Learning Analytics for the Assessment of Interaction with App Inventor

Mark Sherman
Department of Computer Science
University of Massachusetts Lowell
Lowell, Massachusetts
msherman@cs.uml.edu

Fred Martin
Department of Computer Science
University of Massachusetts Lowell
Lowell, Massachusetts
fredm@cs.uml.edu

***Abstract*—Position statement on the ongoing development of an instrument to capture snapshots of App Inventor projects as students are developing, allowing examination of the students' processes. Data mining techniques will be used to process the code snapshots, identifying patterns of behavior. Future tools may include real-time assessment, and improved taxonomy of programming process.**

Keywords—programming, learning analytics, assessment, blocks, process-oriented

I. Introduction

Learning analytics offers new insights into the learning processes of students in computing. In our prior work, we developed an assessment rubric that was based upon the completed state of a student's work product [6]. Here, we present a plan for a process-oriented assessment of students work.

II. Research Concept

Add instrumentation to app inventor that will capture the state of the project, including code, the design, and the live-running app interaction, live as the user builds their app. Evaluate this data to discover user interactions with App Inventor that are not captured in wor

III. Research Questions

Can this form of data be used to indicate anything about student learning? How does a block-based environment affect this analysis? How does a live-running environment effect this analysis? Comparison against textual languages is addressed below, in Future Work.

Orthogonal to being a blocks language, App Inventor is also purely event-based. What insights can be found with these data about users interacting with event-based programming? What insights can be found about users interacting live with their app while programming?

Can real-time assessment instruments be developed from this technique? What improvements to the ex-post-facto instrument need to be made to allow for real-time analysis? Can relevant interventions for students be automatically triggered by their work?

IV. Background

The core idea is to apply big data analytics to student work, but not just the final product- instead, collect snapshots of the student program, live, as they work on it. This ability to see the process of programming in situ, not just the final product, was a huge advancement, with literature really picking up with and around the time of Piech 2012 [5].

Another data-mining paper, Berland 2013, focused on tinkering [1]. They used data mining techniques, with human-selected factors and machine-generated code state clustering. They analyzed one problem in a specific-domain educational language in a non-formal summer camp environment. They isolated certain factors, such as number of unique primitives in a program, as important for clustering the program snapshots into related "states," and how students transitioned through states, over time. This method of time-based assessment was a finding. Their analysis leaves open the opportunity for additional analysis, as it only looks at the entire cohort in aggregate, and treats that averaged data as a single, generalized student. This approach did not answer how any specific students moved through their program development.

Piech used unsupervised machine learning to perform a more robust analysis. The test problem was in Java, and in a large, university CS1 class. The problem studied was a variant of Karel the Robot, which provided a problem-constrained set of tools and API calls. They built a pattern of states of code, and mapped when and how often the students transitioned among these states. They were able to group the students into three groups based on their state transition patterns. They also looked at the students' mid-term grades, and found significant and consistent differences among those three groups. Students of a particular group performed similarly on the exam as others in that group. This was encouraging, that this method provides results that can be used immediately for course improvement, such is identifying flailing students, and was validated against a traditional assessment- the exam.

Building from Piech, another work, Blikstein 2014, examined that data further, and performed an additional study [2]. Results include that time spent tinkering did not relate to course performance. However, the ability to shift mental modes, such as from tinkering to planning, did relate.

978-1-4673-8367-7/15/$31.00 ©2015 IEEE

Many of the techniques outlined by these papers are already in use. Lipman built a system that shares similar goals with the one we propose here, which collected fine-grain, atomic interaction data from the development environment in a university CS1 course [4]. Such studies are excellent templates to leverage for the work discussed here.

V. Research Design

Most of the studies in related literature implemented a highly-constrained problem for the students to solve under observation. Piech used Karel the Robot [5], and Berland used another limited-input virtual robot [1]. Our first study will implement a similarly constrained problem, where the we will have a reduced world in which the students will act. This study will be conducted with a small cohort, both with the instrumented programming environment, and under researcher observation of their behavior. This small test, with both observation and data history, will be used to calibrate our understanding of what the data from the instrument may mean. With this knowledge we will be able to finalize the design for the second-stage experiment.

