WIP: Characterizing Student Programming Activity

Ruben Acuña
School of Computing and Augmented Intelligence
Arizona State Univerity
Mesa, Arizona, USA
Ruben.Acuna@asu.edu

Ajay Bansal
School of Computing and Augmented Intelligence
Arizona State Univerity
Mesa, Arizona, USA
Ajay.Bansal@asu.edu

***Abstract*—This work-in-progress research paper analyzes student activity as captured by the sequence of programming submissions they make. Although students who complete assignments in a computing course are often assessed in a summative way through the submission of a completed program, their activities during development provide additional insights into their work and problem-solving processes that cannot otherwise be captured. In the context of a data structures & algorithms course taught at the sophomore level, we first examined the number of attempts that students made while completing assignments throughout the semester. We then examined the cumulative portion of the class that had submitted an assignment relative to the due date, and the time periods after assignment release that students are active. Our work helps to illustrate how a programming activity trace provides a unique source of information on how a submission evolves. It was found that even with only feedback as motivation, students made many submissions and that number increased during the semester. Our initial work indicates that a substantial amount of data is available and helps suggest what aspects might provide information useful for further analytics using traditional statistics or machine-learning approaches.**

***Index Terms*—exploratory data analysis, educational data mining, automated assessment, programming activity**

I. Introduction

Teaching an effective programming process in computing courses is essential [1]. However, most students are assessed only based on final results, which provides only an indirect measure of their process and problem-solving ability [2]. Programming's process-centric nature means that neither programming homework nor exams are sufficient to measure student ability [2], [3]. Student activity is typically opaque due to two limitations: 1) Only the final submission is collected for assessment. 2) Evaluating formative versions of a program is time-intensive when performed by graders. These limitations can be addressed through automated assessment tools (AAT) coupled with statistical and data mining approaches.

AATs are often deployed into introductory computing courses to support scaleable and reproducible assessment [4]. Students working on an assignment upload their work to an AAT, which typically works by executing test cases (and potentially running static analysis tools) that examine the submission and evaluate how well it meets various requirements. As a side effect, these tools can collect a trace of student activity as they are used to grade student submissions and provide feedback. Since programming is an iterative process, this data significantly helps to characterize the actions of a learner. As students work through an assignment, they make a series of submissions, which we call a *trace* (see [16]). Each submission is evaluated according to some standard and an evaluation aligned to a rubric is produced. In its simplest form, this evaluation looks like a binary vector where each element corresponds to a rubric criteria, and true is used to indicate that that criteria was met (false otherwise). Each entry in the trace is also accompanied by metadata such as the student's ID, the earned score, and the time of submission.

Our goal is to leverage this source of information to understand the process by which students complete programming assignments. To this aim, we have conducted an exploratory data analysis on the submission activity of students. Without sufficient data, traces provide no more information than final submissions. In addition, examining submissions enables us to see how (when) students work on assignments. For this preliminary work, we have focused on three aspects of trace activity to investigate:

1) The number of attempts (submissions) students made.
2) The cumulative number of submissions made as an assignment's deadline approaches.
3) The activity patterns (i.e., when students are working) on assignments as the deadline approaches.

II. Related Work

Large courses often use automated assessment to effectively return grades to students [5]. These tools use a combination of techniques, ranging from unit tests to industry static analysis tools such linters, to analyze the state of a student's submission to return feedback and assign a grade [6], [7]. AATs include functionality to execute assignments, return feedback to students, and then synchronize grades with a learning management system [8]. Instructors often develop their own AATs, adapted to the needs of their class [9].

A variety of data-driven approaches have been developed for programming assignment sequence data. Carter and Dewan [10] leveraged student activity during programming to detect when they had difficulties. They monitored the IDE actions students took during development and used a decision tree model to predict if they needed help. Pensieve is an environment that teaching assistants can use to review a student's software development process and to make suggestions to them during tutoring [11]. It has been shown that log-based feedback improves grades [12]. An appropriately designed

AAT can capture data about the process that a student uses to complete an assignment. The results from the system Web-Cat have been analyzed to determine general patterns such as the relation between number of lines/comments and the portion of the assignment that has been completed [13]. Following the sequence of actions students perform can be used to judge the effectiveness of their process [14]. For example: checking if the order they implement methods follows a rule about doing simplest things first. Several works create a graph from the union of the activity sequences that students use to complete assignments. The Hint Factory is a technique for automatically generating contextualized hints based on how students previously completed assignments, which was applied in software for teaching deductive logic [15]. The approach used by Stamper et al. in the Hint Factory involved constructing a graph from the sequence of actions that students took when completing a proof. In another work, a graph was constructed to inform instructors about how students are completing assignments [16]. Graph features are leveraged to determine how students work (e.g., sink nodes indicate where students get stuck), and the common path they take when completing it. Another approach uses a graph for hint generation [17]. A student's submission is compared to other submissions within a graph, and a similar submission (node) is identified. Based on that node's position in the graph, a code change will be hinted that will help the student to progress in that graph to the final completed state.

