44% | 28% | 28% | 54% | 23% | 23% |
| | 8th grade | 50% | 50% | 0% | 60% | 20% | 20% |
| Analysis by gender | Female | 52% | 31% | 17% | 45% | 22% | 33% |
| | Male | 68% | 29% | 3% | 70% | 20% | 10% |

## RESULTS

Table 2 describes the types of computational artifacts created by grade, gender, and programming environment. Our findings showed approximately the same percentage of students programming games across Scratch and AI datasets; however, a much higher percentage of utility apps were coded in AI compared to Scratch, and fewer stories were coded in AI than in Scratch. This is not surprising, given that App Inventor projects are mostly designed to be mobile apps, and mobile apps are generally built more for games and utility functions than for storytelling. *Also, in both the Scratch and AI programming environments, female students created significantly fewer games than male students and created a higher proportion of stories and utility applications compared to the male students.*

***Broad trends observed in Scratch projects.*** 83% of Scratch projects had few or no bugs, and 55% terminated at some point. 85% of projects correctly used sprite motion, collision detection, screen changes, and keyboard input for interactivity. All variables had meaningful names, and simple constructs (forever loops, broadcasts, IF-THENs) were correctly implemented in 85% projects. 60% projects used some special effects (e.g., show/hides, random/guided motion, and changing costumes for effect), and

45% were personalized with music and media. 17% projects demonstrated difficulty in configuration (e.g., forgetting to initialize or properly terminate, which was a common source of bugs) and 55% lacked evidence of use of advanced constructs (such as Repeat Until loops, procedures, nested conditionals and loops, compound Boolean operators, variables changing values in loops). 59% projects used the 'forever' loop, while 33% used the 'Repeat' block, and only 19% used the 'Repeat Until' block. 62% projects employed the IF-THEN conditional construct at least once. Only 12% used an IF- THEN-ELSE construct, and barely any used nested IFs. Only 23% projects used a Boolean expression, and just over 6% used Boolean AND/OR operators. Only 7% of Scratch projects used user-defined methods.

*Salient findings from correlational analyses between Scratch scores and Scratch project metadata:*

- Positive correlation between project scores and teacher's years of experience ($r = 0.28$, $p = 0.03$).
- Time spent on project did not have a significant effect on project scores ($r = 0.1$, $p > 0.05$).
- Even after controlling for level of support, there was no significant difference in students' project scores by grade (ANCOVA: $F(2,56) = 1.71$, $p = 0.19$).
- No statistical difference ($p = 0.29$) between total project scores by gender ($n = 31$, Males: mean = 77.13, SD = 44.21; Females: $n = 29$, mean = 66.17, SD = 34.79).
- Though the sample distribution by ethnicity was skewed, Asian American students outperformed students from other ethnicities, while Latino/a students underperformed. Difference in scores by ethnicity persisted even after controlling for time on task and teachers' support levels.
- Pair projects ($n = 15$, mean = 95.87, SD = 43.6) scored significantly higher than individual projects ($n = 45$, mean = 63.82, SD = 35.72).

There was no significant difference in the project scores between grades, but we found **improvement in correctness of program from 6th through 8th grade**, as seen in Figure 1. None of the 8th-grade students had programs with many bugs, and a higher percentage had programs with no bugs. Also, program initialization improved considerably from 6th through 8th grade. The percentage of students who initialized their entire programs correctly increased from 6th to 8th grade, while the fraction of students who did not initialize or incorrectly initialized their programs or initialized a part of their programs correctly decreased from 6th to 8th grade.

