A Framework for Using Hypothesis-Driven Approaches to Support Data-Driven Learning Analytics in Measuring Computational Thinking in Block-Based Programming Environments

SHUCHI GROVER, SATABDI BASU, and MARIE BIENKOWSKI, SRI International
MICHAEL EAGLE, NICHOLAS DIANA, and JOHN STAMPER, Carnegie Mellon University

Systematic endeavors to take computer science (CS) and computational thinking (CT) to scale in middle and high school classrooms are underway with curricula that emphasize the enactment of authentic CT skills, especially in the context of programming in block-based programming environments. There is, therefore, a growing need to measure students' learning of CT in the context of programming and also support all learners through this process of learning computational problem solving. The goal of this research is to explore hypothesis-driven approaches that can be combined with data-driven ones to better interpret student actions and processes in log data captured from block-based programming environments with the goal of measuring and assessing students' CT skills. Informed by past literature and based on our empirical work examining a dataset from the use of the Fairy Assessment in the Alice programming environment in middle schools, we present a framework that formalizes a process where a hypothesis-driven approach informed by Evidence-Centered Design effectively complements data-driven learning analytics in interpreting students' programming process and assessing CT in block-based programming environments. We apply the framework to the design of Alice tasks for high school CS to be used for measuring CT during programming.

CCS Concepts: • **Social and professional topics** → **Computational thinking**; **Student assessment**; **K-12 education**; • **Information systems** → *Data mining*; • **Computing methodologies** → *Semi-supervised learning settings*;

Additional Key Words and Phrases: Blended learning analytics, hypothesis-driven, data-driven, block-based programming environments, evidence-centered design, computational psychometrics, K-12 computer science education

ACM Reference format:
Shuchi Grover, Satabdi Basu, Marie Bienkowski, Michael Eagle, Nicholas Diana, and John Stamper. 2017. A Framework for Using Hypothesis-Driven Approaches to Support Data-Driven Learning Analytics in Measuring Computational Thinking in Block-Based Programming Environments. *ACM Trans. Comput. Educ.* 17, 3, Article 14 (August 2017), 25 pages.
https://doi.org/10.1145/3105910

This work is supported by the National Science Foundation, under grant IIS-1522990.
Authors' addresses: S. Grover, S. Basu, and M. Bienkowski, SRI International, 333 Ravenswood Avenue, Menlo Park, California 94025; emails: {shuchi.grover, satabdi.basu, marie.bienkowski}@sri.com; M. Eagle, N. Diana, and J. Stamper, Human-Computer Interaction Institute, Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213; emails: {meagle, ndiana, jstamper}@cs.cmu.edu.

https://doi.org/10.1145/3105910

1 INTRODUCTION

"Computational thinking" or CT [65] is now recognized as a foundational competency for success in STEM careers and also as a means for creative problem solving in other disciplines. In the past decade, systematic endeavors have gained momentum to take computer science (CS) education and CT to scale in K-12 classrooms in states across the United States as well as internationally. With the assertion that all children from kindergarten through high school need to learn CS and be equipped with the CT skills they need to be creators, not just consumers, and to participate in our technology-driven world, "Computer Science for All," initiated by the highest offices in the United States, now has the support of federal funding agencies, academic research institutions, professional associations, prominent industry partners, and nonprofit organizations.

Most K-12 CS curricula emphasize creativity and collaboration in computing in addition to the core disciplinary ideas of CS and the enactment of authentic CT practices in the problem-solving process. Authentic CT practices such as iterative refinement, decomposing complex problems and solutions, debugging, abstraction, and appropriate use of algorithmic concepts and constructs to create computational solutions have been called out as essential components of CS learning in the national K-12 framework for CS [32]. Learning to program is a central feature of these K-12 CS curricula. However, programming has historically been difficult for novice programmers to learn [60]. Novices can be overwhelmed by the formal syntax and need for precision inherent in programming languages. Both CS and programming have also been plagued by negative stereotypes and other misperceptions of the field [27], leading to a disproportionately low number of females and minorities [18, 20]. This has led to a rethinking of tools and pedagogies for introducing CS coursework for the novice K-16 learner. Introductory approaches to teaching programming are now attempting to lower the barrier to creating creative computational artifacts. These approaches are geared to all learners, but especially to minorities historically underrepresented in CS with the goal of making CS accessible to all.

New high school CS curricula such as Exploring Computer Science (ECS) and AP CS Principles (AP CSP), and most efforts in K-12 use block-based programming environments (BBPEs) that offer visual and direct manipulation interfaces (with immediate visualization of program effects). BBPEs support open-ended programming activity approaches for introducing learners to computational thinking in personally meaningful ways. Easy-to-use BBPEs such as Scratch, Alice, MIT App Inventor, AgentSheets, and Kodu employ the "low floor, high ceiling" guiding principle that goes at least as far back as Papert's [44] treatise on children and the LOGO programming language. These and other variants of BBPEs (e.g., Code.org's Code Studio that uses Blockly to create sequences of programming puzzles) have gained popularity for teaching CT skills in K-12 settings. Learner agency, creative expression, tinkering, and visual feedback are well supported in these environments [62] and are effective ways to keep students engaged.

These programming environments are attractive for motivating young people, but there are challenges to using them in formal K-12 settings. Although interest-driven activity remains important, more structured CS curricula aim to also consciously teach and assess CT practices, which represent the integration of problem solving, CT skills, and the disciplinary ideas of computing. Environments such as Scratch do not automatically foster deeper engagement with advanced CT constructs [13, 39]. For deeper learning [47] of CS in more structured school settings, pedagogy and curricula need to ensure that learners move from tinkering to deliberate application of CT. In order to make this transition, learners need scaffolds to productively apply CT concepts and practices, including strategies for constructing artifacts—*how* one goes about developing programs is just as important as *what* one produces [21, 31, 39].

As CS educators and researchers look for ways to support deeper learning of CT skills in formal learning settings [e.g., 26], they are hampered by a lack of understanding of the programming process of novice learners. There is a need to understand how to measure and support this development of complex CT skills in the context of programming in K-12 learning settings. Assessments and measurements of student learning remain underdeveloped and underresearched in this domain [69] and are often called out among key future CS education research imperatives [8] going forward. However, without sufficient attention to thoughtful measurement of learning, it is believed that CT can have little hope of scaling in K-12 schools [25].

Thus, students' actions need to be detected and measured *as students work* so that formative feedback can be provided to steer learning. We see great promise in bringing together multidisciplinary approaches to solving the challenge of detecting and measuring, drawing from multiple disciplines such as the learning sciences, assessment development methodologies, and learning analytics. In using emerging computational learning analytics (EDM/LA) approaches [2, 3, 53] along with hypothesis-driven ones informed by traditional assessment design methods, we are pushing into new computational psychometric methodology territory in the context of BBPEs. As such, we believe that sharing our experiences and providing guidelines will guide future work in better real-time measurements of learning for providing formative scaffolds in BBPEs to novice learners.

This article presents a preliminary theoretical framework and an instantiation of its application. The framework aims to formalize blended analytics approaches—that combine data-driven with hypothesis-driven techniques—to measure student learning and understand the programming process. We pay special attention to the need for appropriate task design as well as qualitative approaches to discern a priori patterns to look for in log data that can then be combined with patterns mined using discovery- and data-driven approaches. The article is organized as follows. Following a review of relevant literature, we describe our exploratory work to analyze and interpret log data gathered through a prior research effort from a programming task designed in Alice [9], a popular BBPE, along with manually scored programs based on a scoring rubric. Based on our findings and learning in that early phase, we describe a framework that articulates a systematic set of guidelines for how to perform hypothesis-driven analyses that can support data-driven ones for measuring student learning and understanding of CT in BBPEs. We then describe our experience in implementing the framework to guide hypothesis-driven analyses of newly designed Alice programming tasks for high school ECS [19] classrooms. A priori patterns derived will be blended with data-driven analytic approaches (described in [12]) as part of a larger research effort.

2 LITERATURE REVIEW

Programming is a complex cognitive activity that involves understanding a problem as a computational task, mapping a design for the program, drawing on problems previously programmed that have a similar structure, instantiating abstract program patterns, coding the program, and then testing and debugging [46]. Barring a few efforts [e.g., 22, 24, 39, 70], research on measuring these skills in programming has relied on examining the finished programs for use of programming constructs (e.g., loops, conditionals, and logic). This approach is incomplete because (1) the presence or absence of language constructs is not always an accurate indicator of what students know [37]; (2) students' processes of constructing programs can be better indicators of student understanding than finished products [7]; and (3) what students struggle with most is not the use of constructs, but rather the strategies and process of composing a computational solution [59]. Despite the vast body of literature on children and programming, relatively little is understood about the programming *process* [5].

EDM/LA approaches offer ways to model student concepts, misconceptions, and steps to a solution using clickstream data from digital environments. Researchers have now begun to apply

EDM/LA in more exploratory learning to model detailed learner pathways [1]. In the context of programming, EDM methods and related LA have recently been used to study patterns of behaviors in program construction, compilation, and debugging [5, 7, 8]. For example, Berland et al. [5] used clustering techniques to classify students as following either the "tinkerer" or the "planner" approach. Piech et al. [50] and Blikstein et al. [8] used EDM techniques to analyze program snapshots from CS1 students programming "Checkerboard Karel" using Karel the Robot, a programming environment for novices that provides learners with five commands to move Karel in a grid. Using purely data-driven approaches, they derived patterns of program states, examined how students transitioned through these states, developed predictors for good and bad states, and identified "sink" states from which students would only exit with great difficulty. Piech et al. [49] also studied data collected from hundreds of thousands of users working on Code.org's *Hour of Code* activities that are essentially constrained block-based programming puzzles based on the Blockly programming environment. The goal of the research was to analyze users' partial solutions to predict the hint that a teacher could provide a student about the next step. Formative work presented by Jernigan et al. [30] on a prototype "idea garden" system aimed at end-user programmers extends Gidget, an online puzzle game that focuses on debugging, and defines "antipatterns" as implementation patterns that suggest some kind of conceptual, problem-solving, or strategy difficulty.

