

Tapping into Student Behavior Insights tool to detect struggle in CS programming assignments

Gabriel Beal, zyBooks, A Wiley Brand

Dr. Chi Yan Daniel Leung, zyBooks, A Wiley Brand

Chi Yan (Daniel) Leung is the Senior Content Author (Labs) at zyBooks. He oversees the content creation and maintenance of labs across different titles at zyBooks. Before joining zyBooks, he was a lecturer at the School of Engineering at the University of California at Merced. He received his Ph.D. in Computer Vision from the University of California at Merced.

Joe Mazzone, zyBooks, A Wiley Brand

Joe is the learning tools and zyLabs product development lead at zyBooks. Joe is also an adjunct professor at the University of Rhode Island teaching software engineering. Prior to working at zyBooks, Joe was an award-winning career and technical educator, teaching computer and software engineering to high school students. He is President of the Computer Science Teachers Association (CSTA) of Rhode Island and is a member of the CS4RI (Computer Science for Rhode Island) board, where he serves as a voice for CS educators and the computing industry.

Chelsea L Gordon, zyBooks, A Wiley Brand

Chelsea Gordon received her PhD in Cognitive Science at University of California, Merced in 2019. Chelsea works as a research scientist for zyBooks, a Wiley company that creates and publishes interactive, web-native textbooks in STEM.

Dr. Yamuna Rajasekhar, zyBooks, A Wiley Brand

Yamuna Rajasekhar is Director of Content, Authoring, and Research at zyBooks, a Wiley Brand. She leads content development for the Computer Science and IT disciplines at zyBooks. She leads the authoring and pedagogy team at zyBooks, developing innovative learning solutions that drive measurable student success. She is also an author and contributor to various zyBooks titles. She was formerly an assistant professor of Electrical and Computer Engineering at Miami University. She received her M.S. and Ph.D. in Electrical and Computer Engineering from UNC Charlotte.

Tapping into Student Behavior Insights to detect struggle in CS programming assignments

Abstract

Over the past decade, academic integrity has been an important consideration for computer science courses. Although there is vast research on mitigating cheating, tapping into student behavior that leads to cheating provides the opportunity to intervene early and has a positive impact on student learning. To further complicate this, AI is transforming the learning landscape, raising more concerns about academic integrity. With free access to tools that can easily generate solutions for programming assignments, struggling students turning to AI has become more prevalent, and detecting cheating has become even more challenging. Traditional similarity detectors are no longer sufficient, as AI-generated content can bypass these systems.

To address this, we are introducing Student Behavior Insights, a new beta feature for programming assignments in an online interactive textbook that supports autograding. This feature helps instructors identify students who may need additional support focusing on key behavior metrics. The feature provides valuable behavior data on the number of submissions, time spent, program runs, and the amount of code pasted during program development. These metrics offer a deeper understanding of how students engage with their assignments, making it easier to identify irregular behavior and struggling students so instructors can provide targeted interventions.

This paper explores various approaches to effectively utilize the Student Behavior Insights feature, providing early use cases and recommendations. It is important to note that Student Behavior Insights is not a "plug-and-play" solution for detecting cheating but is intended for use at the instructor's discretion. Additionally, this feature can serve as a predictor of student struggles. We will present examples of how to use the feature to gain insights into: 1) a student who works earnestly, 2) a student who is utilizing an outside source for their work, 3) a student who is struggling and resorts to outside assistance to complete the work.

Introduction

Student cheating on programming homework assignments in introductory computer science courses is a long standing trend [1-4], a problem that widespread access to large language models has substantially exacerbated such as ChatGPT. A survey from 2023 found that 30% of students frequently used GenAI tools for completing assignments [5]. Many academics are expressing concern that this may largely undermine learning processes and decrease academic integrity [6].

Now that advanced LLMs can generate content that is relatively indistinguishable from human created content [7-11], cheating detection has become much more difficult. Research investigating LLMs used for completing CS programming assignments shows high performance resembling human completion [12-16]. Detection methods relying on similarity detection that once proved effective [17-20] are not enough to detect use of complex LLMs. While webcam and screen-recording detection methods have proven successful [21], these are quite invasive and are likely to be perceived by students as indicative of low trust and shift their focus away from genuine learning and more towards avoiding getting caught cheating [22,23].

Detection methods based on behavioral analysis such as response patterns and timing have proven effective [24-26]. In this paper, we introduce a tool that looks at students' behavioral patterns in programming homework assignments, flagging particular instances of suspicious behavior and instances of student struggle to enable early intervention.

