See the Collaboration Through the Code: Using Data Mining and CORDTRA Graphs to Analyze Blocks-Based Programming

Mike Tissenbaum, University of Illinois at Urbana-Champaign, miketissenbaum@gmail.com
Vishesh Kumar, University of Wisconsin-Madison, vishesh.kumar@wisc.edu

Abstract: This paper describes an exploratory study that leveraged data mining, qualitative analysis, and Chronologically-Ordered Representations of Discourse and Tool-Related Activity (CORDTRA) diagrams to identify and analyze key moments in students' collaborative app building during a 12-week computing curriculum. Our analysis showed that two key practices emerged: 1) Students leveraged their past work and tutorials to support their app development, both on their own and with peers; and 2) Students largely developed their own parts of group apps without feedback from peers or referencing prior work. We discuss how patterns revealed in this mixed-methods approach affected how students constructed code, with an eventual goal of identifying how these patterns shaped students' final projects.

Introduction

Since Papert's early work with Mindstorms, collaboration has been an important part of computing education (Resnick, Ocko & Papert, 1988). While numerous studies have shown that collaborative approaches, such as pair programming, can effectively support collaborative learning in computing education (Preston, 2005), there are questions around how collaboration is enacted by different groups and over various scales of time (Kafai et al., 2012; Lewis & Shah, 2015; Goel & Kathuria, 2010). Further, it is challenging to assess collaboration in computing education, as collaborative processes occur both on screen and physically between collaborators. Much of the research about collaboration in computing education focuses on the end product of students' work or their perceptions of collaboration, rather than students' collaborative practices and how these practices support the work students are doing (Williams et al., 2002). While there has been some research aimed at revealing the effectiveness of collaboration during real-time interactions of students (Fields et al., 2016; Litts et al, 2017), the majority of this research has largely been conducted during short lab- based activities (e.g., Grover et al., 2016). As such, there is a gap in our understanding of how collaboration in computing education plays out over sustained periods of time. In response, we must analyze students' collaborative practices over longer periods of time. However, as noted by Wise and Schwarz (2017), there are significant challenges in conducting rich qualitative analysis of students' engagement in longer activities. When students engage in collaborative activities over weeks, or even months, how can researchers select moments in time for cohesive, meaningful analysis? In response to this challenge, this paper describes an exploratory approach for combining data mining and qualitative methods to identify key moments in students' collaborative app development, as a means for understanding how students collaboratively make use of their own knowledge and that of their peers. Two questions guided our work:

1) Within a long-term programming project, how can we effectively identify key moments in students' group projects for analysis?
2) How can qualitative analysis shed light onto the knowledge-seeking and collaborative practices students engage in when these moments arise?

Below, we discuss our multidimensional approach for conducting analysis of collaboration within the context of a high school computing class. We discuss the patterns that emerged, their importance for understanding collaboration in computing education, and next steps for applying these analyses.

Methods

This study involved 22 students (21 male, 1 female) in a large urban high school in the North Eastern United States. The school is one of the most diverse schools in the US (25% Asian, 23% Hispanic, 20% Black/African American, and 29% White), and 61% of the students are eligible for free or reduced lunch.

Using a computational action approach (Tissenbaum, Sheldon & Abelson, 2019), the students took part in a 12-week curriculum in which they used MIT App Inventor – a blocks-based programming language that allows students to build fully native Android mobile apps without needing to learn the syntax of code – to develop apps that had direct impacts in their community. For this study, students built apps aimed to help clean and bring awareness to pollution issues of a large river running though their city. Students spent the first eight weeks of the curriculum learning about App Inventor and developing starter apps as an introduction to the system and the ways they could develop apps with real-world applications. During the final four weeks, students worked in pairs or

triads to design and build their own apps. This version of App Inventor was specially designed to allow students to collaboratively develop group apps in real-time on multiple computers, rather than having to take turns (e.g., as in pair programming). This provided a unique opportunity to understand how students collaborated and developed their apps when each student was free to work on their own sections. For the purposes of this study, we only analyzed the apps that groups built during the final four-week period.