Research on EDM/LA in open-ended BBPEs is still very nascent. In Agentsheets [51], metrics like Program Behavior Similarity and tools like the Computational Thinking Pattern (CTP) graphs have been developed to assess students' programs and provide them with semantic evaluation feedback immediately after submitting their programs [36]. The CTP graph compares CT patterns in students' programs with desired CT patterns. Kelleher [33] built an Alice logging system that distinguished actions as belonging to one of three different categories of Alice activities: programming, scene layout, or program play. Werner et al. [64] built a parser to translate low-level logging data from Alice into high-level actions that correspond more closely to semantically meaningful user actions. It is this same Alice dataset that we analyzed in the exploratory phase of our effort described in this article. Students' skill progressions and program complexity in apps built in MIT App Inventor have also been explored by examining the existence and frequency of certain blocks that denote varying levels of computational complexity [67, 68].

Most of the efforts described previously in BBPEs either use analyses of frequency or presence/absence of blocks as measures of CT, or broad patterns of behavior derived from purely data-driven analytic approaches. Few, if any, approaches provide insights into a student's understanding and application of CT while creating programs: sequences of actions they are using that could be suggestive of CT practices or how and what their use of a block or construct conveys about their understanding. Given that mere existence of a construct is not indicative of understanding, we need either evidence of repeated (and correct) use or some insight into the process actions surrounding the use of the construct. For measurement purposes, a purely data-driven analysis approach appears to fall short in providing all the necessary insights about individual students' understanding of CT from trace data to make an assessment of their CT skills.

2.1 Blended Analytics Approaches

Most of these early efforts have analyzed log data post hoc looking for constructs or patterns largely from the "bottom up" [66] using discovery-driven approaches such as clustering to provide information on patterns of program construction or use of specific actions or blocks over the course of constructing programs. More recently, however, blended approaches involving hypothesis-driven and discovery-driven analyses to examine learning processes have been proposed and used, especially in the context of interpreting user actions in open-ended learning environments (OELEs) more generally.

Hypothesis-driven analyses require that we define, a priori, patterns or even "antipatterns" to look for in data as evidence for conceptual understanding or the lack of it. Having a priori patterns lets researchers (1) explore data from smaller datasets, (2) look at datasets from different types of programming tasks, and (3) guide the iterative refinement of hypothesis generation and confirmation [56] about student actions and the learning concepts for which they provide evidence. Such techniques have been used to assess student learning in the context of game-based environments for science and math learning [e.g., 17, 34, 40, 58, 61]. The a priori patterns in these approaches are often derived from applying Evidence-Centered Design (ECD), a principled method for modeling domains for assessment purposes that is especially useful in domains with hard-to-assess concepts and practices [41, 58]. ECD is thus complementary with the analytics from discovery-driven approaches that use EDM/LA methods [40]. In ECD, an assessment is viewed as an argument from evidence: what we observe students say, do, or make in a particular circumstance is used to makes inferences about what they know, can do, or have accomplished more generally. ECD directs researchers' attention to the importance of three models and their interrelationships: a student model (what are targeted cognitive constructs supposed to look like?), a task model (what activities allow students to demonstrate those cognitive constructs?), and an evidence model (what data provide evidence of those cognitive constructs?). When taken as a whole, student, task, and evidence models can help researchers construct principled arguments for interpreting and acting on measures of cognitive constructs [41]. The documentation produced during the ECD process renders design decisions in reusable design artifacts. In assessment, observation of task performance can take many forms: paper-and-pencil tests and scoring of artifacts [11, 22]; strategies, tactics, and moves in gameplay [34, 58]; student moves in simulation-based and tutored learning environments [17]; and environments that make students' thinking visible [38]. Importantly, evidence is obtained not only from passively observing but also by deliberately putting students in situations, challenges, or tasks that will elicit the needed evidence. Our experience, as well as others' [40], is that a rigorous and principled approach to assessment design not only yields valid assessments of knowledge and skills but also produces "templates" for developing new sets of assessments that can be used to measure the same knowledge or skills and perhaps even in other settings. In this way, the varieties of delivery forms described previously can be generated from reusable templates.

In standard assessment design, evidence models are built manually. With EMD/LA methods, we can build evidence models computationally, at least in part, from log data using discovery-driven EDM/LA techniques for detection of activity flow that characterize learners and their knowledge. Log files can then be tagged with semantically meaningful labels prior to statistical analyses to build evidence models of successful strategies for solving the problem [55].

Gobert et al. [17] used a two-stage analytics approach involving mining of log files from a science inquiry OELE, where clips of the log files (chunks of log entries) were first tagged using a text replay tool. Their detectors for the automated coding of log files, subsequent analysis of tagged log files, and other performance information helped to assess learners' knowledge states with a view to providing targeted scaffolding. Kerr and Chung [34] used ECD to inform design of an OELE for K-8 math, and subsequently used the design choices to inform cluster analyses of log files to understand strategies of learners. In other work, a blended approach using ECD to develop a game-based assessment mechanic was used that starts empirically from what the players do in a well-crafted game and detects patterns that may indicate implicit understanding of salient phenomena [54]. These patterns or strategies are then detected in student-generated game data using data-mining techniques to demonstrate students' implicit understanding of fundamental physics. Other more recent work [23] has used blended approaches to analyze student programming processes in the open-source Blockly Games puzzles (http://blocklygames.sri.com/).

The goal of this effort is to share hypothesis-driven analytics approaches using ECD that can support data-driven ones in the context of measuring student understanding in BBPEs using EDM/LA. To our knowledge, such a blended approach to analyzing student actions has not been applied to interpreting programming processes and student learning in the commonly used BBPEs in K-12 CS classrooms. As such, it aims to share findings and learning that may be helpful in informing this nascent field and guiding future efforts.

3 EXPLORATORY WORK: ANALYSES OF DATA FROM THE "FAIRY ASSESSMENT" ALICE TASK

In the early phases of our work, we focus on datasets with a bounded scope to gain traction in our analyses. Because data captured from more open-ended activities contain patterns of learning that are more diffuse and more difficult to detect [5], our work begins with data from directed programming tasks with defined end-states (rather than completely open-ended ones). In this way, we can look for detectable patterns, including student progress toward goals. In future work, we will move to progressively more open-ended tasks. This work is part of a broader effort to understand and measure learner behavior in BBPEs guided by the research question: *What configurations and patterns of behavior in data logs from block-based programming environments provide evidence of learners' use of CT concepts and practices?* This research is an important early step toward a long-term goal to develop a framework for designing better summative assessments of CT learning in programming as well as measurements in real time that support formative feedback and other scaffolds to improve learning in BBPEs.

As part of this initial exploratory work, we analyzed a dataset from a performance assessment task, built in Alice, that was used in prior research by Denner and Werner [10] to measure students' understanding of both Alice programming and more general CT concepts. Werner et al. [63] developed and administered the Fairy Assessment (FA) task (so called because it involved programming fairy characters on an Alice stage) to test what students learned after they had completed approximately 40 hours of game programming exercises in Alice. In their study, 118 females and 202 males aged 10 to 14 years (primarily white: 46%, or Latino/a: 37%) completed the FA that consisted of three independent but consecutive tasks to be completed in 30 minutes. The FA presented students with a starter Alice world and code that needed to be modified to accomplish the tasks. Task 1 was to make a female fairy turn to face another fairy while he is walking into a forest, turning for the same amount of time as the male fairy talks. Task 2 involved debugging an existing program for an event handler to properly resize one of the characters when the up arrow key is pressed. Task 3 was to program the male fairy to fly toward the female fairy when the user clicks the mouse on him, starting from the code as present after Tasks 1 and 2. The FA tested students' ability to comprehend code, modify code according to new requirements, find and fix bugs, and use specific Alice commands and methods.

The dataset for our initial exploratory work consisted of (1) each student's final program at the end of the three tasks, (2) a score for each task in the FA, and (3) the corresponding Alice log data from their work during the 30-minute FA. The final programs had been scored manually using a grading rubric for components of algorithmic thinking and abstraction. Human scorers assigned scores to partial and complete solutions for each of the three tasks, and the overall score was the sum of the three scores. The low-level log data captured very granular details of user action in the Alice BBPE. Werner et al. [64] provide additional details of the dataset. While each task within the FA was independent, students completed them consecutively, so we could not separate work on different tasks in the log data. Our analysis used data from the 229 students for whom we had access to both log data and manually graded final programs. Working with the FA data, we first applied ECD to "reverse engineer" [29], or analyze, each task in the FA, in order to map, post

facto, the specific CT focal concepts and skills that the tasks were measuring, and how evidence of those might look in log files. We also performed sequence analysis on the dataset and correlational analyses between manual scores and frequencies of different action types. This allowed us to gain an understanding of students' programming process using data-driven analytics.

3.1 Sequence Mining and Results

In order to extract student activity sequences from the FA log data, we did a significant amount of preprocessing to remove events that reflected system actions as opposed to student-initiated actions. All the student actions remaining in the logs reflected running, inserting, editing, or deleting code (adding or deleting Alice objects was not logged) and could be categorized as one of the following types:

- *Play*: Students press the Play button in the Alice interface to observe their program's behavior.
- *MultiPlay*: Students press the Play button multiple times, without changing the program in between.
- *PropertyChange*: Students change the parameter(s) of a method(s).
- *InsertBehavior*: Students insert a mapping between an event trigger and the behavior that will be triggered.
- *InsertCallToUserResponse*: Students insert a call to a method based on user inputs or responses.
- *InsertResponse*: Students insert a block in their program (e.g., inserting a *DoTogether* block or a *Move* block).
- *InsertChild*: Students create variables or insert or change a parameter for a block in their program; this is also logged by default when they create new behaviors with default parameters.