Online interactive textbook and labs system

In this study, we collected data for an Introductory Programming in Java course, which comprises 73 students. The textbook being used in this course is an online interactive textbook in which topics are taught using minimal text content combined with animations and learning questions [27]. The textbook also includes integrated programming homework problems. In these problems, students enter snippets of code in a coding window or indicate the output of a given piece of code. Each submission of a problem is immediately graded automatically.

In addition to the textbook and homework features mentioned above, the textbook provides a programming labs system that supports the lab component of the course [28]. This labs system supports over 50 programming language configurations, a professional-grade Linux development environment, powerful instructor tools, and advanced autograding capabilities. Student Behavior Insights is a newly added feature to the labs system.

Besides allowing instructors to author their own lab assignments, the labs system comes with over 200 pre-built lab assignments across various topics and in different difficulty levels. The Student Behavior Insights data was collected from 44 pre-built lab assignments. Each assignment assesses a student's mastery of a concept and is designed such that a novice student can complete it in between 15 and 30 minutes.

Student Behavioral Insights Beta

Student behavior insights (SBI) is a new beta feature for instructors to use on students' programming assignment data. This feature is enabled in the labs system and to date has been used in 1,472 online interactive textbooks. The feature has provided a holistic overview of student programming metrics that help instructors identify student behaviors and students who may need additional support or warrant additional investigation.

The feature highlights the following metrics, identifying students who deviate significantly from the class average as outliers worth investigation:

- Submissions: Number of student submissions made to the autograder.
- Explore runs: Number of code runs as students develop their code.
- Time spent: Total time spent by the student in the lab. The interquartile range method is used to identify upper and lower time spent outliers.
- Pasted code: The percentage of pasted characters input from when the lab was first opened to now. This gives instructors an idea of how much code was pasted throughout the time the student worked on the lab. The qualifier to be an outlier is a ratio greater than 50%.
- Pasted code at “first highest” submission: Percentage of characters pasted into the lab at the earliest submission with the best score. Outliers have a paste ratio greater than 50%. This metric accounts for cases where students modify their code after achieving full points.
- Abnormal development: A beta feature designed to detect potentially unnatural code development behaviors, especially manual copying from external sources. It is a 0 (no abnormal behavior) to 1 (abnormal behavior) score calculated using:
 - Thinking pauses: Gaps in activity while working on the assignment.
 - Switching frequency: How often students alternate between new code development and modifying existing code.
 - Linear development rate: The extent of continuous linear coding without revisions.
- Max score: The max score achieved by the student in the lab.

Student behavior insights (beta)

[Feedback?](#)

Analyzed Files [Edit](#)

Flower.java Plant.java PlantArrayListExample.java

Class activity averages [i](#)

Student Insights are only viewable by instructors and TAs with view scores permission.

5	14	20 min	80 lines	31 lines
avg submissions	avg explore runs	avg time spent	avg code pasted	avg lines of code

Student activity outliers [i](#)

[Download outliers data](#)

Students that deviate significantly from the class average. Outlier stats will be highlighted with an orange marker. These metrics are designed to provide a holistic view of student behavior in the lab and should be used as a supplementary tool alongside the workspace history feature. Clicking on a row will open the workspace for that student.

Student	Submissions	Explore runs	Time spent	Pasted code	Pasted code at submission	Abnormal Development
	2	12	15 min	95%	96%	0.00
	7	1	6 min	100%	100%	0.57 i
	3	5	11 min	61%	100%	0.51 i
	6	1	18 min	88%	100%	0.49 i
	11	6	38 min	58%	55%	0.73 i
	11	25	58 min	71%	98%	0.85 i
	13	17	30 min	96%	89%	0.00
	8	17	26 min	73%	95%	0.64 i
	13	11	27 min	10%	0%	0.44 i
	3	0	9 min	100%	99%	0.35 i

[<](#) [1](#) [2](#) [3](#) [4](#) [5](#) [>](#)

Figure 1: Example of the Student Behavior Insights feature.

Examples of outlier students' behavioral data

In this section, we describe real examples of using Student Behavior Insights to detect students who appear to be (1) completing assignments as intended, (2) using an external source, such as ChatGPT to complete their assignments, and (3) struggling on the assignments and resorting to an external source. We also highlight patterns of student behavior that we observed.