III. Context

We gathered data from a course on data structures & algorithms taught in spring 2022. Students take this class after completing two semesters of Java and one semester of discrete mathematics. The course was taught face-to-face at our university in a 16-week semester with 43 undergraduate students. The course uses Algorithms by Sedgewick and contains 12 modules of material corresponding to sections of that text. In this work, we focus on the modules containing a programming assignment that was automatically assessed:

- **Module 1 (Matrix)**: Implement an immutable matrix class using standard OOP and a 2D array.
- **Module 2 (Deque)**: Implement a double-ended queue using a doubly linked structure.
- **Module 5 (Sorting)**: Implement data generation methods and benchmark various algorithms.
- **Module 6 (Merging)**: Implement three algorithms related to mergesort.
- **Module 9 (BSTs)**: Implement various methods for manipulating a binary search tree.
- **Module 10 (Hash Tables)**: Implement chaining and linear probing hash tables.

Following best practices for instruction (see [4]), students were allowed unlimited submissions (without a delay mechanism) before the deadline. Relevant to our later visualization of submission times, the course syllabus included a strict policy for late submissions. Late submissions were not accepted unless a student used a late pass (they had two). Using a late pass allowed students to submit their assignment up to 24 hours after the original due date (which was 11:59 pm).

Data was collected using an AAT developed to assess students in our course by applying static and dynamic analysis [18]. Static analysis is used to support checking for non-functional requirements such as performance, while test cases are used to evaluate the behavior of the student's submission dynamically. The tool is implemented in Python and runs on the Gradescope platform [19]. The data captured by the tool was preprocessed by removing students who dropped the course since their submissions may not represent the student population as a whole. Students who failed but completed the course were included.

IV. Attempt Count

As a step towards determining the scale of the data that can be collected with an AAT, we visualized the number of attempts (submissions) that students made. If most students made only one attempt, that would indicate that trace data holds little additional information over inspecting a final result. In this course, students were given no incentive to submit to the autograder except for the ability to gather feedback about the state of their assignment. We looked at the total number of attempts for each student, which formed a distribution.

Table I contains summary statistics that describe this distribution for each assignment. The weight of the assignments is provided (and may be compared) since students may use them to assign priority (i.e., higher-weight assignments receive more effort). The sample size is expected to vary since not every student submits every assignment. The minimum number of attempts is 1 since otherwise it is excluded from the distribution. In Fig. 1, histograms show the detailed distributions of attempts for the M1, M2, M5, and M9 assignments. These were selected as they span across the semester. M2 is additionally given since it was extended (see further discussion in Sec. V).

TABLE I
Summary statistics for the number of attempts.

Module	Weight	Min	Max	Mean	SD	N
1 (Matrix)	32	1	47	11.05	11.84	39
2 (Deque)	32	1	64	10.26	12.49	39
5 (Sorting)	24	1	46	6.0	7.46	40
6 (Merging)	32	1	38	7.42	8.22	40
9 (BSTs)	40	1	105	18.43	20.78	37
10 (Hash Tables)	50	1	75	16.05	18.75	37

The general trend seen in the figures seems to be for the number of attempts made per student to increase over the semester. On the first assignments, several students made only one submission: 7 for M1, 5 for M2, and 7 for M5. Later in M9, only 1 student made 1 submission. This may have been due to 1) the increased difficulty of the assignment or 2) increased familiarity with the automated assessment process. The former might also stem from the increased weight of the assignment, with students seeing a higher weight as an indication that they need to put in more effort (even before they