On average, each student performed 27 such actions during the course of the FA. We used the TraMineR package for R (designed for mining, describing, and visualizing sequences of states or events and, more generally, discrete sequence data [16]) to analyze sequences of logged student actions. Sequence mining yielded a number of sequences of variable length. Using a support metric, where the support for a particular sequence is defined as the fraction of sequences that contain the particular sequence, we limited our analyses to action sequences with support greater than 0.2 to highlight the most frequent sequences. We discovered 1,871 such sequences, each comprising between one and six actions (we use "→" to show sequences of actions next). The more frequent sequences of student actions with support values in the range of 0.6 to 0.95 were:

- Play;
- MultiPlay;
- Play→PropertyChange(TriggerResponse)→Play;
- InsertBehavior(MouseButtonClickBehavior)→PropertyChange(TriggerResponse).

Some of the less frequent sequences (support values between 0.2 and 0.35) were:

- PropertyChange(Duration)→Play→PropertyChange(OnWhat)→Play→Play;
- PropertyChange(Amount)→MultiPlay→PropertyChange(TriggerResponse)→Play;
- Play→InsertResponse(DoTogether)→Play.

We also compared action sequences between groups of students who received an overall high (above the median) and low (below the median) score to determine whether certain sequences were more or less common in each group. Support metrics were calculated separately for each group and

Table 1. Correlation Matrix for Number of Student Actions and FA Score

	Number of Plays	Number of MultiPlays	Number of Nonplay Actions	Proportion of Nonplay to Play Actions	FA Score
Number of Plays	r = 1				
Number of MultiPlays	r = 0.212*	r = 1			
Number of Nonplay Actions	r = 0.712**	r = 0.35**	r = 1		
Nonplay to Play Actions	r = -0.087	r = 0.297**	r = 0.436**	r = 1	
FA Score	r = 0.276**	r = 0.099	r = 0.227**	r = 0.07	r = 1

*$p < 0.005$, **$p < 0.0001$.

for each sequence. We performed a chi-square statistical significance test for each sequence (with the null hypothesis being that there is no difference between the groups) in terms of observing the sequence in question. The values were Bonferroni adjusted for the number of tests. P-values less than 0.05 indicate that the sequence is distinguishing, and a student with that sequence of actions is more likely to fall into one group compared to the other. Some examples of sequences that were significantly more common among students with high grades compared to students with low grades were:

InsertResponse(Turn)→Play→PropertyChange(Amount)→ Play→Play;
Play→Play→Play→Play→Play→MultiPlay; and
Play→PropertyChange(TriggerResponse)→Play→PropertyChange
(TriggerResponse)→MultiPlay.

Thus, we find that the ***Play*** action (testing the program) in conjunction with others in the sequence has significance for students who perform better. Most ***Play*** actions here book-end changes to code. This resonates with recent work [48] where Test actions that followed or were preceded by Pause and Construct actions were associated with better learning.

On the other hand, there was only one sequence that occurred significantly more frequently for students with low grades compared to students with high grades:

MultiPlay→Play→Play→PropertyChange(TriggerResponse)→PropertyChange
(TriggerResponse)→PropertyChange(TriggerResponse).

This more frequent occurrence could be because sequences that were more common among students with low grades had a low overall support value and were excluded from the analysis. It is reasonable to expect that we could assign a more abstract label to the common or uncommon sequences, such as "trial and error." However, as we will describe in Section 3.3, interpreting these sequences is nontrivial, especially in the absence of any means of ground-truthing.

3.2 Correlational Analyses

We reasoned that some overall measure of student actions might relate to their FA scores. To test this, we calculated the following four measures for each student and then computed the correlation with students' scores for the whole FA (Table 1).

- Number of *Play* actions: Students tested their programs by pressing the Play button an average of 5.9 times (SD = 3.9) during the FA.
- Number of *MultiPlay* actions: Students tested their programs using *MultiPlay* an average of 2.1 times (SD = 1.6) during the FA.
- Number of nonplay actions: Students performed an average of 19.3 actions (SD = 15.4) during the FA that were neither *Play* nor *MultiPlay* actions.
- Proportion of nonplay to *Play* actions: On average, students performed 3.4 nonplay actions (SD = 2.1) for every *Play* action they performed.
- Students with a higher number of nonplay actions and *Play* actions have higher final scores.
- Students who perform more nonplay actions tend to test their program more frequently (i.e., a higher number of *Play* and *MultiPlay* actions).
- Students with a higher number of *Play* actions also perform a higher number of *MultiPlay* actions.
- Students with a higher number of nonplay actions per *Play* action (it appears that they plan their FA tasks and test their programs in chunks as opposed to testing every new block of code) tend to perform a higher number of *MultiPlay* actions. This suggests that they demonstrate the important skill of testing a program multiple times, perhaps to establish generalizability of their solution. However, we cannot currently discern from the logged data whether students vary parameters or event triggers while testing the program multiple times.
- The proportion of nonplay actions to *Play* actions does not seem to have a bearing on their final scores.

3.3 Lessons from Exploratory Work

The exploratory analyses with the FA dataset provided many valuable insights into how we can interpret student actions from log data, but fell short in judging several aspects of CT in the process of programming. The nature of the FA gave rise to a few challenges. The short duration (30 minutes) was insufficient time for students to solve a complex task that would yield rich process data logs as students applied more strategic CT skills. Also, prior literature in both constrained Intelligent Tutoring Systems and OELEs underscores the need to see evidence of appropriate as well as repeated use of constructs—either demonstrating growth over time or mastery—to warrant a claim of evidence of skill [14, 35, 67]. Our reverse engineering showed that the FA does not require that students apply many of the CT skills that we would like to look for and assess in introductory programming—the appropriate usage of nested constructs for iteration and/or conditionals; the creation of new blocks of code with iteration and conditional constructs; the creation of methods to isolate and abstract functionality in the process of problem decomposition, the testing and debugging of components of the solution; the creation of generalizable solutions that work for a range of inputs as opposed to a hard-coded solution that works for one case; the need for a solution that addresses boundary conditions and appropriate termination conditions; and iteratively growing and refining a solution, among others, as described in the CT literature. Further, log data with just seven main action types ultimately resulted in action sequences that could not be interpreted for information on programming processes beyond data-driven visualizations of pathways from beginning to end based on the final FA score and other purely data-driven analyses (described in [12]).

Lastly, because we only had the log files, students' final program, and their manually assessed score, we were unable to verify our interpretations of sequences of actions with any other data source. We had difficulty interpreting some of the action sequences that distinguished students with high and low overall grades. For example, the sequence *Play*→*Play*→*Play*→*Play*→*Play*→*MultiPlay* is more common in high-scoring students and could be interpreted as making changes

to the program, testing each change (*Play*), followed by a final testing for generalizability of the solution by testing multiple times without changing the program (*MultiPlay*). However, without additional measures for triangulation or mapping the sequences back to specific instances in students' programming progressions that we have evidence for, we could not validly interpret such sequences.

Building on these findings of the strengths and limitations of the FA for revealing the programming process, we decided to apply ECD for future work in its typical forward-design application, beginning from important focal concepts and practices, and proceeding to task implementation. This approach yields an overall methodology for combining top-down ECD-like approaches to assessment development and delivery with bottom-up, data-driven approaches. We present our emerging framework next, before describing how we applied it in our next phase of research.

4 A FRAMEWORK FOR USING HYPOTHESIS-DRIVEN APPROACHES TO COMPLEMENT DATA-DRIVEN LEARNING ANALYTICS IN BBPES

Informed by our experiences in the exploratory phase with the FA dataset and by past blended LA efforts that use ECD (albeit in nonprogramming OELE contexts), we apply deliberate hypothesis-driven analyses to complement data-driven findings of student programming actions from Alice log data. We propose a framework for effectively using such a hybrid approach to interpret student actions or action sequences in log data from BBPEs in general.

The framework, depicted in Figure 1 describes a stepwise, iterative process. It begins with **domain modeling** (Row 1), a first step that ensures that we begin the process by articulating the focal CT concepts and practices for which we wish to gather evidence in students' programming. Next is **task modeling** (Row 2), a step that involves designing (or choosing from predefined) tasks and rubrics that provide evidence of those focal concepts and practices that helps us place students in task situations that will elicit those concept and practices, and to plan for potential observables. In the **Programming Task Piloting** (Row 3), or discovery phase, we capture external observations of students' behaviors—perhaps with the use of screen-capture software—in addition to capturing data logs and final programs, in order to interpret and make sense of log data sequences. Detailed code analysis of students' varied solutions (to the same task) helps us get a sense for use of constructs (correct or otherwise) and approaches to solutions (Row 4). Piloting the tasks also helps us judge the demands of the task with different students and yields multiple datasets. Often these pilots will drive changes to the task to improve validity so that there is better alignment between the concepts and practices and the observables from the task delivery, which leads to the design of a refined final task that will be used for larger-scale data capture.

We then develop rubrics (Row 5) to drive our analysis of program code and qualitative data. Analyzing data from observations captured manually or via video or screen recordings reveals students' actions that are unseen in the solution programs—the number of attempts before the student got the correct construct in place; the ways in which the student iteratively built and tested the code; or the number of hit-and-miss attempts that show that a student does not understand a concept even though his or her final solution is correct. Qualitative observations help us derive and discover patterns of programming practices and processes (Row 6). Taken together, qualitative analyses of the solutions and process data from a well-designed task help us understand behaviors in a way that is often difficult solely from computational analyses and visualizations of log data. They help us derive anticipated code blocks and code sequences pertaining to focal concepts and practices that could then be detected in data logs of the larger-scale data capture from the task, as well as antipatterns that denote less effective strategies.