The second phase experiment will take place within a CS Principles MOOC. One particular assignment will be selected for study, just as with other. This environment for the students will be less constrained, but still have a specific goal and well-defined expected path towards that goal. This experiment will have a large number of participants, likely hundreds, and will not involve in-person observation, relying entirely on the instrumentation data.

We are considering a third phase, where we would invite any student working on App Inventor in any class in the world to participate. They would declare to us, by a short entry survey, what they are trying to accomplish, and opt in to data collection while they work towards their goal. Knowing their (self-reported) goals, we will expand our analysis techniques to work on larger, less specific data sets. These students will not all have the same goals, but much of their projects and their interactions with their projects may have similar properties that we can extract and compare against their coded goals, in aggregate. This study could have upwards of 10,000 students involved.

We are expecting to discover a certain class of work trajectories, such as: repeatedly undoing and redoing a change, exhaustively looking through block menus without recruiting a block, and removing code from execution path, putting blocks off to the side.

The trajectories in this list may or may not appear as strong signals in testing, but serve as examples of the types of patterns we believe we will identify with this method. This class of patterns may be suitable for a future real-time analysis tool, which has the potential to assess when specific students require particular interventions to improve their understanding.

VI. Future Work

Studies do exist that compare graphical and text languages with students, such as Lewis 2010, but the intersection of comparable features and comparable learning goals is often small, limiting the experimental design, and therefore offer limited generalizability. All studies that try to directly compare any two systems will have to overcome that different systems do better at certain things, usually by design. In designing the experiment, it is all too easy to set up one language or the other to fail. David Weintrop is working on overcoming this bias, and is developing assessments that work equally well in Snap!, JavaScript, and hybrid environments [7].

There are two kinds of analysis in this space: real-time and ex-post-facto. The methods described here are the latter-depending on snapshots captured from an entire and complete programming experience. The former is also of interest, as such tools could monitor students while they are working, detecting patterns that signal gaps in comprehension, and triggering a need-based intervention to aid that student. The intervention could be a pop-up referring to a specifically informative help page, a notification to an instructor to attend to the student, or something else. Follow-up work on tools like these will ensue promptly.

This paper describes a method for assessing student work progression in App Inventor (a blocks language), along with data-driven conclusions about student development behavior with this language. The methods described here may be developed further to create a meta-descriptor of programming process, which can provide a future research vector towards a neutral method of comparing learning in dissimilar languages. This meta-descriptor of language interaction will require identification of analytic measures (suitable for data analytics and/or machine learning) that are common to all (or many) languages, and a taxonomy of how those measures are represented in individual systems. This method will be informed by the work described here, and we hope to carry on this path of research for many years, towards that lofty goal.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1433592. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] M. Berland, Martin, Benton, Smith, and Davis. Using learning analytics to understand the learning pathways of novice programmers. Journal of the Learning Sciences (2013), 22(4):564–599.

[2] P. Blikstein, Worsley, Piech, Sahami, Cooper, and Koller. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. J. Learning Sciences (2014), 23(4):561–599.

[3] C. M. Lewis. How programming environment shapes perception, learning and goals: Logo vs. scratch. SIGCSE (2010), 346–350.

[4] D. Lipman. LearnCS!: a new, browser-based C programming environment for CS1. J. Comput. Sci. Coll. 29, 6 (June 2014), 144-150.

[5] C. Piech, Sahami, Koller, Cooper, and Blikstein. Modeling how students learn to program. SIGCSE (2012), 153–160.

[6] M. Sherman, Martin. The assessment of mobile computational thinking. 2015. J. Comput. Sci. Coll. 30, 6 (2015), 53-59.

[7] D. Weintrop. Minding the gap between blocks-based and text-based programming: Evaluating introductory programming tools. SIGCSE (2015). ACM.