Example student 1, who earnestly completed the programming assignments.

When an individual earnestly writes code, it is an iterative and dynamic process. Students progressively develop their code by frequently running and testing it to

ensure functionality. This involves running code multiple times before submitting their work to an autograding system throughout the development cycle, often fine-tuning their code after each run and submission. Unlike mere copying and pasting from external sources – where a student may submit to the autograder immediately after pasting a new solution – normal code writing requires a non-linear approach. Students routinely revisit and modify various parts of their codebase, reflecting on and revising previous work as they discover and address new challenges. This reflective approach is characteristic of normal development when students do not copy from external sources, such as a peer’s work or AI-generated code.

Student 1 was a student that we identified as working in earnest on each lab in the courses. Fibonacci sequence is a classic computer science problem used as an exercise in many classes for many years. It is a problem that has many solutions online and easy for AI to generate code for. The Fibonacci sequence lab is a great example to analyze when looking for students working in earnest versus using external aids like peers, online solutions, and generative AI.

First, we have Student 1’s Coding Trail, a textual visualization of the student working on the lab. They started work on Wednesday, November 6th and spent 37 minutes working on the lab from start to finish. Each dash is an explore run the student performed and numeric value is each time the student submitted their code to the autograder with the value representing what the autograder gave them out of a max of 10 points.

Coding Trail:

11/6 W-----6-4,6-----8,2,8-8,2-8----10 min:37

The coding trail shows Student 1 checked their code’s behavior frequently, running their code and submitting to the autograder for feedback. Furthermore, with the code playback history feature we can see that between each run the student made logical changes to their code before testing again. The playback history timeline also shows code runs and submissions but includes large deletions and insertions (pastes). The student only pasted code once, and that was to move a return statement to another part in a condition. That type of pasting activity is considered normal in earnest development of code.

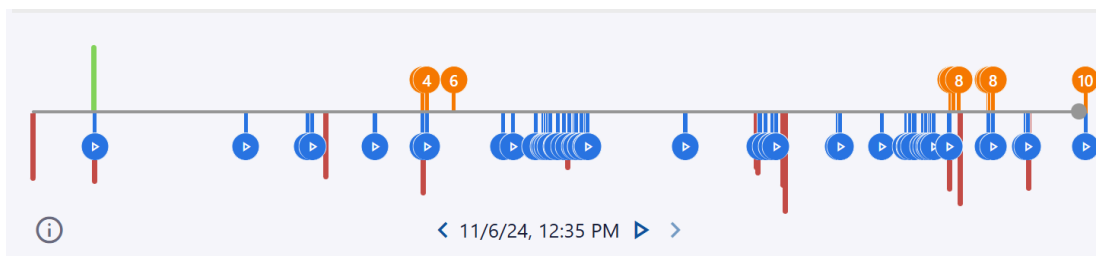


Figure 2: An example of workspace history for Student 1.

The submission table of Student 1's submission history shows the type of iterative development modifying code based on the feedback provided by the autograder you would expect from a student working on a programming activity in earnest.

★ Most recent highest scoring submission:

	Date	Time	Score	Test Results	
★	11/6/24	12:36 PM	10 / 10	2 2 2 2 2	▼
	11/6/24	12:33 PM	8 / 10	2 2 2 0 2	▼
	11/6/24	12:32 PM	2 / 10	0 2 0 0 0	▼
	11/6/24	12:32 PM	8 / 10	2 2 2 0 2	▼
	11/6/24	12:30 PM	8 / 10	2 2 2 0 2	▼
	11/6/24	12:29 PM	2 / 10	0 0 0 2 0	▼
	11/6/24	12:29 PM	8 / 10	2 2 2 0 2	▼
	11/6/24	12:03 PM	6 / 10	2 2 0 2 0	▼
	11/6/24	12:02 PM	4 / 10	2 0 0 2 0	▼
	11/6/24	12:01 PM	6 / 10	2 2 0 2 0	▼

Figure 3: Student 1's submission results.

Our Student Behavior Insights feature did flag Student 1 as an outlier. However, there are no warnings from this that indicate Student 1 cheated or used external sources. Instead, they were flagged for having higher than the standard deviation of submissions, explore runs, and time spent. They had little to no paste activity and zero abnormal development behavior was detected. It is our hypothesis that the student was flagged for submissions, explore runs, and time spent because most of the class used external sources to complete their code, reducing the number of runs and submissions they needed and of course time spent on the activity.