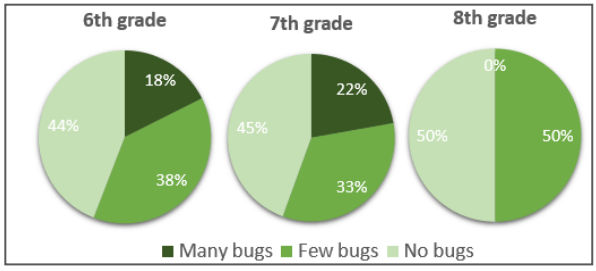


**Figure 1. Bugginess of Scratch projects by grade**

***Broad trends observed in AI projects.*** Overall, there was considerable diversity among the student projects, which ranged from very simple stories or (built-in) text-to-speech apps to collision detection games of various complexities, more complex utility applications, and games with complex interface layouts. Most projects had few or no bugs and initialized the program state correctly. About half the students (47%) demonstrated correct use of conditionals, while close to 70% of the students demonstrated appropriate use of operators and expressions. About one-third of the students used some form of advanced media, a sensor like an accelerometer, or connectivity features like Bluetooth or connections to web pages, but only a fifth of the student projects used procedures in their programs.

Salient findings from correlational analyses between AI scores and AI project metadata: The small sample size ($n = 19$) made correlational analyses of project scores with various metadata factors difficult.

Our analysis revealed no differences in students' AI project scores based on gender, grade level, ethnicity, or type of collaboration (individual or pair project). Although there was no statistical difference between AI project scores of 7th-grade students (mean = 79.54, SD = 46.01) and those of 8th-grade students (mean = 89.6, SD = 42.35), 8th-grade students scored slightly higher than 7th-grade students on almost all the rubric dimensions. Also, we found an improvement in students' program correctness and abilities to correctly initialize their AI programs from 7th to 8th grade.

Lastly, we found that the minimum project score was much higher for students who worked in pairs compared to students who worked individually. The scores were also more homogeneously distributed for students who worked in pairs. The low sample size may have affected our ability to see stronger trends.

***Comparing Scratch and AI projects.*** AI projects had fewer bugs than Scratch projects (Fig. 2). Overall, AI projects also had higher mean and median project scores than those for Scratch projects, perhaps suggesting year-on-year growth in student learning (since AI students had taken a Scratch based CS course the previous year). The use of methods was low in general across all projects, but more so in Scratch than AI projects. 95% of the Scratch projects contained no methods while the remaining 5% used more than one method. On the other hand, 79% of the AI projects contained no methods, and 16% used more than one method. This may be because the tutorials provided with AI that were used by teachers provided examples of use of methods. The use of operators was also significantly higher in AI projects (mean=5.8) compared to Scratch projects (mean=2.7). Students using AI were better able to combine arithmetic and logical operators to create expressions for updating values and checking conditions. They also used operators and expressions for calculating the placement of objects on the canvas. Differences were also observed in the use of conditionals between Scratch and AI (Fig. 3). The higher use of simple conditionals in Scratch could be due to the fact that Scratch allows users to use simple IF-THEN constructs with sensing blocks like "touching __?" and "key __ pressed?" without having to create expressions using operators. However, in AI, conditional statements were generally associated with an expression using one or more relational and mathematical operators. It is interesting to note that though most projects used Forever loops and a few used Repeat and Repeat Until loops, none of the AI projects used a While or other loop construct.