Log data was collected each time a change was made to a group's project. This data included: (i) which student made the change, (ii) which element (i.e., block) was changed, (iii) the element's beginning and ending location on the screen (i.e., if and where it was moved), and (iv) the timestamp of the event. Screen and voice recordings were also captured for each student who gave informed consent using *Screencastify* (a Chrome browser extension). *Screencastify* was embedded within our special instance of App Inventor so that it launched when students logged into App Inventor. It also automatically uploaded recordings to our secured server, along with start and stop timestamps and students' login IDs, when students logged out or closed the browser window. By synching the screen recordings with App Inventor, we ensured that timestamps in the log files exactly matched timestamps in the screen recordings.

Using the log files, we implemented a data mining approach that provided timestamps for each episode of students adding particular blocks (e.g., a block to create a list, open a new screen, store data to a database, or implement a map function) into their app for the first time. We were interested in those instances because they marked times for further investigation of why students decided to add a particular block. We also planned to examine how students figured out how to use the block in their app. Using data mining to determine these moments helped us to reduce the time-consuming (and error prone) process of manually watching and coding screen recordings of students' work, as critical incidents of students' collaboration.

Once the episodes of students' app building were identified using the data mining approach above, we triangulated this data with the screen and voice recordings to examine students' work immediately before and after the student added the new block. We used a combination of inductive and deductive coding to develop 12 codes for students' interactions in App Inventor around these episodes (Table 1). We began coding students' work starting at two minutes before the block was added in order to capture relevant discussion or coding students may have engaged in prior to adding the block. We coded for at least two minutes after the block was added. In cases where the students continued to work with the block or elements connected to it, we continued to code the video until students moved on to a new task. In some cases, two or more blocks were added during the same episode. In these cases, we considered the event as continuous and coded them together. To allow finer-grained analysis and visualization of student work, we segmented each episode into discrete 15-second blocks. We included event codes if they happened any time during the 15-second block. Each code was only marked once, regardless of how many times it happened during the 15-second block. We coded a total of eleven episodes (mean duration 15.22 minutes). These episodes were chosen as they were the ones that contained the identified events and all group members consented to have their voices recorded (which was needed to analyze the collaborative discourse).

Table 1. Event codes for video analysis of student app building

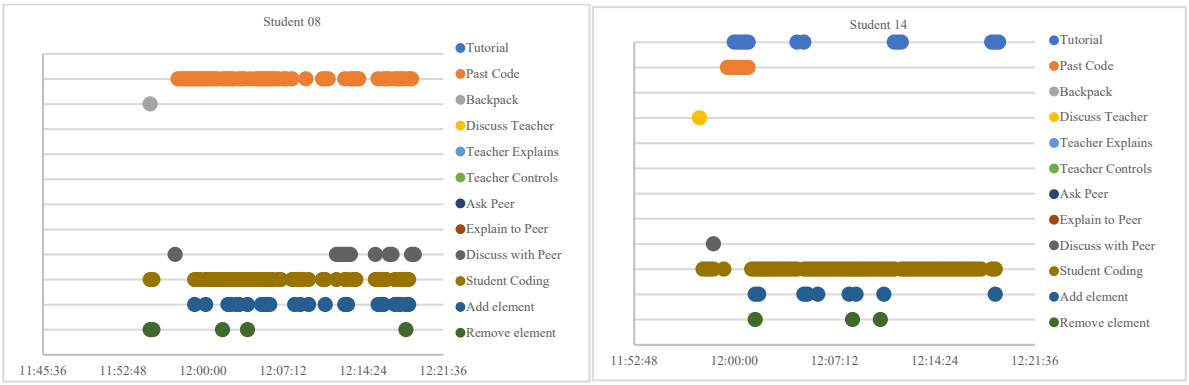
Coded Event	Description	Coded Event	Description
Tutorial	Student looked at a tutorial from a previous exercise	**Past Code**	Student looked at code from a previous project or exercise
Backpack	Student used the backpack (a way of storing and retrieving code across projects)	**Discuss with Teacher**	Student discussed their work with the teacher
Teacher Control	Teacher takes control of the student's computer	**Ask Teacher**	Student asks the teacher for help
Explain to Peer	Student explains how something works in their app	**Discuss with Peer**	Student discusses how they might do something in their app
Ask Peer	Student asked a peer how to do something in their app	**Student Coding**	Student is building/coding their app
Add Element	Student adds an element (i.e., a coding block or a button) to their app	**Remove Element**	Student removes an element from their app