Student	Submissions	Explore runs	Time spent	Pasted code	Pasted code at submission	Abnormal Development
	10	58	37 min	12%	0%	0.00

Figure 4: Student flagged in the Student Behavior Insights for excess submissions, explore runs, and time spent.

Analysis of Student 1's metrics across 44 attempted labs revealed behavioral patterns characteristic of earnest academic engagement. The student demonstrated consistent effort with mean values of 14.77 minutes time spent per lab, 4.66 submissions, and 15.39 explore runs. Notably, their pasted code percentage remained low at 14.09%. These metrics align with expected patterns for authentic programming skill development, where students progressively build solutions through iterative testing and refinement.

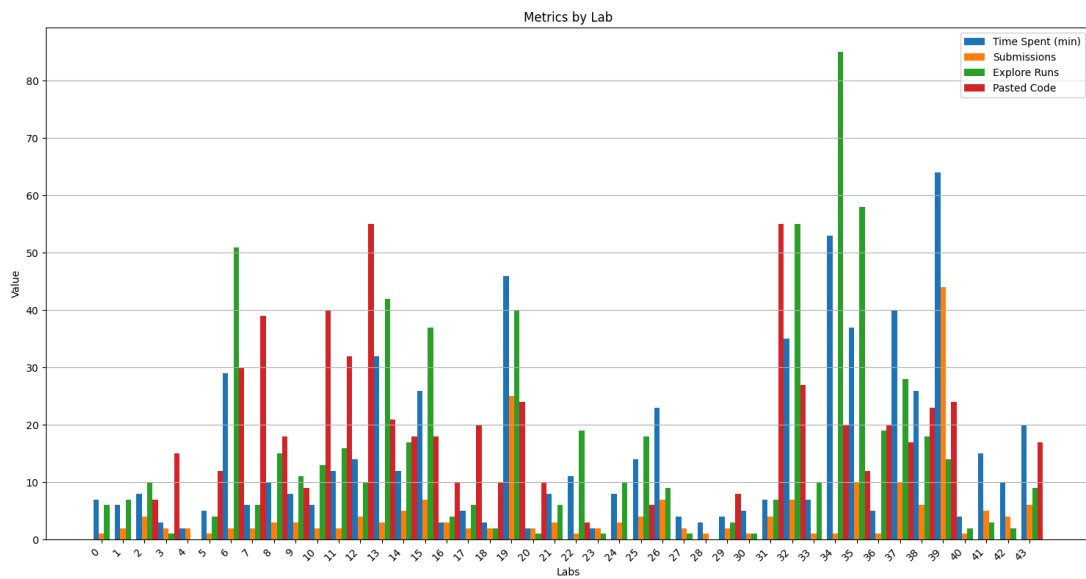


Figure 5. Bar chart showing Student 1's metrics across the 44 attempted labs

Example student 2, who likely acquired solutions from external sources.

Student 2 caught our attention because of the abnormalities observed in the student's behavior data. The time spent and completion data shows that Student 2 received full credit for all but two lab assignments. While the average time spent on a lab was 5.23 mins, 22 out of the 44 attempted labs were completed in one minute or less. The student was also flagged frequently as an outlier by Student Behavior Insights.