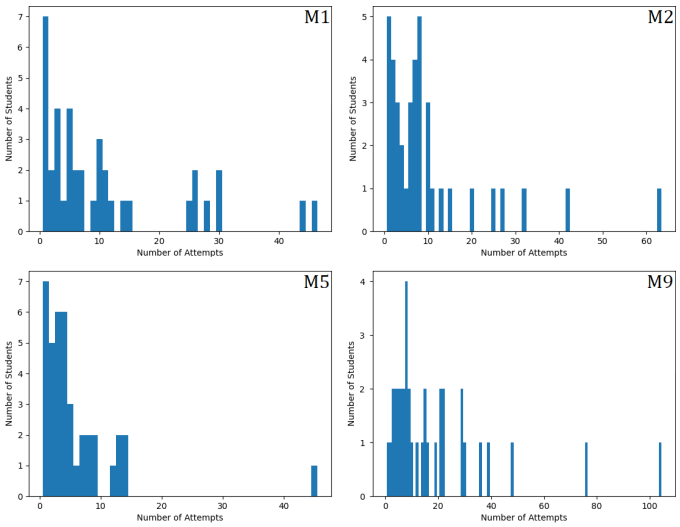

Fig. 1. Distribution of attempts for M1 (top left), M2 (top right), M5 (bottom left), and M9 (bottom right)

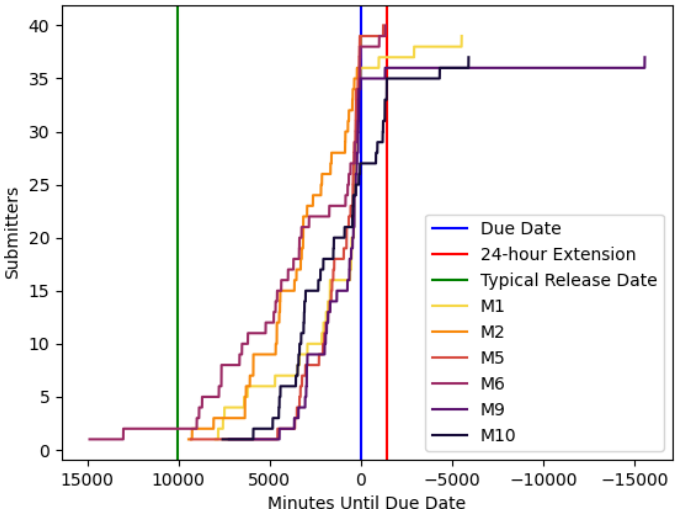

Fig. 2. Cumulative number of submitters.

have assessed the assignment for themselves). The increase is less evident in Table I, which shows an increased attempt count for M10 but a decrease for M5 and M6. For M5, the lower assignment weight/difficulty may not have necessitated as many submissions. M6 overlapped with spring break, which may distort data. In general, our results suggest a baseline amount of data that can be gathered. Providing incentives for students (beyond autograder feedback) should help to increase the attempts (and therefore data availability for analysis).

V. Cumulative Submitters

Another way to quantify the activity of students in making submissions is the cumulative number of submitters. At the beginning of an assignment, this number is always 0 (no student has submitted) and then increases until reaching the total number of students that submit that assignment. The same process was used for each assignment to compute the number of submitters. For each student, we examined timestamps on each of their submissions and calculated the time remaining (in minutes). At each point in time t (in minutes), we look at each student and count those who previously made a submission.

Fig. 2 shows the number of submitters. In this figure, the trends for each of the assignments in the course are shown together. The color scheme has been set so that earlier modules are plotted in a lighter shade, while later modules are dark. The figure has been annotated with vertical lines to indicate three key points in the time: green is the typical assignment release date (one week before due), blue is the due date, and red is a 24-hour extended due date that students may request. Note that submissions for M6 begin early due to the assignment being released early (spring break). Also note that M2 was extended by 24 hours (see Fig. 4). For M1, M9, and M10, some students had unexpected technical issues that prevented them from submitting by the original or extended deadline.

There appears to be a pattern on the early assignments (M1, M2) where students did not start submitting until a few days had elapsed. On M1, this was followed by a gradual increase in submissions. For M2, students still waited to start but a more significant fraction of the class submitted in advance of the deadline. Students appear to be submitting more early during the middle of the semester (M6). This suggests that students are finding some benefit in submitting to the system (since we do not directly incentivize it, students only gain from the feedback). This pattern does not apply to M5, but it seems that (from the previous section) students proceed through that assignment in a different way. Interestingly, students submit later on the assignments due nearer the end of class (M9, M10). Our characterization of attempts already suggests that those assignments are more complex than the previous ones or that students are more comfortable with the system at that point in the semester. Yet, students submit closer to the deadline. This may indicate that students are too busy with other assignments (perhaps having switched to an earlier-deadline-first approach for their homework) or that they are tired and find it difficult to start the assignment.

