Measuring Learning in an Open-Ended, Constructionist-Based Progamming Camp:

Developing a Set of Quantitative Measures from Qualitative Analysis

Deborah A. Fields
Instructional Tech. & Learning Sci.
Utah State University
Logan, USA
deborah.fields@usu.edu

Lisa C. Quirke
Faculty of Information
University of Toronto
Toronto, Canada
lisa.quirke@mail.utoronto.ca

Janell Amely
Instructional Tech. & Learning Sci
Utah State University
Logan, USA
jamely@aggiemail.usu.edu

Abstract*— *In this paper we raise the issue of how to assess novice youths' learning of programming in an open-ended, project-based learning environment. One approach could be a way to apply quantitative measures to the analysis of programming education across frequent saves in a variety of open-ended projects. This paper focuses on the first stage of this endeavor: the development of exploratory quantitative measures of youths' learning of computer science concepts through deep qualitative analysis.

Keywords—computer science education; big data; Scratch; assessment; novice programmers; constructionism

I. INTRODUCTION

Since Seymour Papert introduced the idea of constructionism [1], many computer science educators have sought to engage kids in learning through interest-driven project-based programming. This playful approach to learning-by-designing emphasizes the social and contextual factors that affect learning [2]. This approach is open and free-form, allowing users to participate and collaborate in the creation of products that are meaningful to them: no project is the same [3]. Yet while this can be motivating for learners, who can bring in outside interests and representations of themselves, their families, and communities into their projects [3], it brings challenges for educators and researchers who are tasked with measuring and assessing their learning.

Though constructionist programming environments allow for diverse programming styles such as "structured" or "bricoleur" [4], there is no guarantee that all students will learn the basics of key programming concepts. As Maloney, Peppler, Kafai, Resnick and Rusk [5] showed in their two-year study of a Computer Clubhouse, few students ever began using programming concepts such as variables, conditionals, randomization, or Booleans on their own. Similarly, in their study of a random sample of 5000 active users on the Scratch online community (scratch.mit.edu), Fields, Giang, and Kafai [6] noted that only 16% of users used key programming concepts such as variables, conditionals, and Booleans in their programs. This raises an important question of how educators can deepen and broaden students' grasp of programming in ways that allow for the creativity and ownership of constructionist learning environments.

This work was supported by a collaborative grant from the National Science Foundation (NSF#1319938) to Deborah A. Fields, Taylor Martin and Sarah Brasiel. The views expressed are those of the authors and do not necessarily represent the views of the National Science Foundation, Utah State University, or the University of Toronto.

II. BACKGROUND

In recent years, many programming environments have been created to support students' learning of computer science concepts without typical challenges of spelling, syntax, and punctuation [7]. Building on the work of Logo [1], visual block-based languages such as Scratch [8] and Alice [9] enable novice programmers to focus on learning programming concepts without worrying about these other issues [10]. Visual programming environments make coding simpler by making it impossible for users to make syntax errors and providing immediate feedback. Yet despite the availability of languages that support novices' entry into computer programming, there is a dearth of research on how novice programmers, and children in particular, learn computer science concepts in these environments, especially when they are used in a constructionist manner [e.g. 11]). The studies that do exist highlight the importance of key computing concepts as well as the challenges that novices may face in mastering these.

One difficulty constructionist educators have faced is how to evaluate students' programming or more broadly computational thinking [12], especially in interest-driven environments where projects are open-ended without a "correct" solution [13]. One approach is to look for the application of specific programming concepts. Brennan and Resnick [10] argue for several computer science concepts they see as central, including sequences, loops, parallelism, events, conditionals, operators and data. They separately consider computational practices such as debugging and remixing (see also [14]) and computational perspectives including expressing, connecting, and questioning.