This through-line from the ECD domain modeling stage to the patterns derived (as demonstrated in Section 5.3) helps us interpret student actions in log data in terms of the CT skills and practices

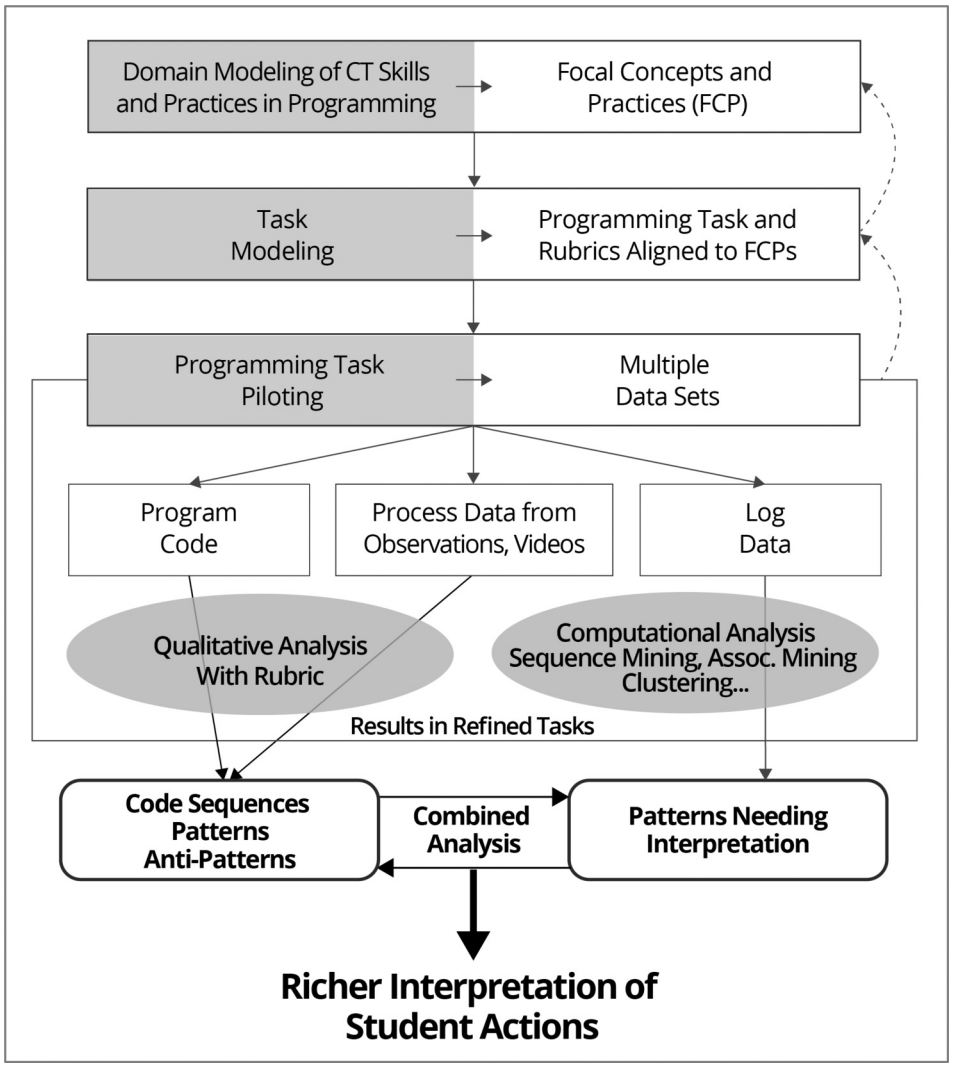

Fig. 1. Blended learning analytics framework describing hypothesis-driven analyses to support data-driven learning analytics in BBPEs.

that we aim to measure (both formatively and summatively). In addition to postulating detectors for patterns and antipatterns, blended approaches involve interpreting patterns revealed bottom-up through data-driven analytics. Some examples of data-driven approaches used in this work are described in Eagle et al. [12]. This combined top-down and bottom-up approach is aimed at providing richer interpretations of student process through log data from BBPEs. Section 5 further explicates this stepwise process and framework "in action."

5 APPLYING THE BLENDED LEARNING ANALYTICS FRAMEWORK: DESIGNING AND PILOTING NEW ASSESSMENT TASKS

This section describes our experience in applying the framework described previously in the context of introductory CS in high school. We used the hypothesis-driven approaches suggested by

the framework to derive a priori patterns (and antipatterns) to be combined with subsequent data-driven analytics for meaningful interpretation of data logs from introductory programming activity in Alice. Guided by the framework, we used ECD for domain modeling to identify CT focal concepts and practices relevant in an introductory programming context in high school, followed by a systematic approach to designing two programming tasks and related rubrics to elicit evidence of the focal CT concepts and practices that had been identified. These were then piloted in two high school classrooms in Northern California as part of ECS course Unit 4 (Programming) activities to capture student programs and process data from observations as well as data logs (logged to CMU's DataShop). We describe our comprehensive qualitative analyses conducted to derive patterns of blocks and sequences of code in the programs as well as discover sequences of actions taken during programming that were not evident from log files but were evident from the screen recording. These contributed to the patterns and antipatterns that we will use to support the data-driven analytics (described in Section 7).

5.1 Domain Modeling of CT Concepts and Practices Using ECD

ECD helps with the design of explicit learning outcomes and measures as part of the assessment design process. It begins with a domain modeling that entails identifying target outcomes and articulating focal knowledge, skills, and other attributes (FKSAs)[1] related to CT in programming. We built on our prior work on domain modeling of CT practices for high school CS [6] to identify which FKSAs were relevant to CT in the context of introductory programming in ECS. The domain-modeling step led to the identification of the following skills and abilities (in no particular order) as key to designing tasks aimed at eliciting evidence of CT in the programming process:

- Use conditionals to incorporate different pathways based on selection criteria
- Design an abstraction to represent a problem or solution
- Create algorithms that capture features, aspects, characteristics, relationships, and/or structure of problems that meet specified purposes
- Create solutions to problems at multiple levels of detail, including describing what components are at each level and relationships among components and between levels
- Incorporate repetition of blocks of code using looping constructs
- Use sequence or parallel execution in algorithmic instructions
- Identify errors in computational work (such as a program, website, or problem solution)
- Create event handlers to deal with event triggers such as keyboard inputs or mouse clicks
- Use Boolean logic expression to incorporate selection and looping criteria
- Use predefined functions or methods in new situations (or to achieve a new goal)
- Create a generalized solution to a problem (as opposed to hard-coding to meet a very specific case)
- Iteratively refine the design of the computational solution (based on results from implementation)
- Implement testing and debugging methods to test and fix a computational solution

5.2 Task Modeling: Designing Programming Tasks Aligned to Focal Concepts and Practices

The FKSAs listed previously then guided the design of programming tasks that would elicit evidence of specific CT skills. Based on the classrooms that we were working in and ECS students'

[1]FKSAs are ECD terminology and are synonymous with FCPs in the framework.

Table 2. Criteria for Programming Task Design

Task Design Constraints	Features Satisfying Task Constraints
Elicit evidence of FKSAs listed in Section 5.1	Each task requires students to complete at least two separate portions to elicit evidence of problem decomposition. One portion requires modifying code, while the other involves adding new code
Draw on concepts that students know about from their Alice curriculum	Task requires use of CT concepts like methods, parameters, event handling, sequencing (do together, do in order), functions, if/else, loop, while, and random numbers (but not lists, recursion, or variables)
Task should be doable in one class period (45 minutes)	Scaffold students' work where necessary to lower the task demands, e.g., by providing specifics in the student instructions for the task
Task should be challenging enough so that not all students will complete it, but all will be able to attempt it and accomplish some aspect(s) of it	Programming the task involves use of various functions and methods, but task instructions will include multiple specifications about desired program behavior
Program output should not be obvious so that the program requires testing under different circumstances	Provide students with different initial scenes while testing their programs; require use of random numbers for the task
Minimize mathematics knowledge and Alice-specific knowledge required that is irrelevant to the target CT FKSAs	Provide students with the background scene, characters, and Alice-specific knowledge

presumed abilities (provided to us by our classroom teacher partners), we also sought to satisfy a list of task design constraints described in Table 2.

We adapted two tasks from the Alice textbook, which provides a comprehensive introductory Alice curriculum [9]. The first task, "Midas Touch" (Figure 2), required students to create a program where a girl's touch turns a candy cane to gold. Students were asked to program arrow-key events such that the girl moved forward until she was a particular distance from the candy cane. Students received an initial Alice (version 2.4) world and a rubric (associated with points toward a final grade) that highlighted important aspects of the expected program behavior. The rubric reflected potential observables based on the FKSAs we wanted to assess, as follows:

(MT 1) The girl moves forward when she is too far from the candy cane
(MT 2) The girl turns left and right using the appropriate arrow keys
(MT 3) Candy cane is touched only when the girl is close enough
(MT 4) The girl bends to touch candy cane
(MT 5) Candy cane changes color

In the second task, "Horse Race" (Figure 3), students were required to program a horse race. They were provided an initial scene with three horses, which would have to race toward the finish line until one horse won. Students were given two prebuilt functions: (1) *isGameOver* to test if the race

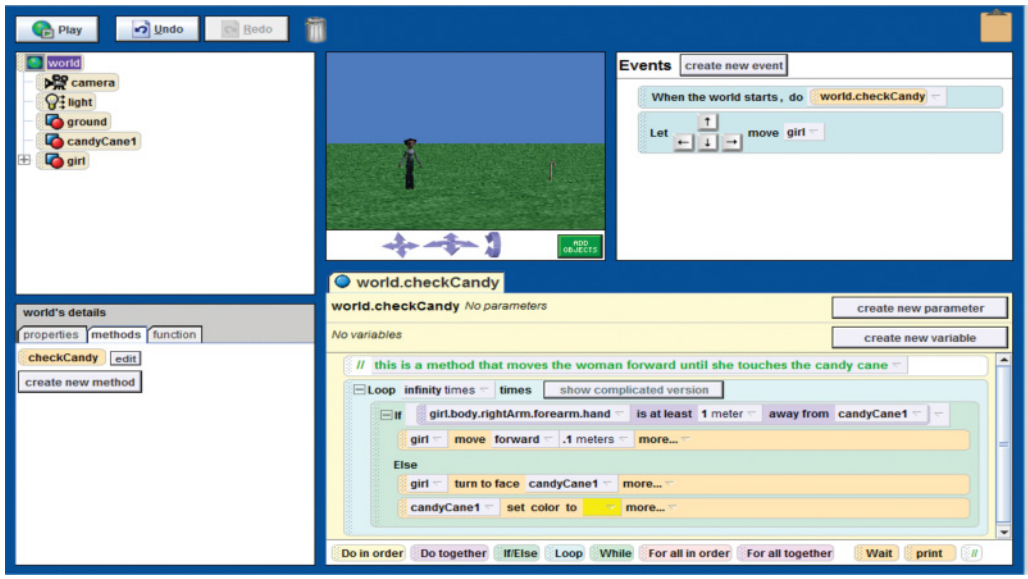

Fig. 2. Sample code and the Alice world for the "Midas Touch" task.