VI. Active Students

We also examined how many students were concurrently active on the assignment at times leading up to the due date. When no students are submitting work for assessment, there are zero active students. If a student submits, they become *active* for 60 minutes. This threshold was chosen based on preliminary student discussions and may need to be adjusted. For each assignment, the same process was used to compute the number of active students. For each student, we again calculated the time remaining (in minutes). For the entire period that the assignment was available, then then determined if at a given point in time (call it t; in minutes) if a student was active. A student is active at time t if they made a submission was made within the previous 60 minutes.

Fig. 3 through Fig. 6 show the number of active students for the M1, M2, M5, and M9 assignments. Similar to our examination of attempts, these four modules were selected

since they span the semester. Results for all six of the modules are shown side by side in Fig. 7. When the assignments are released, no students are working on them, therefore the number of active submitters is zero. During the early portion of when the assignment is live, it is often the case that only one student (out of the 43 in class) is active at a time. As expected, the number increases as the deadline approaches. For example, in M1 we see a peak of 13 active students around 6 hours before the deadline. The results for M2 in Fig. 4 are unique among the modules we analyzed due to an extension the instructor made. The assignment is not more complicated than M1 but students requested extra time. The extension was announced in class approximately 12 hours before the original due date, which was extended 24 hours. This assignment thus shows us how extensions impact student working patterns.

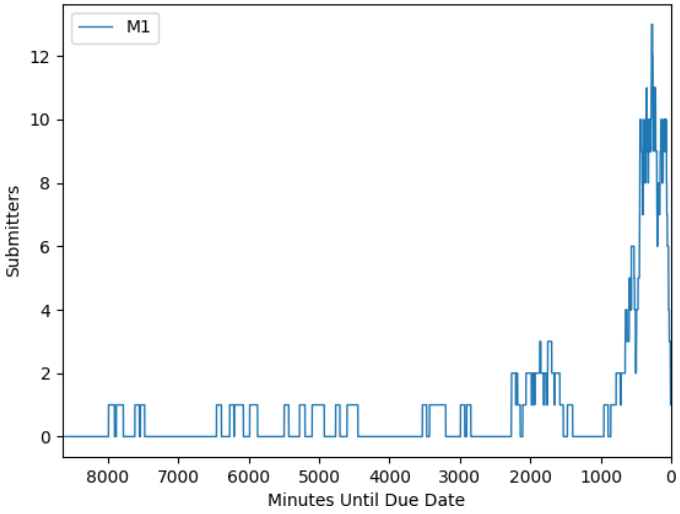

Fig. 3. Active students during M1.

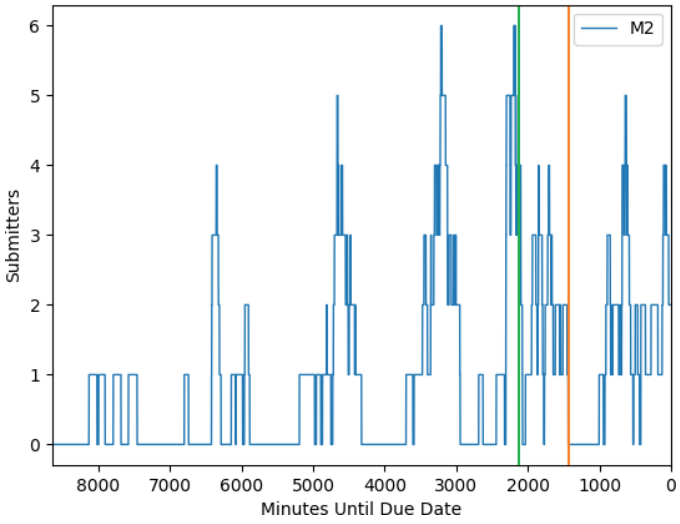

Fig. 4. Active students during M2. Green vertical bar indicates when a 24-hour extension was announced. Orange indicates the original due date.