Many researchers have applied the idea of programming concepts in evaluating students' programs, but often these are applied in very basic ways (e.g. frequency counts), especially on a larger scale where programs cannot be individually analyzed. For instance, Maloney et al [5] and Fields et al [6] differentiate students by the mere presence of particular blocks of code. The Scrape tool developed by Wolz, Hallberg, and Taylor [15] shows somewhat more breadth in students' code by identifying every block used in a single or series of Scratch projects amongst sets of particular blocks. Still, applications of this tool generally presume that the presence of select blocks denotes students' learning of a concept. While these

approaches can differentiate between the programming patterns of hundreds or thousands) of students, they do not necessarily show that students have developed competence or mastery with a particular programming concept. Brennan and Resnick [10] highlight this point, noting the major differences found between students' projects using Scrape and their actual understanding of how their code worked.

In this position paper we discuss the issue of bringing rigor to constructionist programming environments and assessments of learning in these spaces. We share our "studio model of pedagogy" and our efforts to develop measures with which to assess student learning at a larger scale. As part of our analysis we have developed a set of measures of programming concepts including initialization, events, parallelism, conditionals, variables, randomization and Booleans. We developed these measures through deep qualitative analysis of changes in students' projects and have begun to apply them analytically across more than 600 project saves per student (roughly every 2 minutes). Below we briefly discuss the pedagogy we applied to three Scratch Camps followed by a brief introduction to some of the measures we are creating. We hope to engage in conversation at the workshop about authentically assessing students' learning while maintaining the richness of constructionist environments for creating with programming.

III. Scratch Camp: A Creative, Project-based Pedagogy

The questions examined in this position paper emerge from a study of children's learning trajectories with Scratch. The project includes data collected at three, week-long Scratch Camps held in summer 2014 at an intermountain university with the local 4-H club. Across five days and 25 hours of programming, campers aged 10-13 made a series of creative projects. Campers were mentored and taught by one professor and three graduate students.

A key inspiration for the design of our camps was a "studio model" of pedagogy [16], a prominent approach in arts instruction. This model involves three elements: (1) complex, authentic, real world projects, (2) guided problem solving with creative constraints, and (3) externalization and reflection with public feedback. In his study of art studies, Sawyer [16] found that art teachers created a series of carefully structured projects that necessitated students' learning of particular techniques. Although projects are open-ended and allow for creative expression, key constraints help to structure the project, focusing students' creative efforts on the targeted problems. Finally, open discussions and critiques of student work allowed the process to be transparent, as all students could learn from instructors' critiques and see changes in each other's projects.

It was with this theory of pedagogy in mind that we created a series of open-ended, genre-specific project challenges with constraints intended to impel students to learn particular programming techniques. In an earlier study, Fields, Vasudevan, and Kafai [17] successfully targeted the programming concepts of initialization and synchronization through a music video design challenge. We included that as one of the following projects in this series of camps: Scribble time & Name project (Day 1), Story project (Day 2), Music Video (Day 3), Video Game (Day 4), and Free Choice Project (Day 5). Scratch Camp concluded with a special gallery walk where interested parents could browse campers' completed work and attend the graduation ceremony.

IV. Example of Measures

Due to space limitations, we share just one example of some measures we have developed to see students' progress with events and parallelism over time. The graph below shows measures focused on broadcasts and use of green flags that illustrate students' use of events and parallelism in programming (see Figure 1). They show how these measures of programming stand out over the accumulated progress Virginia, age 10, made throughout the camp.

In Figure 1, the red line represents the use of working broadcasts. In essence these are "events" in Scratch—issuing a broadcast and linking it with a receive triggers a new set of commands. (Note: this measure only includes broadcasts that have a matching receive and are connected to a working script.) Looking at the graph, it is possible to see how Virginia began to use broadcasts with corresponding receives over time. The linear dotted grey lines delineate the five days of the Scratch Camp. She began to use broadcasts during the "Story" project which began at the very end of Day 1 and continued into the majority of Day 2. One of the goals of this project was to authentically introduce a reason to use broadcasts, and creating a narrative with multiple scenes successfully provided a context for that learning. Though some campers introduced broadcasts during the first hour of their story program, Virginia caught on later (we verified this with qualitative data analysis), despite multiple introductions to the concept on Days 1 and 2. The graph also shows the growing size of the projects as more broadcasts are introduced in the Story and then immediately plateau at the start of the Music Video project.