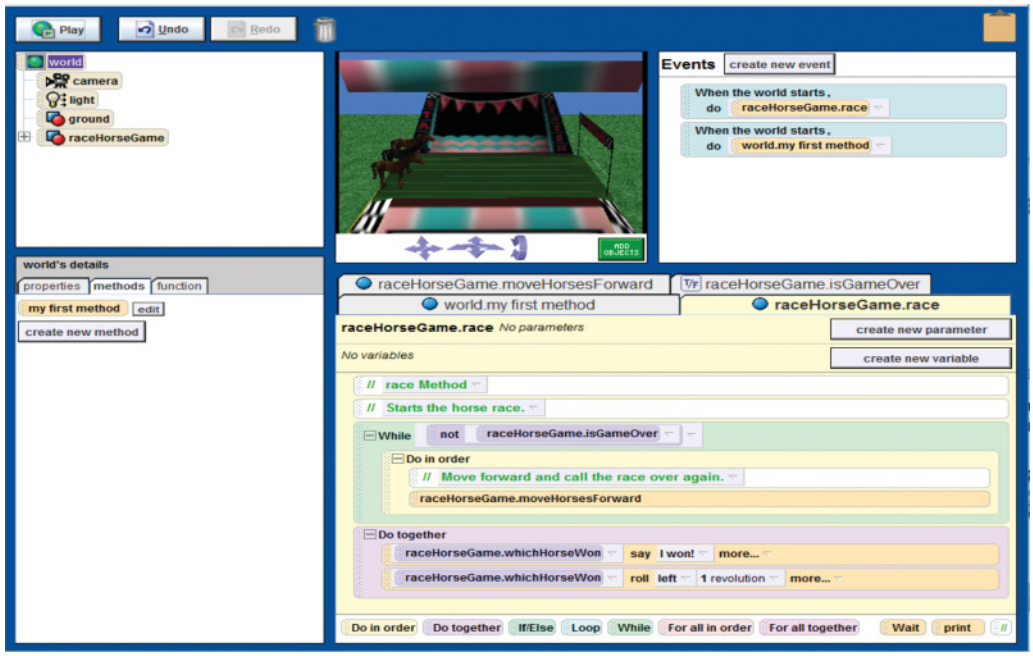

Fig. 3. Sample code for the "Horse Race" task, along with the resultant Alice world.

was over, and (2) w*hichHorseWon* to determine which horse won. Similar to Midas Touch, students received a rubric for the Horse Race task highlighting the following observables:

(HR 1) Each horse moves forward a random amount each time
(HR 2) When a horse wins, the winning horse says "I won!" and does a roll

(HR 3) The simulation stops when a horse wins
(HR 4) Functions provided (isGameOver, whichHorseWon) are used

5.3 Task Piloting: Analyzing Program Files and Screen Recordings

We piloted the tasks in two high school classrooms to gather student programs, as well as process observation data via screen- and video-capture software (Camtasia) and data log capture for a few students. Both the assessment tasks were piloted in a Northern California high school (Asian: 68%, Hispanic: 20%, Caucasian/White: 7%, African American/Black: 2%, Hawaiian Native/Pacific Islander: 0.5%, eligible for free or reduced lunch fees: 16%) during an ECS class in two different sections with 27 and 28 students. Roughly 70% were ninth-grade students, while the rest were in the 11th or 12th grades. Student and parent consent were obtained prior to students attempting the tasks. Data measures included (1) paper-based survey for students to self-report start time, finish time, prior programming experience, and perceived task difficulty; (2) final Alice (.a2w) files; (3) Alice log data; and (4) a screen recording for three students in each section.

In-depth qualitative analyses of student programs and screen recordings of students' actions helped us refine the tasks, and also arrive at observable actions and patterns that we believe will support the hybrid hypothesis-driven/data-driven LA approach. We analyzed students' final programs (Alice files) using a detailed scoring rubric that we developed for this task. While the final programs lacked information about students' processes and their testing and debugging skills, they provided valuable information about students' uses of various programming constructs, predefined functions and methods, and abilities for creation of working and generalizable solutions that satisfy given requirements. Two researchers independently scored the first five student files for each task. Then, they met to discuss and resolve differences in their scoring. Once they established greater than 95% agreement, implying a "very good" interrater reliability rating [28], the researchers split the scoring task for the remaining student files.

The scoring rubric was based on the CT FKSAs that we intended to measure. For both the Midas Touch and Horse Race tasks, we identified eight FKSAs each that we wanted evidence for (see Column 1 of Tables 3 and 4). FKSAs #7 and #8 for both tasks require evidence from students' programming process data that cannot be observed in students' final programs. While mapping the remaining six FKSAs to program behavior observables described in Section 5.2 (see Column 2 of Tables 3 and 4), we realized that the observables in the high-level rubrics were incomplete and at a larger grain size than that required to make conclusive claims about the presence or absence of the target FKSAs. Hence, we developed a detailed rubric (see Column 3 of Tables 3 and 4) containing a larger number of observables that were primarily at the construct level. For example, in the Midas Touch task, the high-level rubric consisting of five observables is broken down into a 17-point detailed rubric. Similarly, in Horse Race, the rubric comprising four observables maps onto a more fine-grained 16-point rubric.

Our analysis of students' Alice files using these detailed rubrics revealed similar difficulties in both tasks, including:

- More than half the students (59%) created solutions that worked correctly for a specific scenario, but most failed to create generalizable solutions.
- Students (72%) had trouble using the "while" construct for repeated checking of a Boolean condition.
- Students (40%) faced difficulties with determining the terminating Boolean logic for loops and constructs like "while."

Table 3. Detailed Scoring Rubrics for the Midas Touch Task Based on Target FKSAs

FKSA	High-Level Rubrics (Program Behavior Observables)	Detailed Rubrics (Construct-Level Observables)
1. Ability to use sequential or parallel execution in algorithmic instructions	MT 4: The girl bends to touch candy cane. MT 5: Candy cane changes color. Parallel execution: Bending involves having the body move forward and touch the candy cane at the same time. Sequential execution: The candy cane changes color after being touched.	i. The student modeled: Turn to face candy cane. ii. The student modeled: Woman bends to touch candy cane, including revolves body, moves body forward, and turns right arm. iii. The student modeled "woman bends to touch candy cane" in part ii, and the bending actions are executed in parallel using a "Do Together" (Parallel execution). iv. The student modeled: Candy cane changes color. v. The student modeled parts i, ii, iv or parts thereof in sequence: Turn to face first, then bend to touch cane and then changes the candy cane's color (Sequential execution).
2. Ability to use conditionals to incorporate different pathways based on selection criteria	MT 3: Candy cane is touched only when the girl is close enough.	vi. The student uses a conditional to indicate that the candy cane is not touched till the girl is close enough vii. The student uses a conditional to indicate that the girl should move forward when she is far from the candy cane. viii. If the student uses a conditional, which construct does he or she use (If/While)?
3. Ability to incorporate repeating actions using looping constructs	MT 1: The girl moves forward when she is too far from the candy cane.	ix. Repeated checking of condition: The girl keeps moving forward till close enough to the candy cane. x. If the student repeatedly checks the condition, what construct is used (While/Repeat)?
4. Ability to use Boolean logic expression to incorporate selection and looping criteria	MT 1: The girl moves forward when she is too far from the candy cane.	xi. Correct stopping Boolean condition for the girl: the student has not hard-coded the distance traveled by the girl or the number of times the loop should run.
5. Ability to create event handlers to deal with event triggers such as keyboard inputs or mouse clicks	MT 2: The girl turns left and right when the appropriate arrow keys are used.	xii. The student modeled an event handler for the left arrow key. xiii. The left arrow key makes the girl turn left. xiv. The student modeled an event handler for the right arrow key. xv. The right arrow key make the girl turn right.
6. Ability to create a generalized solution (as opposed to hard-coding to meet a very specific case)	No observable mentioned in high-level rubric	xvi. The student achieves a working solution. Requirements: The girl moves forward till 1 meter from or close enough to the candy cane, then bends to touch the cane and changes its color. xvii. The program is generalizable. It works correctly even if the girl is moved off the default path using arrow keys.
7. Ability to iteratively refine the computational solution	Cannot be observed from the final program file	*Cannot be observed from the final program file*
8. Ability to test and debug a computational solution	Cannot be observed from the final program file	*Cannot be observed from the final program file*