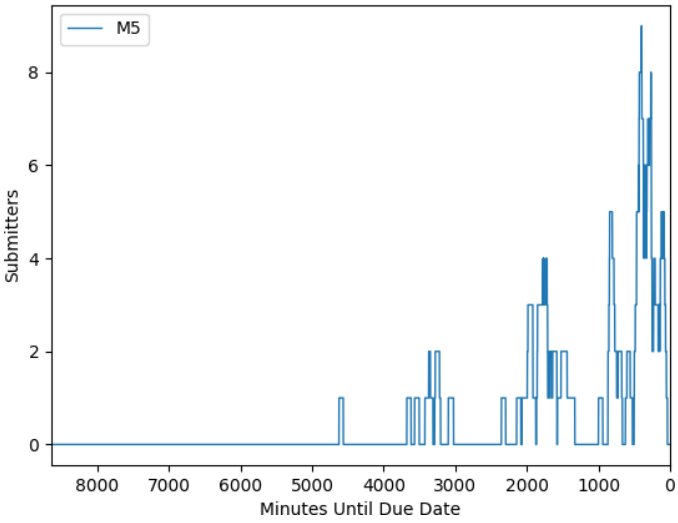

Fig. 5. Active students during M5.

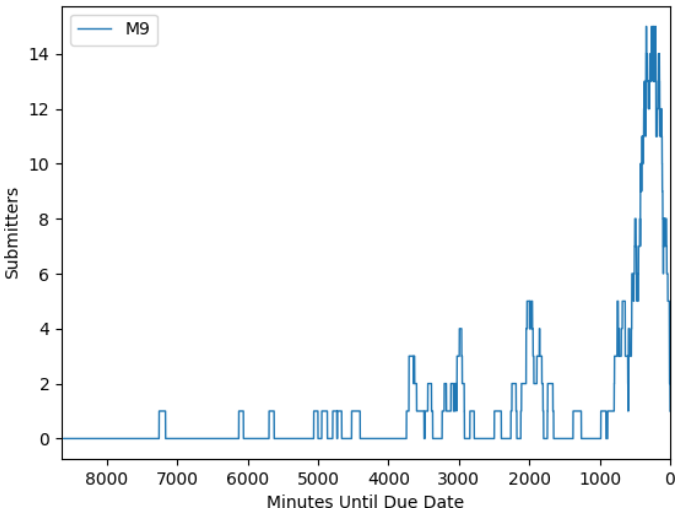

Fig. 6. Active students during M9.

The most immediate pattern in the data is a periodic working cycle. Fig. 7 illustrates how students work in bursts, which are from late morning to the end of the day. Students are inactive during the early morning hours. In most cases, there is a peak of activity about six hours before the deadline. This is interesting because it indirectly suggests that only a small number of students are rushing to complete it at the end. However, this pattern doesn't hold for M1, which shows a rush until the end, which is perhaps not unexpected if students are still adapting to the course workload.

The results for M2 show an almost immediate drop in the number of students actively working on the assignment when the extension was announced. Almost immediate is expected since we use a 60-minute window to determine activity. Activity resumes several hours later but is not as intense as earlier in the day. After the periodic lull in activity that is typically seen, activity on the extended day continues. The activity at this time appears more regular and does not show a

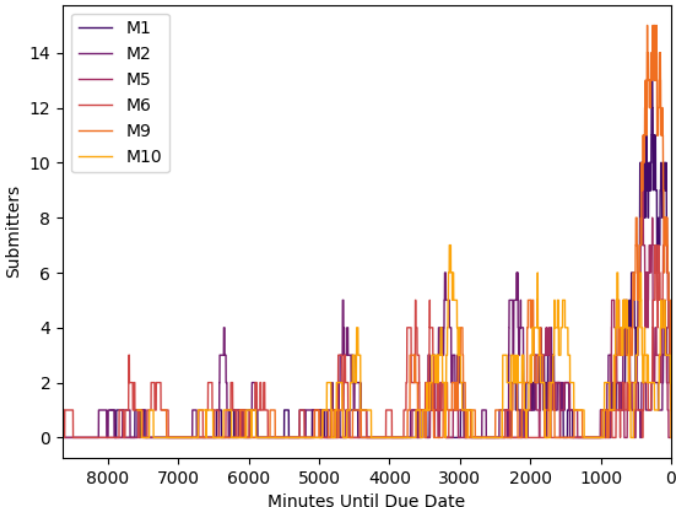

Fig. 7. Active students for all modules. Data is aligned to due date.

strong activity spike as seen in the other assignments (without extensions). This pattern suggests that applying an extension led students to have a more consistent working pattern on the assignment. It did not appear to increase procrastination (which we would expect if the same strong spike occurred at the end of the submission window).