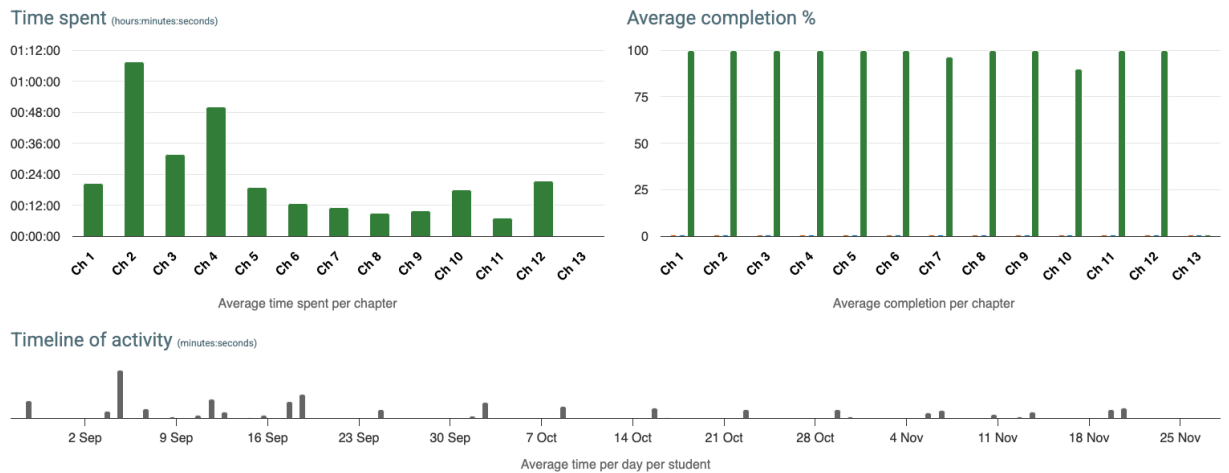


Figure 6. Time spent and completion data of Student 2.

Student 2 initially typed code manually and executed explore runs in the early assignments (Chapter 1 and early Chapter 2), but we observed the change of their behavior starting from Lab 2.20 in which they typed code manually but struggled in using Scanner to read from input. The student then deleted the entire code and pasted a complete code with comments (about creating a Scanner object, reading from input, printing the division 3 times, and closing the scanner). The comments

were then removed before the explore runs and submissions. The submitted code could be generated by AI because it printed the same division 3 times based on the instruction "outputs userNum divided by divNum three times." The output example shown in the instructions would have clarified what the statement meant. The student also attempted to fix the code manually but was unsuccessful due to the wrong interpretation of the problem. The student then deleted the entire code again and pasted a new complete program with comments that were more detailed ("Output userNum divided by divNum three times, updating userNum each time", "Add a space after each number except the last one", and "End with a new line"). Comments were removed before the explore run and submission. We also noticed that the submitted code used both loops and branches to format the output, but students were not expected to have learned those constructs until in later chapters. The similar observations were also made, where the student might not have learned the concepts used in the pasted solution. As a result, the student spent more time on correcting errors originated by the pasted solution.

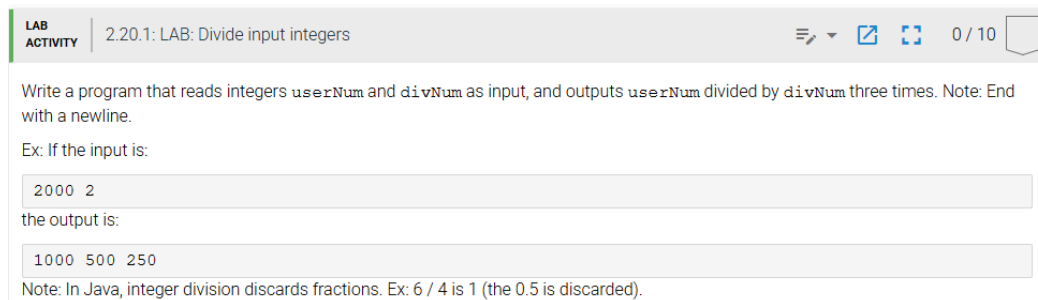


Figure 7. Instructions of Lab 2.20.

Since Lab 2.20, the pattern of deleting the entire content of the template file and pasting a new complete code with detailed comments persisted in the following labs. The time spent on each of the following labs also became shorter. Perhaps the student developed the solution in an external IDE, but the attempt of troubleshooting the errors in the pasted code suggested that an external IDE was not used to verify the correctness of the code before being pasted into our platform. Also, deleting the detailed comments from the pasted code is questionable if the student developed the solution individually. In some of the later assignments, the student did not delete the template code and pasted a new complete code for submissions. Instead, the student replaced each TODO comment in the template code with a block of pasted code.

```
Triangle.java x TriangleArea.java x
4 public static void main(String[] args) {
5     Scanner scnr = new Scanner(System.in);
6
7     Triangle triangle1 = new Triangle();
8     Triangle triangle2 = new Triangle();
9
10    double base1 = scnr.nextDouble();
11    double height1 = scnr.nextDouble();
12    triangle1.setBase(base1);
13    triangle1.setHeight(height1);
14
15    // TODO: Read and set base and height for triangle2 (use setBase() and setHeight())
16
17
18    System.out.println("Triangle with smaller area:");
19
20    // TODO: Determine smaller triangle (use getArea())
21    //      and output smaller triangle's info (use printInfo())
22
23 }
24 }
```

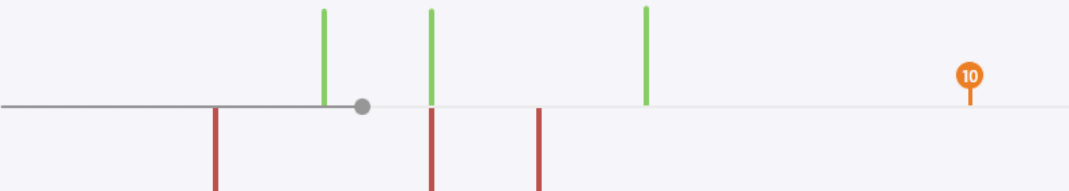


Figure 8. A TODO comment was replaced with a block of pasted code.