Two other measures show uses of parallelism in students' programming: the green line shows the number of broadcasts with more than one "receive" in a given project, while the blue line illustrates the number of sprites that had multiple green flags with scripts in a given project.

We found that using multiple receives with a single broadcast or using multiple green flags in a single sprite introduced a interesting level of abstraction, a form of parallelism, into students' programming. It generally took some time before students caught on to the utility of this idea of triggering multiple things to start at the same time, especially within a single sprite or with a single broadcast. Virginia began to use these in the development of her story projects almost as soon as she began to use broadcasts. Then Virginia applied this strategy in a different way in in her music video (Day 3) using a large number of green flags in many sprites, returning to this strategy with fewer sprites in her video game on Day 4. Looking at these measures as a whole shows students' progressions across using a concept in multiple projects and in different ways.

This example shows the beginnings of how we are trying to evaluate the quality of programming and trajectories of

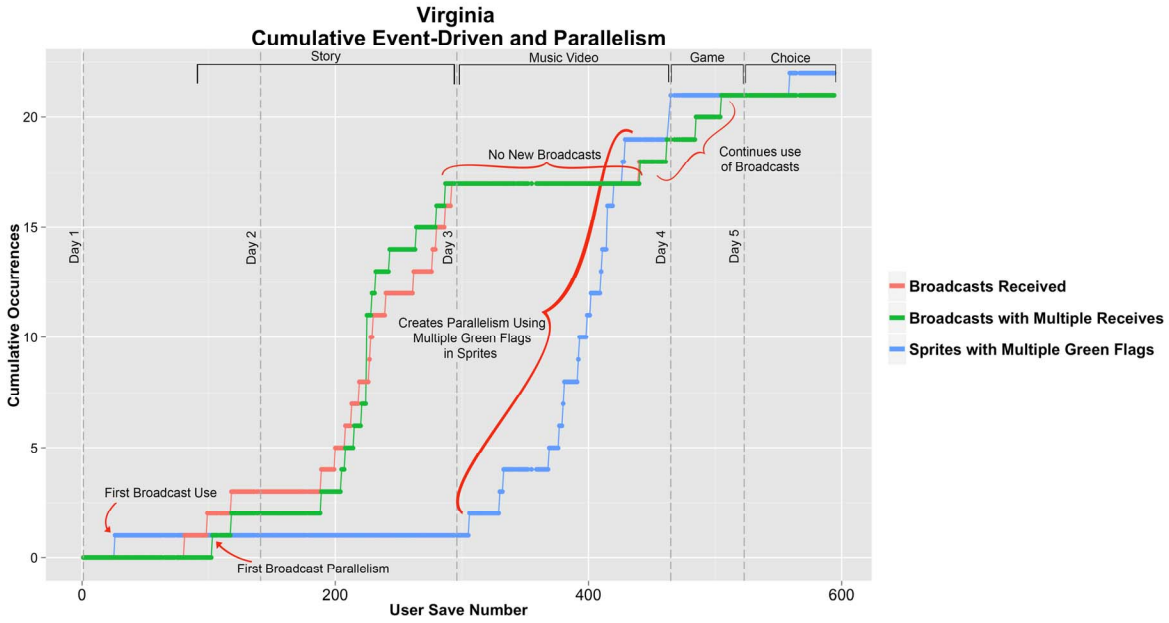

Fig. 1. Virginia's use of events and parallelism across project saved throughout the Scratch Camp.

learning in a relatively open-ended environment. Our other measures include categories of initialization, events, parallelism, conditionals, variables, randomization and Booleans. Throughout this process we have gone back and forth between studying students' learning through hand-analysis of projects (with videos and field notes providing additional context for their learning) to the development of quantitative measures. We are in the process of evaluating the quantitative measures to see which ones are most valuable, at least in the context of the projects at the Scratch Camps. This is but one approach to the challenge of finding ways to authentically trace students' learning in open-ended environments. We look forward to discussions on this issue of supporting students' interest and creativity while challenging them to deepen their skills and thinking.