Event codes were added by the two authors. The two authors independently coded 20% of the data to establish intercoder reliability. Across all twelve codes, the combined intercoder agreement was 96% (Cohen's kappa = .852). The two authors coded the remaining data separately.

In order to better understand the multiple processes involved as students collaboratively developed their apps, we plotted a timeline of event codes using Chronologically-Ordered Representations of Discourse and Tool-Related Activity (CORDTRA) diagrams (Hmelo-Silver, Chernobilsky & Jordan, 2008). CORDTRA diagrams are particularly useful for these kinds of activities, as they combine log data with fine-grained event coding in a visualization that allows for more holistic analysis than purely text-based coding schemes (Hmelo-Silver et al., 2008). Below, we discuss our findings on students' collaborative practices and tool use when introducing new elements to their apps.

Results

Across the 11 episodes, a few patterns emerged. One pattern was that of students revisiting past work as a reference on how to implement a new block in their app (see Figure 1). Students moved back and forth between the app they were currently building and earlier exercises and tutorials they had completed during the first eight weeks of the curriculum. In a similar episode (not shown here), student went back and re-used whole pieces of code from their backpacks in their new project. In these cases, students would sometimes talk with their peers to clarify what they were doing; in other episodes, they largely worked alone. Across both examples in Figure 1, the students were observed doing some variation of "copy and paste." In the tutorial and past-code cases, students used the same, or very similar, names for variables and labels, even if they did not fit their current apps. Similarly, when the students re-used code snippets from the backpack, they focused on removing errors, rather than creating appropriate names for their variables or blocks. When errors occurred, students tended to engage in "guess and check" debugging (i.e., semi-randomly changing variable names) rather than focused debugging. This seems to indicate that they students were focused more on *making the apps work* than understanding *how the apps work*.



a. b.

Figure 1a. an example of a student looking at past code and discussing with a peer;
Figure 1b. a student using past code and past tutorials, but largely working alone.

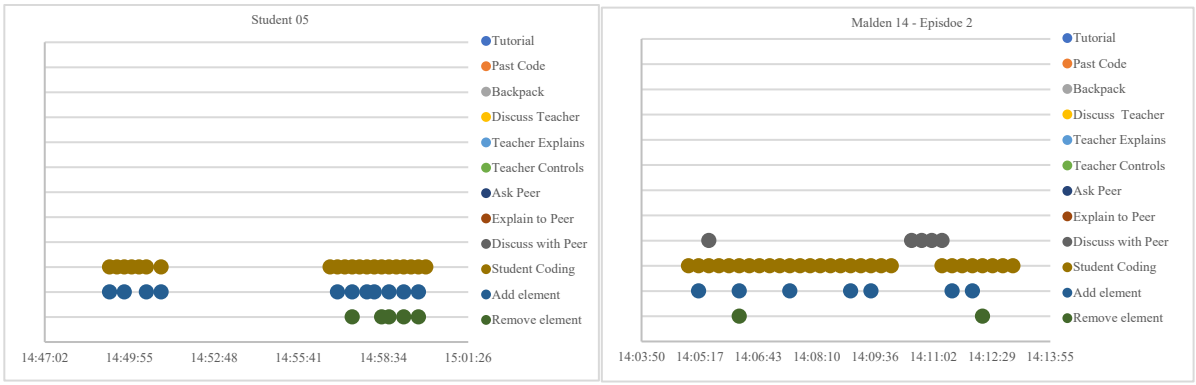


Figure 2a. An example of a student lacking a debugging strategy.
Figure 2b. An example of lulls in student work after a successful coding strategy.