We also observed some mistakes made by the student while pasting the solution. In one lab, the student deleted the entire template code and pasted the lab instructions, which were deleted immediately. The intention of having the lab instructions saved in the computer's clipboard was questionable. Furthermore, the student pasted a complete code with a different class name in one assignment (PhotoFileRenamer vs the generic LabProgram used in most labs).

```
LabProgram.java
1 Write a program that reads a list of integers and outputs those integers in reverse. The input beg
2
3 |
```

Figure 9. Instructions of the lab were pasted in the program file.

```
LabProgram.java
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.Scanner;
4
5  public class PhotoFileRenamer {
6      public static void main(String[] args) {
7          Scanner scnr = new Scanner(System.in);
8          String fileName;
9
10         // Read the name of the file containing the list of photo file names
11         fileName = scnr.nextLine();
12         scnr.close();
13
14         try {
15             // Open the specified file
16             File file = new File(fileName);
17             Scanner fileScanner = new Scanner(file);
18
19             // Read each photo file name from the file
```

Figure 10. A program with a different class name was pasted.

Finally, we observed the student submit a series of labs in a short period of time. Unless the solutions were developed externally at a different time, it would be difficult to solve many programming problems within a short period on our platform.

Comparative analysis of Student 2's metrics revealed behavioral patterns consistent with external solution sourcing. The student averaged significantly lower engagement metrics compared to Student 1, with mean values of 5.23 minutes time spent (64.6% lower), 2.80 submissions (39.9% lower), and 3.86 explore runs (74.9% lower). Most notably, their pasted code percentage averaged 74.11% (60.02 percentage points higher than Student 1), suggesting heavy reliance on external code sources rather than original development. These metrics align with patterns typically observed in cases where students primarily use AI-generated or externally sourced solutions rather than engaging in authentic programming practice.

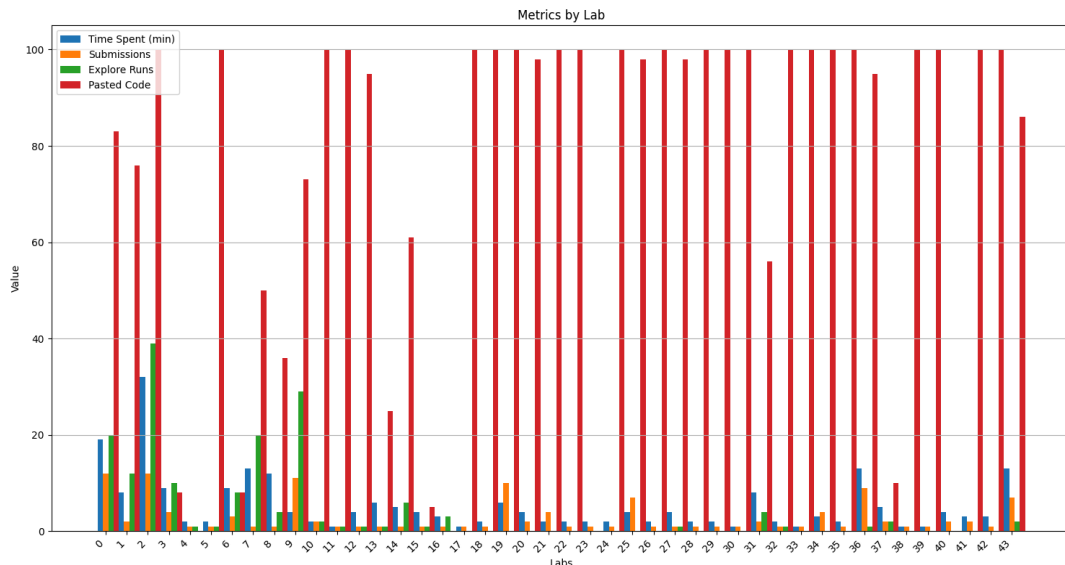


Figure 11. Bar chart showing Student 2's metrics across the 44 attempted labs. Note the extremely high pasted code on average coupled with minimal engagement metrics.