Table 4. Detailed Scoring Rubrics for Horse Race Task Based on Target FKSAs

FKSA	High-Level Rubrics (Program Behavior Observables)	Detailed Rubrics (Construct-Level Observables)
1. Ability to use sequential or parallel execution in algorithmic instructions	HR 1: Each horse moves forward a random amount each time (Parallel execution).	i. All three horses move forward when the program is run. ii. The horses move forward by random distances. iii. Distance traveled by horses is between 0 and 0.5 meters at each step. iv. All three horses move together each time (Parallel execution). v. The winning horse says "I won." vi. The winning horse rolls after saying "I won" (Sequential execution).
2. Ability to use predefined functions or methods in new situations (or to achieve a new goal)	HR 4: The existing functions that are provided (whichHorseWon and isGameOver) are used.	vii. When the world is started, the race method is executed. viii. The given "IsGameOver" function is used in the race method. ix. The given "whichHorseWon" function is used in the race method. x. The "moveHorsesForward" method is used in the race method.
3. Ability to use conditionals	HR 2: When a horse wins, the winning horse says "I won!" and then does a roll.	xi. The horses move forward only when the game is not over.
4. Ability to use Boolean logic expression to incorporate selection and looping criteria	HR 4: The existing functions that are provided are used.	xii. The "isGameOver" function is used to check till when the horses should keep moving forward.
5. Ability to incorporate repetition of blocks of code using looping constructs	HR 1: Each horse moves forward a random amount each time.	xiii. The horses keep moving forward till the game is over. xiv. When one horse reaches the finish line, the other horses stop.
6. Ability to create a generalized solution (as opposed to hard-coding to meet a very specific case)	No observable mentioned in high-level rubric	xv. The student achieves a working solution. Requirements: One horse reaches the finish line and the other horses stop where they are; the winning horse says "I won!" and rolls. xvi. The program is generalizable. The distances traveled by the horses are random and not hard-coded.
7. Ability to iteratively refine the solution	Cannot be observed from the final program file	*Cannot be observed from the final program file*
8. Ability to test and debug a computational solution	Cannot be observed from the final program file	*Cannot be observed from the final program file*

- Students (34%) found it challenging to distinguish which actions to execute in parallel and which to execute sequentially.
- Students (50%) faced challenges with using predefined functions and decomposing their program into functions and methods.
- In the Horse Race task, students (20%) faced difficulties with programming the horses to move by random amounts as opposed to fixed amounts.

Based on the survey responses, 62% of students thought the tasks were easy or of medium difficulty, while 38% found the tasks to be hard. Students who found the tasks to be easy or of medium difficulty mostly had a working solution, but the solutions were often not generalizable. However, several of the students who said they found the task "hard" did not even get to a working solution, let alone a generalizable one. Based on our code analysis, we have refined the tasks to make the instructions more explicit and also to make the tasks longer (over two to three class periods) and more complex (by removing some of the preprogrammed methods) to elicit richer process data in the next phase (Section 7).

We also analyzed the screen recordings from three students in each section. Using a "process over product" lens, we sought potential observables of CT practices related to abstraction, testing, and debugging, and modularization of program behavior into separate methods. The recordings revealed that some students tested individual methods in isolation instead of only testing the main method that contained references to other methods. Although students were instructed to create and use some of the methods, some students also created their own methods, tested the methods in isolation, and then used the methods in the main method. These practices demonstrate the important CT skills of abstraction, modularization, and testing in parts. They could serve as useful patterns to search for in students' log data, which could provide evidence for the presence of CT practices. In addition, we notice certain phases during students' programming process when a student appeared to be flailing and unable to progress. The student added, deleted, or reorganized existing actions and repeatedly tested the program after each small edit. While repeatedly testing the program after every edit is not a practice we necessarily need to discourage, it does become problematic when it is not purposeful and does not result in progress toward the task goal even after repeated editing and testing. Without any intervention, such circumstances can easily lead to frustration and loss of engagement. Therefore, such situations can be good candidates for potential antipatterns to be detected in the programming process that would warrant additional scaffolding.

5.4 Sample *a Priori* Hypotheses-Driven Indicators of Process: Patterns and Antipatterns

We have discussed in Section 4 that log data alone cannot help us draw conclusions about students' programming behavior or the presence or absence of CT practices. When hypothesis-driven indicators of students' programming process obtained from program code, videos, and other student observations are combined with patterns generated from the log data, it leads to richer interpretation of students' actions. On the other hand, when we carefully design assessment tasks to elicit evidence of certain skills and abilities (as with Midas Touch and Horse Race), analysis of students' final solutions and programming process (through video or screen recordings) can reveal patterns that provide evidence for or against students' CT skills. Using these meaningful patterns, we can tag sequences of events or actions from students' log data to specific higher-level skills and abilities or common challenges.

For example, the patterns of behavior observed in the screen recording analyses described in the previous section would point toward log data sequences that correspond to actions depicting CT strategies such as testing in isolation. The event sequence shown in Figure 4 indicates a problem decomposition (or test-in-parts) strategy that we discovered through video analyses of Camtasia screen capture. The user specifies the world.raceHorseGame.moveHorsesForward method as a world-start behavior (1) and then presses the "Play" button (World-start event followed by World-stop) to test the particular method in isolation from other methods (2).

Other similar analyses of data from the Midas Touch and Horse Race tasks have yielded insights into a number of desired CT practices and common student missteps, which we will use to inform our blended learning analytics of data collected in the next phase of the research.

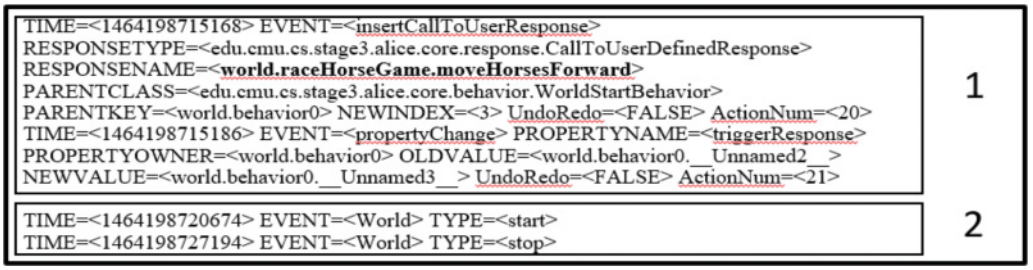

Fig. 4. Event sequence indicating a test-in-parts strategy.

In conclusion, drawing up such hypothesis-driven indicators of students' programming process can help guide mining log data for high-level skills and abilities, instead of low-level actions that are usually uninterpretable. This new approach to blending data- and hypothesis-driven analysis methods in analyzing processes in BBPEs follows three main phases: (1) task modeling using ECD to elicit evidence for specific concepts and abilities, (2) a qualitative analysis of student programs and processes leading to a hypothesis of anticipated student approaches and missteps, and (3) informing data-driven analysis using the identified hypothesis-driven indicators of process.

6 DISCUSSION

EDM/LA techniques are showing promise for helping researchers understand the moment-by-moment actions that students take in terms that are meaningful for personalizing learning. In our work to understand process over product in BBPEs, we have developed a preliminary framework that helps researchers combine hypothesis-driven, knowledge-engineering approaches with data-driven, data-mining approaches. We have shown that this framework is especially helpful in building ways to make claims about student understanding of CT skills in the context of programming. This will help us blend EDM and LA techniques with ECD to create automated ways to better understand student behaviors in programming environments.

Our prior work using ECD shapes the way we think about interpreting learner actions captured through trace data in log files: as seeking evidence that warrants claims about students' learning. ECD's disciplined approach forces us to make explicit the components that go into making claims about learning and to create tasks that prompt students to generate observable evidence of hard-to-assess skills. From the domain modeling and analyses that assessment and domain experts conduct to reify the components of concepts like abstraction, design, and algorithmic thinking, we obtain a better understanding of what the measurable grain size is and what individual knowledge components make up a higher-level concept or skill. Domain analyses also reveal what work products students could produce that use these knowledge and skills and what observations may give us evidence of their presence.

When we create, pilot, and validate a new task using ECD, we are more likely to have a close connection between the task demands and the skills required, resulting in better measurement of students' skills. However, this approach can also be used with predefined tasks. When we encounter a predesigned task, we can apply reverse engineering to model that task in terms of demands on the student. We can unearth assumptions that task designers made and answer questions such as: What background knowledge does this task assume that students have (such as understanding geometric descriptions for movements in a 3D space or knowing Boolean logic)? Does this task properly differentiate students who have mastered concepts versus those who are still developing

them? Do irrelevant factors get in the way of students' ability to demonstrate their knowledge (e.g., forgetting the name of a command or receiving too much scaffolding)?

When we place learners into a situation where they encounter the task and the stakes are appropriately high (as in a designated assessment activity) or the task is sufficiently engaging that they take it seriously, we obtain observables from "looking over the shoulder" of the students using in-person or screen-recording methods. Along with these observations, we get product and process data, including products that are incorrect or incomplete, and processes that are not strategic (e.g., antipatterns). Thus, we end up with a constellation of artifacts and data that we can interpret through the lenses of domain models, task models, and learner models.

In terms of process data, however, we know that there is still a gap in creating an effective blend of hypothesis-driven and data-driven approaches in BBPE contexts. Consider traditional assessment design. Task designers create artificial situations that funnel potential student responses toward a set of acceptable ones. Even when the response is complex, like a short-answer text, it is possible to build models that extract features from samples of human-scored responses and then to auto-score new responses. But such models of the final product do not apply to log or process data used for assessment.

In using log data from a BBPE such as Alice, where the logging was originally created for debugging the software (but incrementally improved during this project), we know that the features present in just one log entry are not sufficiently rich to serve as evidence of what a student knows or can do. We can get a better picture by instead looking at sequences across a time span, such as watching the expansion or contraction of a program. Prior work in analyzing log data from programming environments looked at features such as when and how frequently students compile/play; how many compilations are successful; whether they have entered a "sink state"; whether they tinker or plan; whether they use particular programming constructs; length of time taken to solve a problem; and so on. These data-driven approaches get us so far but fall short in more open-ended programming tasks in yielding accurate assessments of understanding of individual concepts and skill levels. They also do not necessarily give us pointers to places in learning where formative assessment may help. In these cases, a more complex set of behaviors, suggested by our hypothesis-driven analysis of the domain, is more helpful along with in-depth qualitative analyses of student artifacts and processes as described in Section 5.

7 NEXT STEPS AND FUTURE WORK

We are conducting the next phase of our research, applying our proposed framework to derive a priori patterns as described in Section 5. We are currently gathering program files and log data on refined versions of our programming tasks from ~100 students in three high school ECS classrooms in Northern California. Data collection includes classroom observations, screen recordings, and interviews with three students in each classroom. These will be used to validate program snapshots created from log data, in addition to aiding student recall of process during one-on-one interviews with each student. These interviews will also be used to ascertain the nature and timing of help that students may have liked to support their work. This will help us understand the nature of formative feedback and supports that can help scaffold learning for individual students.