Another pattern that emerged was that of students rarely, if at all, revisiting past code as a means of supporting their app building. Instead, students attempted to figure out codes on their own or through discussion with their group members. In this pattern, we found some important differences between students working on

their own compared to when they talked with their peers. When students did not talk with their peers, we observed that students seemed to aimlessly test problem-solving strategies without a clear debugging strategy. This was punctuated by long gaps in their coding, in which students did little (see Figure 2a). In this episode, the student was trying to get maps to work, but he used the wrong element (a canvas drawing element). Eventually, the student simply removed many of the elements and blocks without resolving the issue. In the second example (Figure 2b), the student was able to implement the code (i.e., change screen) without any significant complications. The only break in coding (at 14:11:02) was to discuss work their peer was doing. However, it is worth noting that after successfully completing this code, the student did no additional work for over 10 minutes.

Discussion

This paper is an exploratory attempt to combine data mining, qualitative analysis, and chronological visual representations of students' interactions as they developed their own learner-driven mobile applications. While only a first step in our analysis, this multi-dimensional approach provided insights into key moments in students' app development, problem-solving strategies, and collaboration. While some clear patterns emerged, a critical next step is to analyze the groups' final projects and examine how code developed during these episodes manifested itself in final products. For example, do students re-use or discuss past code in their final projects? Do any of these patterns result in more functional apps? If so, using similar mixed-methods approaches may help us provide necessary help to students in real-time by providing context- or goal-specific scaffolds, such as suggesting that students revisit prior projects for similar blocks of code. Understanding students' collaborative practices in computing education also helps us to bridge collaboration across coding on-screen and in the physical world.

References

Fields, D. A., Quirke, L., Amely, J., & Maughan, J. (2016, February). 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* (pp. 150-155). ACM.

Goel, S., & Kathuria, V. (2010). A novel approach for collaborative pair programming. *Journal of Information Technology Education: Research*, *9*, 183-196.

Grover, S., Bienkowski, M., Tamrakar, A., Siddiquie, B., Salter, D., & Divakaran, A. (2016, April). Multimodal analytics to study collaborative problem solving in pair programming. In *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge* (pp. 516-517). ACM.

Hmelo-Silver, C. E., Chernobilsky, E., & Jordan, R. (2008). Understanding collaborative learning processes in new learning environments. *Instructional Science*, *36*(5-6), 409-430.

Kafai, Y. B., Fields, D. A., Roque, R., Burke, W. Q., & Monroy-Hernandez, A. (2012). Collaborative agency in youth online and offline creative production in Scratch. *Research and Practice in Technology Enhanced*.

Lewis, C. M., & Shah, N. (2015, July). How equity and inequity can emerge in pair programming. In *Proceedings of the eleventh annual International Conference on International Computing Education Research* (pp. 41-50). ACM.

Litts, B. K., Lui, D. A., Widman, S. A., Walker, J. T., & Kafai, Y. B. (2017). Reflections on Pair E-Crafting: High School Students' Approaches to Collaboration in Electronic Textiles Projects. Philadelphia, PA: International Society of the Learning Sciences.

Preston, D. (2005). Pair programming as a model of collaborative learning: a review of the research. *Journal of Computing Sciences in colleges*, *20*(4), 39-45.

Resnick, M., Ocko, S., & Papert, S. (1988). LEGO, Logo, and design. *Children's Environments Quarterly*, 14-18.

Tissenbaum, M, Sheldon, J. & Abelson, H. (2019). From computational thinking to computational action. *Communications of the ACM*.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, *12*(3), 197-212.