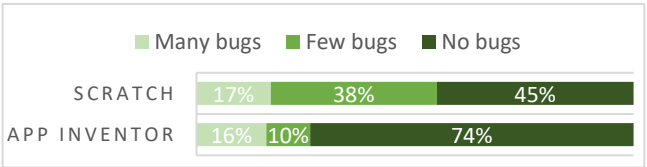


**Figure 2. Program correctness in Scratch and AI projects**

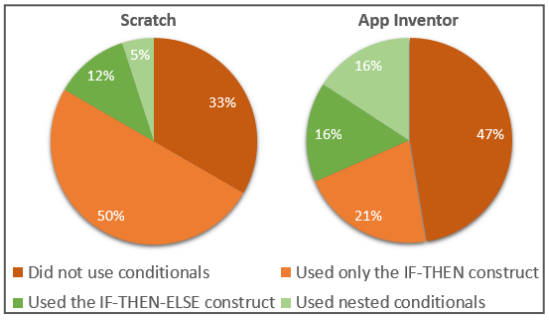


**Figure 3. Use of conditionals in Scratch and AI projects**

Students' Scratch projects used more variables on average compared to AI projects. 48% (29 of 60) Scratch projects used variables and 27% used more than one variable. In contrast, only 42% (8 of 19) of AI projects used variables and 16% used more than one variable. We believe that was due to the fact that students using AI often referred to the values of UI components to set and update their values, instead of explicitly creating variables. All these differences suggest year-on-year growth in CS learning (since AI was used by older students on average) as well as differences in affordances (and curricula) of AI versus Scratch.

## 4. DISCUSSION

This research was motivated by a need to understand what we can glean about student learning in a district implementation of CS in middle grades through an examination of authentic, free-choice artifacts students created at the end of 9 (or 12) weeks of introductory CS learning. Overall, we found that the performance of students as evidenced in Scratch and AI projects was impressive, especially taking into account the fact that these are early years of CS in the middle grades in the district, and the first year that AI was taught in 7th and 8th grade classrooms. Projects ranged from very simple ones with one or few sprites to extremely complex ones with numerous sprites, outstanding special effects, and multiple levels of play. Although not reported as part of findings, overall, student projects were creative and engaging.

In terms of growth between Year 1 and Year 2 CS students, students who built their projects using AI had a higher degree of program correctness than students who used Scratch to build their projects. Program initialization, a key concept in programming, improved considerably from 6th through 8th grade for both Scratch and AI projects. Finally, the use of operators and expressions was significantly higher in AI projects than in Scratch projects. Overall, AI final projects had higher mean and median project scores than those for Scratch final projects.

However, student projects did not provide evidence for a robust understanding of some other key concepts, such as variables, operators, advanced loops, and Boolean logic; and abstracting data and functions using variables and methods. This mirrors findings from earlier research in informal settings [6,15].

Though perhaps not surprising, it was disheartening to find significant differences by ethnicity. However, it was heartening to find no significant differences in project scores by gender. Also, interesting were the gender-wise preferences in the choice of programming artifact created. Stories and apps were created mostly by females and games created mostly by males. The gender difference was starker for apps than for stories. Women's leaning toward "utility" and "helpful" uses of CS has been well-documented at the college level [16]. It is interesting to see such proclivity in middle school as well. The mean project score for pair projects was higher than that for individual projects for both Scratch and AI projects, although the difference was statistically significant only for Scratch projects. Although this may seem like an obvious outcome, it helps to see this emerge from empirical analysis of projects. Such findings could be used to convey to teachers and students skeptical of collaborative work the value of students working in pairs (at least for some percentage of projects) in introductory CS classrooms.

Additionally, we believe that our tools for learning shape our thinking, and consequently how and what we learn from them [7]. For example, while the emphasis on storytelling in Storytelling Alice has been instrumental in introducing young girls to computing [21], in a quantitative analysis of over 300 student projects created using Scratch and Alice, it was found that while Scratch was used to program mainly games and music videos, Alice was used primarily for storytelling projects. These different types of projects motivated students to use quite different computing concepts [1]. "Students creating games used the most variables, if statements, and loops. Students creating music videos used nearly as many loops as in games, but far fewer variables and if statements. Students creating story-telling projects used the fewest loops, variables, and if statements" [1, pg. 648]. We found interesting differences in the use of constructs in Scratch and AI projects––a finding that suggests that teachers and curriculum designers should be mindful of the inherent affordances of different novice block-based programming environments that exist by virtue of the design and CS learning philosophies embedded in these environments [13]. These affordances, or the lack thereof, should be leveraged or compensated for in the curriculum. Lastly, how instruction and curricula attend to aspects of coding and constructs impacts student learning––and teachers play a substantial role in what and how students learn [10,20].