Example student 3: student working on solution before turning to AI

Student 3 provides us with an example of a student who is struggling with the assignment and eventually resorts to AI assistance in order to achieve full points for the lab. Student Behavior Insights flagged Student 3 for both paste metrics and this pattern of behavior is explored in the following section.

Student	Submissions	Explore runs	Time spent	Pasted code	Pasted code at submission	Abnormal Development
[REDACTED]	10	23	21 min	61%	99%	0.46 i

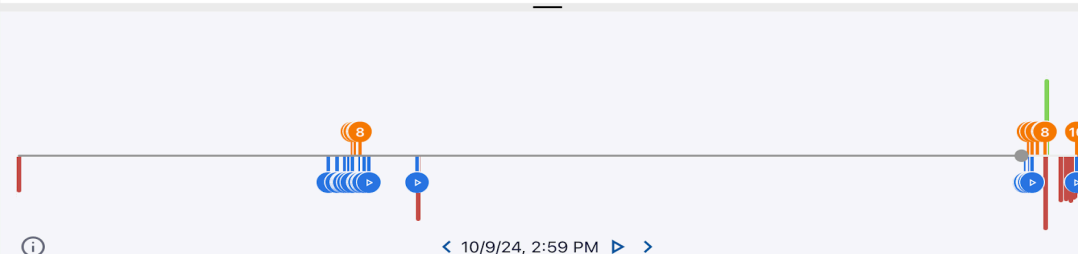
Figure 12: Student Behavior Insights metrics for Student 3, note the paste metrics.

The student's code showcases the workspace history tool where we can see the period of work that the student did on their own without pasting in outside content. The student worked for approximately twenty minutes reaching a maximum score of 8 in a single day's worth of work completed on 10/9/24.

Restore File Exit History

LabProgram.java

```
1  import java.util.Scanner;
2
3  public class LabProgram {
4      public static void main(String[] args) {
5          Scanner scnr = new Scanner(System.in);
6          int[] userValues = new int[9]; // Set of data specified by the user
7
8          int i = 0;
9          int input = scnr.nextInt();
10
11         while(input >= 0 && i < userValues.length){
12             userValues[i] = input;
13             i++;
14             input = scnr.nextInt();
15         }
16         if(i > userValues.length){
17             System.out.println("Too many numbers");
18         }else{
19             int middleIndex = (i - 1) / 2;
20             if(middleIndex >= 0 && middleIndex < userValues.length){
21                 System.out.println("Middle item: " + userValues[middleIndex]);
22             }else{
23                 System.out.println("Index out of bounds");
24             }
25         }
26     }
27 }
```



< 10/9/24, 2:59 PM >

Figure 13: State of workspace from Student 3's work with no pasted content.

The student returned to the lab two weeks later and resorted to pasting in a solution that included three indications of an AI source: (1) the class name was changed from LabProgram to MiddleInteger, (2) the code was filled with many verbose comments, which we described earlier as common in AI-generated solutions, and (3) the solution's approach to the problem differed completely from the previous one.

```

1  import java.util.Scanner;
2
3  public class MiddleInteger {
4
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7
8          // Initialize an array to hold the integers (maximum size of 9)
9          int[] numbers = new int[9];
10         int count = 0;
11
12         // Read integers until a negative number is encountered
13         while (true) {
14             int input = scanner.nextInt();
15
16             // Check for termination condition
17             if (input < 0) {
18                 break; // Exit loop on negative input
19             }
20
21             // Check if the array size exceeds the maximum allowed
22             if (count >= 9) {
23                 System.out.println("Too many numbers");
24                 return; // Exit if too many numbers

```

Figure 14: State of Student 3’s workspace immediately following the paste with a new class name and excess comments.

```

1  import java.util.Scanner;
2
3  public class LabProgram {
4
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7
8          int[] numbers = new int[9];
9          int count = 0;
10
11         while (true) {
12             int input = scanner.nextInt();
13
14             if (input < 0) {
15                 break;
16             }
17
18             if (count >= 9) {
19                 System.out.println("Too many numbers");
20                 return;
21             }
22
23             numbers[count] = input;
24             count++;

```

Figure 15: State of Student 3’s workspace at submission with original class name and comments removed.