VII. Conclusion

In this work, we have described our exploratory analysis of the activity trends within autograder data captured in a sophomore course on data structures & algorithms. Our next step is to develop research questions and formal hypotheses to investigate them. For example: 1) Do students submit sooner and more often as the semester goes on? 2) What is the correlation between aspects of trace and assignment (or other) grades? 3) Can we extract evidence that starting an assignment early helps? In addition, we have access to data for students in an online version of this course as well. This suggests the meta-question of: are the above trends impacted by online vs face-to-face students? For most of these hypotheses, traditional statistical approaches are sufficient. However, as we continue to investigate this data, we are interested in applying time series approaches from data mining to problems such as if a student will benefit from an extension, which will also benefit from using the assessment data itself.

References

[1] J. Bennedsen and M. E. Caspersen, "Revealing the programming process," in *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 186–190. [Online]. Available: https://doi.org/10.1145/1047344.1047413

[2] C. Daly and J. Waldron, "Assessing the assessment of programming ability," *ACM SIGCSE Bulletin*, 2004.

[3] J. Bennedsen and M. E. Caspersen, "Assessing process and product," *Innovation in Teaching and Learning in Information and Computer Sciences*, 2007.

[4] V. Pieterse, "Automated assessment of programming assignments." Open Universiteit, Heerlen, 2013, p. 45–56.

[5] J. C. Paiva, J. P. Leal, and A. Figueira, "Automated assessment in computer science education: A state-of-the-art review," *ACM Trans. Comput. Educ.*, vol. 22, no. 3, jun 2022.

[6] Z. Ullah, A. Lajis, M. Jamjoom, A. Altalhi, A. Al-Ghamdi, and F. Saleem, "The effect of automatic assessment on novice programming: Strengths and limitations of existing systems," *Computer Applications in Engineering Education*, vol. 26, no. 6, pp. 2328–2341, 2018.

[7] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," *ACM Trans. Comput. Educ.*, dec 2023, just Accepted. [Online]. Available: https://doi.org/10.1145/3636515

[8] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. Association for Computing Machinery, 2010, p. 86–93.

[9] J. C. Caiza and J. M. Del Alamo, "Programming assignments automatic grading: review of tools and implementations," in *7th international technology, education and development conference (INTED2013)*, 2013, p. 5691.

[10] J. Carter and P. Dewan, "Mining programming activity to promote help," in *ECSCW 2015: Proceedings of the 14th European Conference on Computer Supported Cooperative Work, 19-23 September 2015, Oslo, Norway*, N. Boulus-Rødje, G. Ellingsen, T. Bratteteig, M. Aanestad, and P. Bjørn, Eds. Cham: Springer International Publishing, 2015, pp. 23–42.

[11] L. Yan, A. Hu, and C. Piech, "Pensieve: Feedback on coding process for novices," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 253–259. [Online]. Available: https://doi.org/10.1145/3287324.3287483

[12] H. Meier and M. Lepp, "Effectiveness of feedback based on log file analysis in introductory programming courses," *Journal of Educational Computing Research*, vol. 61, no. 3, pp. 696–719, 2023. [Online]. Available: https://doi.org/10.1177/07356331221132651

[13] A. Allevato and S. H. Edwards, "Discovering patterns in student activity on programming assignments," in *ASEE Southeastern Section Annual Conference and Meeting*, 2010.

[14] R. Acuña and A. Bansal, "Assessing student programming process using automated reasoning," in *2023 IEEE Frontiers in Education Conference (FIE)*, 2023, pp. 1–8.

[15] J. Stamper, T. Barnes, L. Lehmann, and M. Croy, "The hint factory: Automatic generation of contextualized help for existing computer aided instruction," in *Proceedings of the 9th International Conference on Intelligent Tutoring Systems Young Researchers Track*, 2008, pp. 71–78.

[16] R. Acuña and A. Bansal, "Using programming autograder formative data to understand student growth," in *2022 IEEE Frontiers in Education Conference (FIE)*, 2022, pp. 1–8.

[17] K. Rivers and K. R. Koedinger, "Data-driven hint generation in vast solution spaces: a self-improving python programming tutor," *International Journal of Artificial Intelligence in Education*, vol. 27, pp. 37–64, 2017.

[18] R. Acuña and A. Bansal, "Improving student learning with automated assessment," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 464–470. [Online]. Available: https://doi.org/10.1145/3649217.3653603

[19] A. Singh, S. Karayev, K. Gutowski, and P. Abbeel, "Gradescope: A fast, flexible, and fair system for scalable assessment of handwritten work," in *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*. Association for Computing Machinery, 2017, p. 81–88.