***Threats to Validity.*** While our findings are insightful, it is worth pointing out that the process of establishing interrater reliability involved joint discussions of 12 Scratch projects and 5 AI projects coded by 3 coders, and we did not apply the rubric to each other's projects. Furthermore, the total sample was not very large—it comprised 60 Scratch and 19 AI projects.

### 4.1 Implications for Pedagogy

The lack of evidence of use of more advanced concepts such as Boolean Logic, variables with loops, and methods appears to be a recurring observation in students' programming artifacts created in informal settings [15] as well as in this work. This may have been caused by factors besides a lack of understanding, such as the time students got to work on the final projects, or students being motivated more by their project idea and getting their code to work rather than using more appropriate (advanced) concepts. That being said, the lack of use of these constructs across projects from 20 classrooms does point to a lacuna that likely needs attention. We believe middle school curricula *must* pay more attention than they currently do to certain important computing ideas that have been called out by recent K-12 CS framework and standards work.

***Boolean logic*** is a key part of understanding the logic of computers and programming. This topic has been called for in the K–12 National CS Framework as well. We believe that learning this topic in 7th or 8th grade (if not before) will help students build programs that have a deeper level of complexity, and also get some sense for how the 0/1 or True/False state of bits gets combined at the machine level.

Thinking about code modularization and ***functional abstraction*** is a key CS skill that can be taught in 7th and 8th grades as part of problem decomposition which we believe is a common strategy taught to students. We also believe that the AI environment lends itself well to such instruction. However, experiencing functional abstraction in action in Scratch, perhaps at the end of the 7th-grade, would help provide a foundation for students to do this in AI in 8th grade. These approaches will address many CS teachers' concerns about teaching and assessing for abstraction.

We found use of variables to be very simplistic for the most part in both Scratch and AI projects (more so in Scratch projects). Lists were used minimally in both Scratch and AI projects. Lists and use of data structures to represent compound data types are topics that can be introduced as key training in ***data abstraction***. In general, use of variables that stand for other expressions can also be used a mechanism for using abstraction. This is a technique we are using in our design of conceptually rich curricular activities for introductory programming [12].

Additionally, the lack of use of ***Repeat Until*** blocks in Scratch projects suggests that teachers could do more to enable learners to create artifacts that require them to bring together loops, variables, and Boolean expressions. These could be ways to create a progression in the Scratch curriculum from 6th to 7th grade that involves simple (bounded) loops to more complex loops that are controlled by variables and Boolean terminating conditions. Also, students need to be instructed to be mindful about program initialization and (especially) termination. And lastly, for AI specifically, since built-in AI affordances such as accelerometer, camera, text-to-speech, and web connectivity can help students build more engaging and interesting projects, it would be beneficial to show students how to use these features and encourage them to include them wherever relevant.

## 5. CONCLUSION

This research was conducted with data from a large diverse urban school district that has recently launched CS in all grades of middle school as part of their "CS for All" efforts. We focused on understanding what trends can be observed about students' introductory programming experiences through an in-depth examination of their free-choice projects (rather than delving into the specifics of the Scratch and AI curriculum, which would be distinct from site to site). While we recognize the need for various forms of assessment to understand the many facets of student learning [9], our research highlights interesting trends about student learning and choice in introductory programming. We find that games are the most popular programming artifact created. There is gender parity in the scores students received based on our rubric-based coding, and that pair projects score higher than individual projects. Differences in the tool-provided features used in Scratch versus AI projects suggest that the inherent affordances of tools impacts what students create, and likely, learn. The lack of use of certain constructs and concepts such as Boolean logic, nested conditionals, procedures, and advanced loops echoes earlier findings in informal settings [6,15], and suggests a need for K-12 CS curricula to pay more attention to these topics that are crucial for building a more solid foundation of introductory computing concepts. Differences in projects by ethnicity reveals that much work remains to be done to address the learning needs of underrepresented minority groups.

## ACKNOWLEDGMENTS

We thank Bryan Twarek for his contributions to this research, as well as the CS teachers who provided us student projects.