We will make a first pass at manually coding the students' activity as observed in the videos, and then use the student interviews to disambiguate any actions that we could not code. As needed, the proposed concepts and practices (from the ECD domain analysis) will be refined to capture the actual concepts and practices for which students produce evidence. Additionally, program and process data will be collected for the two tasks described in Section 5.2 and longitudinally across three additional programming projects that the students will be doing in Alice as part of the ECS Unit 4 curriculum. The patterns and antipatterns derived a priori will be combined with the data-driven

approaches (described in [12] for a comprehensive blended analytics approach to understanding programming processes and assessing students' CT skills). Lastly, we will also be using data from students' performance in the ECS Unit 4 assessment [52] for validation of measurements of student learning based on our approaches, as well as additional information on student learning.

Our future work involves distilling process insights gleaned from closed tasks that can be generalized across programming tasks to aid a better understanding of the development of CT skills in programming and how this process can be scaffolded for the learner. These include common antipatterns used by students in their construction of loops across several projects, patterns of testing and debugging, or (hard-)coding for a specific case rather than a generalizable solution and testing a program with different inputs or starting points (as in the case of Alice worlds). Eventually, we also expect to use our methods to build adaptive scaffolds in the context of BBPEs or to provide formative feedback to the learner ***during the process of programming***. Such scaffolds or feedback will require real-time program and student behavior analysis; our post hoc analyses of log data and blended analytic approaches in this exploratory research will be a step toward understanding the programming process and building those predictive models for learners' CT FCPs.

Thus, we envision this early work and this framework guiding the design of intelligent and adaptive versions of current BBPEs that afford learners a personalized, scaffolded learning experience by leveraging log data of student actions that powers a real-time learning analytics engine. Such intelligent BBPEs, or iBBPEs, will also draw on research on scaffolding [45] to provide learners with appropriate guidance when they are detected to be in need of support on the semantics and strategies of constructing programs. These scaffolds would support formative assessment for learners on their problem-solving strategies at appropriate junctures and in appropriate ways to guide them to more productive design approaches and CT practices. These could take the form of hints when learners appear to be flailing or suggestions to refer to past programs that involve similar mechanics to the current program. Alternatively, learners may ask for suggestions for productive debugging. Such responsive adaptation is enabled by systems that sense, interpret, and act [42]. iBBPEs could also support socio-technical features such as social agents to provide feedback for improved learning [as in 43, 15, 4].

In closing, this research presents an emergent methodology and framework for computational psychometrics involving the use of hypothesis-driven analyses to effectively support data-driven learning analytics in BBPEs. The approaches we have described aim for better and richer interpretation of student actions that can aid judicious assessment of student learning and development of CT skills in open-ended programming environments. The framework is generalizable beyond the Alice programming environment to BBPEs more broadly. As schools and classrooms nationally and internationally take bold steps toward "Computer Science for All" and introduce tools and curricula to train the next generation of citizens to become creative computational thinkers and problem-solvers, our work will guide the field in thinking about ways to better measure student learning of computational thinking in the more authentic, naturalistic settings of BBPEs. This is an important step toward ensuring that all learners, regardless of background and prior technology experiences, are well supported in this quest to help them gain mastery over this crucial new competency.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Jill Denner, ETR Associates, for providing the Fairy Assessment dataset. We are grateful to Profs. Stephen Cooper, Linda Werner, Mehran Sahami, Matthew Berland, Margaret Burnett, and Dr. Margaret Heritage for advising our research. Our thanks also go to high school ECS teachers in Northern California for their participation in this project.

REFERENCES

[1] Saleema Amershi and Cristina Conati. 2009. Combining unsupervised and supervised classification to build user models for exploratory learning environments. *Journal of Educational Data Mining* 1, 1 (2009), 18–71.

[2] Ryan S. J. D. Baker and Paul Salvador Inventado. 2014. Educational data mining and learning analytics. In *Cambridge Handbook of the Learning Sciences*, K. Sawyer (Ed.). Cambridge University Press.

[3] Ryan S. J. D. Baker and Kalina Yacef. 2009. The state of educational data mining in 2009: A review and future visions. *Journal of Educational Data Mining* 1, 1 (2009), 3–17.

[4] Satabdi Basu, Gautam Biswas, and John S. Kinnebrew. 2017. Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. *User Modeling and User-Adapted Interaction* 27, 1 (March 2017), 5–53. DOI: https://doi.org/10.1007/s11257-017-9187-0

[5] Matthew Berland, Taylor Martin, Tom Benton, Carmen Petrick Smith, and Don Davis. 2013. Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences* 22, 4 (September 2013), 564–599. DOI: https://doi.org/10.1080/10508406.2013.836655

[6] Marie Bienkowski, Eric B. Snow, Daisy Wise Rutstein, and Shuchi Grover. 2015. *Assessment Design Patterns for Computational Thinking Practices: A First Look*. Menlo Park, CA: SRI International.

[7] Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper, and Daphne Koller. 2014. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences* 23, 4 (October 2014), 561–599. DOI: https://doi.org/10.1080/10508406.2014.954750

[8] Steve Cooper, Shuchi Grover, Mark Guzdial, and Beth Simon. 2014. A future for computing education research. *Communications of the ACM* 57, 11 (October 2014), 34–36. DOI: https://doi.org/10.1145/2668899

[9] Wanda P. Dann, Stephen Cooper, and Randy Pausch. 2009. *Learning to Program with Alice*. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

[10] Jill Denner and Linda Werner. 2009. The Development of Computational Thinking Among Middle School Students Creating Computer Games. NSF Award DRL-0909733.

[11] Jill Denner, Linda Werner, and Eloy Ortiz. 2012. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education* 58, 1 (January 2012), 240–249. DOI: https://doi.org/10.1016/j.compedu.2011.08.006

[12] Michael Eagle, Nicholas Diana, John C. Stamper, Shuchi Grover, and Marie Bienkowski. (in review). Deriving insight into problem solving behavior in interactive open-ended programming environments from transactional data. *Transactions on Computing Education*.

[13] Deborah A. Fields, Michael Giang, and Yasmin Kafai. 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)*. New York: ACM, 2–11. DOI: https://doi.org/10.1145/2670757.2670768

[14] Deborah A. Fields, Lisa Quirke, Janell Amely, and Jason Maughan. 2016. Combining big data and thick data analyses for understanding youth learning trajectories in a summer coding camp. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE'16)*. New York: ACM, 150–155. DOI: https://doi.org/10.1145/2839509.2844631

[15] Jesse Fox, Sun Joo (Grace) Ahn, Joris H. Janssen, Leo Yeykelis, Kathryn Y. Segovia, and Jeremy N. Bailenson. 2015. Avatars versus agents: A meta-analysis quantifying the effect of agency on social influence. *Human–Computer Interaction* 30, 5 (September 2015), 401–432. DOI: https://doi.org/10.1080/07370024.2014.921494

[16] Alexis Gabadinho, Gilbert Ritschard, Nicholas S. Müller, and Matthias Studer. 2011. Analyzing and visualizing state sequences in r with traminer. *Journal of Statistical Software* 40, 4 (2011), 1–37. DOI: https://doi.org/10.18637/jss.v040.i04

[17] Janice D. Gobert, Michael A. Sao Pedro, Ryan S. J. d. Baker, Ermal Toto, and Orlando Montalvo. 2012. Leveraging educational data mining for real-time performance assessment of scientific inquiry skills within microworlds. *Journal of Educational Data Mining* 4, 1 (October 2012), 153–185.

[18] Joanna Goode. 2008. Increasing diversity in K-12 computer science: Strategies from the field. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'08)*. Portland, OR: ACM, 362–366. DOI: https://doi.org/DOI=http://dx.doi.org/10.1145/1352135.1352259

[19] Joanna Goode and Gail Chapman. 2015. Exploring Computer Science, version 6. (2015). Retrieved from http://www.exploringcs.org/curriculum.

[20] Google Inc. and Gallup Inc. 2016. Diversity Gaps in Computer Science: Exploring the Underrepresentation of Girls, Blacks and Hispanics.

[21] Ralph F. Grove. 1998. Using the personal software process to motivate good programming practices. In *Proceedings of the 6th Annual Conference on the Teaching of Computing and the 3rd Annual Conference on Integrating Technology into Computer Science Education: Changing the Delivery of Computer Science Education (ITiCSE'98)*. New York: ACM, 98–101. DOI: https://doi.org/10.1145/282991.283046

[22] Shuchi Grover. 2017. Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom. In *Emerging Research, Practice, and Policy on Computational Thinking*, Peter J. Rich and Charles B. Hodges (Eds.). Educational Communications and Technology: Issues and Innovations. Berlin: Springer International Publishing, 269–288. DOI: https://doi.org/10.1007/978-3-319-52691-1_17

[23] Shuchi Grover, Marie Bienkowski, John Niekrasz, and Matthias Hauswirth. 2016. Assessing problem-solving process at scale. In *Proceedings of the 3rd ACM Conference on Learning @ Scale (L@S'16)*. New York: ACM, 245–248. DOI: https://doi.org/10.1145/2876034.2893425

[24] Shuchi Grover, Stephen Cooper, and Roy Pea. 2014. Assessing computational learning in K-12. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*. ACM, 57–62. DOI: https://doi.org/10.1145/2591708.2591713

[25] Shuchi Grover and Roy Pea. 2013. Computational thinking in k–12 a review of the state of the field. *Educational Researcher* 42, 1 (January 2013), 38–43. DOI: https://doi.org/10.3102/0013189X12463051

[26] Shuchi Grover, Roy Pea, and Stephen Cooper. 2015. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education* 25, 2 (April 2015), 199–237. DOI: https://doi.org/10.1080/08993408.2015.1033142

[27] Shuchi Grover, Roy Pea, and Stephen Cooper. 2014. Remedying misperceptions of computer science among middle school students. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE'14)*. New York: ACM, 343–348. DOI: https://doi.org/10.1145/2538862.2538934

[28] Kilem L. Gwet. 2014. *Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters*. 4th ed. Advanced Analytics.