ACKNOWLEDGMENT

Special thanks to the Scratch Team, Jason Maughan, Xavier Velasquez, Tori Horton, and Katarina Pantic.

REFERENCES

[1] S. Papert, Mindstorms: Children, computers, and powerful ideas. New York: Basic Books, 1980.

[2] Y. B. Kafai, "Playing and making games for learning: Instructionist and constructionist perspectives for game studies," Games and Culture, vol. 1(1), 2006, pp. 36-40.

[3] K. Peppler and Y. Kafai. "Creative coding: Programming for personal expression." In The Proceedings of the 8th International Conference on Computer Supported Collaborative Learning (CSCL). Rhodes, Greece, 2009.

[4] S. Turkle and S. Papert. "Epistemological pluralism: Styles and voices within the computer culture." Signs, 1990, pp. 128–57.

[5] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk. "Programming by choice: Urban youth learning programming with Scratch." ACM SIGCSE Bulletin, vol. 40(1), 2008, pp. 367–371.

[6] D. A. Fields, M. Giang, and Y. Kafai. "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, 2–11. ACM, 2014.

[7] C. Duncan, T. Bell, and S. Tanimoto. "Should your 8-year-old learn coding?" In Proceedings of the 9th Workshop in Primary and Secondary Computing Education, 60–69. ACM, 2014.

[8] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, et al. "Scratch: Programming for all." Comm. ACM. 52, no. 11, 2009, 60–67.

[9] C. Kelleher and R. Pausch. "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers." ACM. Comput. Surv. vol. 37(2), 2005, pp. 83–137.

[10] K. Brennan and M. Resnick. "New frameworks for studying and assessing the development of computational thinking." Paper presented at annual American Educational Research Association meeting, Vancouver, BC, Canada, April 2012.

[11] S. Grover, R. Pea, and S. Cooper. "Designing for deeper learning in a blended computer science course for middle school students." Computer Science Education, vol. 25(2), 2015, pp. 199-237.

[12] J. Wing, J. "Computational Thinking." Comm. ACM. 49(3), 2006, pp. 33-35.

[13] P. Blikstein, M. Worsley, C. Piech, M. Sahami, S. Cooper, and D. Koller. "Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming." J. Learn. Sci. 23(4), 2014, pp. 561-599.

[14] L. Murphy, G. Lewandowski, R. McCauley, B. Simon, L. Thomas, and C. Zander. "Debugging: The good, the bad, and the quirky – a qualitative analysis of novices' strategies." In ACM SIGCSE Bulletin, 40: 163–67. ACM, 2008.

[15] U. Wolz, B. Taylor, and C. Hallberg. "Scrape: A Tool for Visualizing the Code of Scratch Programs." Presented at the ACM SIGCSE, Dallas, Texas, 2010.

[16] K. Sawyer. "Learning how to create: Toward a learning sciences of art and design." In van Aalst, J., Thompson, K., Jacobson, M.J., & Reimann, P. (Eds.), The Future of Learning: Proceedings of the 10th International Conference of the Learning Sciences (ICLS 2012), Volume 1, Full Papers. International Society of the Learning Sciences: Sydney, NSW, Australia, 2012, pp. 33-39.

[17] D. Fields, V. Vasudevan, and Y. Kafai. "The programmers' collective: connecting collaboration and computation in a high school Scratch mashup coding workshop." Interact. Learn. Envir. in press.