## REFERENCES

[1] Adams, J.C. & Webster, A.R. 2012. 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.

[2] Barron B., & Daring-Hammond, L. 2008. How can we teach for meaningful learning.? In Daring-Hammond, L., Barron, B., Pearson, P. D., Schoenfeld, A. H., Stage, E. K., Zimmerman, T. D., Cervetti, G. N., & Tilson, J. L. *Powerful learning: What we know about teaching for understanding*. San Francisco: Jossey-Bass.

[3] Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. 2017. Learnable programming: blocks and beyond. *Communications of the ACM*, 60(6), 72-80.

[4] Baytak, A. & Land, S. 2011. Advancing elementary- school girls' programming through game design. *International Journal of Gender, Science, & Technology*, 3(1).

[5] Computer Science Teachers Association. 2017. *CSTA K-12 Computer Science Standards*, Revised 2017. Retrieved from http://www.csteachers.org/standards.

[6] Fields, D. A., Giang, M., & Kafai, Y. 2014. Programming in the wild: trends in youth computational participation in the online scratch community. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (WiPSCE'14) ACM.

[7] Grover, S. 2013. Should All Computational Tools Be Treated Equal?: Comparing The Affordances Of K-12 Computing Education Tools. *Paper presented at AERA*, San Francisco.

[8] Grover, S. 2015. "Systems of Assessments" for Deeper Learning of Computational Thinking in K-12. In *Proceedings of the 2015 Annual Meeting of the American Educational Research Association* (pp. 15-20).

[9] Grover, S. 2017. Assessing Algorithmic and Computational Thinking in K-12: Lessons from a Middle School Classroom. In *Emerging Research, Practice, and Policy on Computational Thinking* (p. 269-288). Springer International.

[10] Grover, S., Pea, R., Cooper, S. 2015. Designing for Deeper Learning in a Blended Computer Science Course for Middle School Students. *Computer Science Ed, 25*(2), 199-237.

[11] Grover, S., Basu, S., Bienkowski, M., Eagle, M., Diana, N., & Stamper, J. (2017). A Framework for Using Hypothesis-Driven Approaches to Support Data-Driven Learning Analytics in Measuring Computational Thinking in Block-Based Programming Environments. *ACM Transactions on Computing Education (TOCE), 17*(3), 14.

[12] Grover, S., Jackiw, N., Lundh, P., & Basu, S. (In review). Integrating Non-Programming Digital Interactives to Advance Learning of Introductory Computing Concepts for Diverse Student Populations.

[13] Kelleher, C., & Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR), 37*(2), 83-137.

[14] Kurland, D. & Pea, R. 1985. Children's mental models of recursive LOGO programs. *Journal of Educational Computing Research, 1*(2), 235-243.

[15] Maloney, J., Peppler, K., Kafai, Y.B., Resnick, M., & Rusk, N. 2008. Programming by choice: Urban youth learning programming with Scratch. *Proceedings of SIGCSE '08*. New York, NY: ACM Press.

[16] Margolis, J., & Fisher, A. (2003). Unlocking the clubhouse: Women in computing. MIT press.

[17] Martin, C.K., Walter, S., & Barron, B. 2009. Looking at learning through student designed computer games: A rubric approach with novice programming projects. Stanford U.

[18] Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. 2010. Learning computer science concepts with Scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research (ICER '10)*. ACM. 69-76.

[19] 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.

[20] Pea, R., & Kurland, D. 1984. On the cognitive effects of learning computer programming. *New Ideas In Psychology, 2*, 137–168.

[21] Utting, I., Cooper, C., Kölling, M., Maloney, M., & Resnick, M. 2010. Alice, Greenfoot, and Scratch -- A Discussion. *ACM Transactions on Computing Education, 10*(4), 1-11.

[22] Xie, B. & Abelson, H. 2015. *Skill Progression in MIT App Inventor*. Retrieved from http://benjixie.com/wp-content/uploads/2016/06/vlhcc_pre_print.pdf/