[29] Kathleen C. Haynie, Geneva D. Haertel, Andrea A. Lash, Edys S. Quellmalz, and Angela Haydel DeBarger. 2006. *Reverse Engineering the NAEP Floating Pencil Task Using the PADI Design System*. Menlo Park, CA: SRI International.

[30] Will Jernigan, Amber Horvath, Michael Lee, Margaret Burnett, Taylor Cuilty, Sandeep Kuttal, Anicia Peters, Irwin Kwan, Faezeh Bahmani, and Andrew Ko. 2015. A principled evaluation for a principled idea garden. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'15)*. 235–243. DOI: https://doi.org/10.1109/VLHCC.2015.7357222

[31] Saj-Nicole A. Joni and Elliot Soloway. 1986. But my program runs! Discourse rules for novice programmers. *Journal of Educational Computing Research* 2, 1 (February 1986), 95–125. DOI: https://doi.org/10.2190/6E5W-AR7C-NX76-HUT2

[32] K-12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org.

[33] Caitlin Kelleher. 2006. *Motivating Programming: Using Storytelling to Make Computer Programming Attractive to Middle School Girls*. Carnegie Mellon University.

[34] Deidre Kerr and G. Chung. 2012. Identifying key features of student performance in educational video games and simulations through cluster analysis. *Journal of Educational Data Mining* 4, 1 (2012), 144–182.

[35] Kenneth R. Koedinger, Albert T. Corbett, and Charles Perfetti. 2012. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science* 36, 5 (April 2012), 757–798. DOI: https://doi.org/10.1111/j.1551-6709.2012.01245.x

[36] Kyu Han Koh, Ashok Basawapatna, Vicki Bennett, and Alexander Repenning. 2010. Towards the automatic recognition of computational thinking for adaptive visual language learning. In *Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing*. 59–66. DOI: https://doi.org/10.1109/VLHCC.2010.17

[37] D. Midian Kurland and Roy D. Pea. 1985. Children's mental models of recursive logo programs. *Journal of Educational Computing Research* 1, 2 (May 1985), 235–243. DOI: https://doi.org/10.2190/JV9Y-5PD0-MX22-9J4Y

[38] Marcia C. Linn, Douglas Clark, and James D. Slotta. 2003. WISE design for knowledge integration. *Science Education* 87, 4 (July 2003), 517–538. DOI: https://doi.org/10.1002/sce.10086

[39] Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. 2013. Learning computer science concepts with Scratch. *Computer Science Education* 23, 3 (September 2013), 239–264. DOI: https://doi.org/10.1080/08993408.2013.832022

[40] Robert J. Mislevy, John T. Behrens, Kristen E. Dicerbo, and Roy Levy. 2012. Design and discovery in educational assessment: Evidence-centered design, psychometrics, and educational data mining. *JEDM-Journal of Educational Data Mining* 4, 1 (2012), 11–48.

[41] Robert J. Mislevy and Geneva D. Haertel. 2006. Implications of evidence-centered design for educational testing. *Educational Measurement: Issues and Practice* 25, 4 (December 2006), 6–20. DOI: https://doi.org/10.1111/j.1745-3992.2006.00075.x

[42] National Academy of Education (NAE). 2013. *Adaptive Educational Technologies*. Washington, DC: National Academy of Education.

[43] Sandra Y. Okita, Jeremy Bailenson, and Daniel L. Schwartz. 2008. Mere belief in social action improves complex learning. In *Proceedings of the 8th International Conference for the Learning Sciences (ICLS'08)*. Utrecht, The Netherlands: International Society of the Learning Sciences, 132–139.

[44] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.
[45] Roy D. Pea. 2004. The social and technological dimensions of scaffolding and related theoretical concepts for learning, education, and human activity. *Journal of the Learning Sciences* 13, 3 (July 2004), 423–451. DOI: https://doi.org/10.1207/s15327809jls1303_6
[46] Roy D. Pea and D. Midian Kurland. 1984. On the cognitive effects of learning computer programming. *New Ideas in Psychology* 2 (1984), 137–168.
[47] James W. Pellegrino and Margaret L. Hilton. 2013. *Education for Life and Work: Developing Transferable Knowledge and Skills in the 21st Century*. National Academies Press.
[48] Sarah Perez Massey-Allard, J. Butler, D. Ives, J. Bonn, D. Yee, and N. Ido Roll. 2017. Identifying productive inquiry in virtual labs using sequence mining. In *Proceedings of the International Conference on Artificial Intelligence in Education*. Berlin: Springer Verlag.
[49] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. 2015. Learning program embeddings to propagate feedback on student code. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*. Lille, France: JMLR.org, 1093–1102.
[50] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE'12)*. New York: ACM, 153–160. DOI: https://doi.org/10.1145/2157136.2157182
[51] Alexander Repenning. 2000. AgentSheets®: An Interactive Simulation Environment with End-User Programmable Agents. Interaction.
[52] PACT Assessments for Exploring Computer Science. http://pact.sri.com/ecs-assessments.html.
[53] Cristina Romero and Sebastián Ventura. 2010. Educational data mining: A review of the state of the art. *IEEE Transactions on Systems, Man, and Cybernetics* 40, 6 (November 2010), 601–618. DOI: https://doi.org/10.1109/TSMCC.2010.2053532
[54] Elizabeth Rowe, Jodi Asbell-Clarke, and Ryan S. Baker. 2015. Serious games analytics to measure implicit science learning. In *Serious Game Analytics: Methodologies for Performance Measurement, Assessment, and Improvement*, C. S. Loh, Y. Sheng, and D. Ifenthaler (Eds.). Berlin: Springer International Publishing, 343–360.
[55] Andre A. Rupp, Roy Levy, Kristen E. Dicerbo, Shauna J. Sweet, Aaron V. Crawford, Tiago Calico, Martin Benson, Derek Fay, Katie L. Kunze, and John T. Behrens. 2012. Putting ECD into practice: The interplay of theory and data in evidence models within a digital learning environment. *JEDM-Journal of Educational Data Mining* 4, 1 (October 2012), 49–110.
[56] Andre A. Rupp, Rebecca Nugent, and Brian Nelson. 2012. Evidence-centered design for diagnostic assessment within digital learning environments: Integrating modern psychometrics and educational data mining. *JEDM-Journal of Educational Data Mining* 4, 1 (October 2012), 1–10.
[57] Michael A. Sao Pedro, Ryan S. J. d. Baker, Janice D. Gobert, Orlando Montalvo, and Adam Nakama. 2013. Leveraging machine-learned detectors of systematic inquiry behavior to estimate and predict transfer of inquiry skill. *User Modeling and User-Adapted Interaction* 23, 1 (March 2013), 1–39. DOI: https://doi.org/10.1007/s11257-011-9101-0
[58] Valerie Shute and Matthew Ventura. 2013. *Stealth Assessment: Measuring and Supporting Learning in Video Games*. Cambridge, MA: MIT Press.
[59] Elliot Soloway. 1986. Learning to program=learning to construct mechanisms and explanations. *Communications of the ACM* 29, 9 (September 1986), 850–858. DOI: https://doi.org/10.1145/6592.6594
[60] Elliot Soloway and James C. Spohrer. 1989. *Studying the Novice Programmer*. Hillsdale, NJ: L. Erlbaum Associates.
[61] Shauna J. Sweet and Andre A. Rupp. 2012. Using the ECD Framework to support evidentiary reasoning in the context of a simulation study for detecting learner differences in epistemic games. *JEDM-Journal of Educational Data Mining* 4, 1 (October 2012), 183–223.
[62] Ian Utting, Stephen Cooper, Michael Kölling, John Maloney, and Mitchel Resnick. 2010. Alice, greenfoot, and scratch–a discussion. *ACM Transactions on Computing Education* 10, 4 (November 2010), 1–11. DOI: https://doi.org/10.1145/1868358.1868364
[63] Linda Werner, Jill Denner, Shannon Campe, and Damon Chizuru Kawamoto. 2012. The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE'12)*. New York: ACM, 215–220. DOI: https://doi.org/10.1145/2157136.2157200
[64] Linda Werner, Charlie McDowell, and Jill Denner. 2013. A first step in learning analytics: Pre-processing low-level Alice logging data of middle school students. *Journal of Educational Data Mining* 5, 2 (2013), 11–37.
[65] Jeannette M. Wing. 2006. Computational thinking. *Communications of the ACM* 49, 3 (2006), 33–35.
[66] Philip H. Winne and Ryan S. J. d. Baker. 2013. The potentials of educational data mining for researching metacognition, motivation and self-regulated learning. *JEDM-Journal of Educational Data Mining* 5, 1 (2013), 1–8.
[67] Benjamin Xie and Hal Abelson. 2016. Skill progression in MIT app inventor. In *Proceedings of the 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'16)*. 213–217. DOI: https://doi.org/10.1109/VLHCC.2016.7739687

[68] Benjamin Xie, Isra Shabir, and Hal Abelson. 2015. Measuring the usability and capability of app inventor to create mobile applications. In *Proceedings of the 3rd International Workshop on Programming for Mobile and Touch (PROMOTO'15)*. New York: ACM, 1–8. DOI: https://doi.org/10.1145/2824823.2824824

[69] Aman Yadav, David Burkhart, Daniel Moix, Eric Snow, Padmaja Bandaru, and Lissa Clayborn. 2015. Sowing the seeds: A landscape study on assessment in secondary computer science education, comp. *Science Teachers Association*. New York.

[70] Iris Zur-Bargury, Bazil Pârv, and Dvir Lanzberg. 2013. A nationwide exam as a tool for improving a new curriculum. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'13)*. New York: ACM, 267–272. DOI: https://doi.org/10.1145/2462476.2462479

Received October 2016; revised May 2017; accepted June 2017