Analysis of Student 3's metrics revealed an intermediate engagement pattern, with mean values of 8.18 minutes time spent, 3.86 submissions, and 3.43 explore runs. Their code paste metrics averaged 44.86% overall, with 48.00% at final submission. The concerning aspect as seen in the graph below is that this behavior of resorting to AI usage as seen in the earlier labs transitions to behavior we saw

with Student 2 in later assignments. This pattern suggests that insufficient mastery of fundamental concepts in earlier labs may have compromised the student's ability to tackle more advanced material independently, leading to increased reliance on external solutions.

This finding highlights a critical consideration for early intervention: identifying and addressing knowledge gaps during foundational coursework may prevent cascading difficulties as course material increases in complexity.

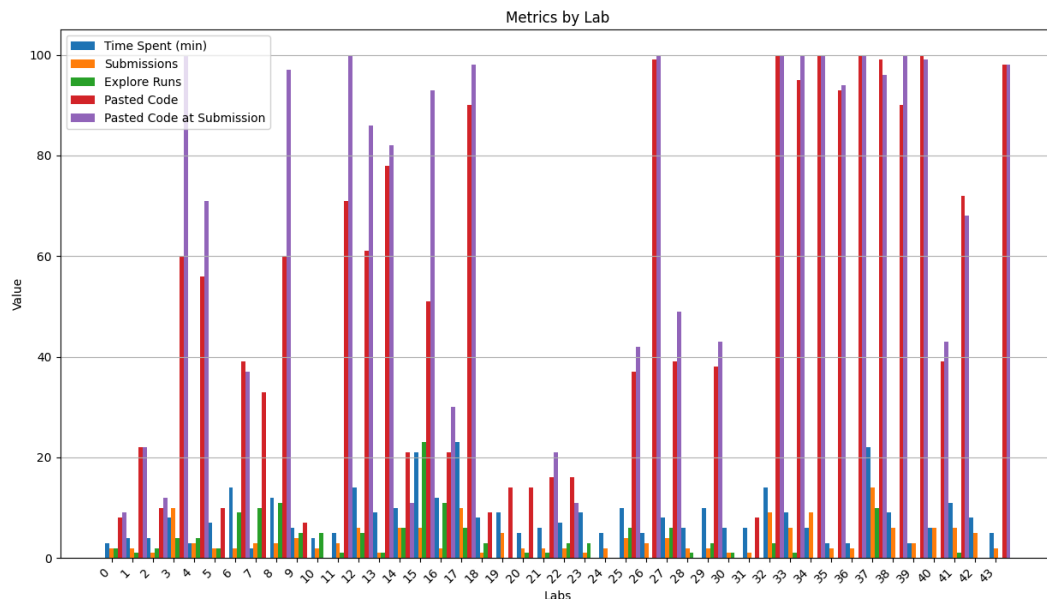


Figure 16. Bar chart showing Student 3's metrics across the 44 attempted labs. Note the trend of 100% paste metrics in the later labs.

Pattern of behavior: students working on solution before turning to AI

With the identification of Student 3 we wanted to see how many students in the class exhibited similar behavior. With Student Behavior Insights identifying these students was done with relative ease. From our sample set, we could identify 104 student cases across 38 of the 44 attempted labs. Of these 104 cases there were 39 unique students identified out of a class of 73 students, 53% of the class. The general pattern of behavior was individual work for 5-15 min wherein the student would run their code and likely submit as well for less than full points. This would be followed by one or several pastes of code coming from an outside source. The code was either a cleaned up and corrected version of their own work or a completely new solution. The primary indicator of AI sourced code was an abundance of verbose comments throughout the pasted code that were then usually deleted by the student.

From our sample set we can see a clear pattern to identify students who exhibit this behavior. Moderate submissions (4.14 on average), explore runs (12.41 on average), and time spent (16.12 min on average) indicate that the student attempts to solve the problem on their own for a portion of time. The two most important factors are overall pasted code (71.19% on average) and pasted content at time of submission (95.5% on average), which indicate that much of the work done came from the student; however, the work submitted at the end almost entirely came from a pasted outside source.

	Submissions	Explore Runs	Time Spent (min)	Overall Pasted Content %	Pasted Content % at time of Submission
count	104.000000	104.000000	104.000000	104.000000	104.000000
mean	4.144231	12.413462	16.115385	71.192308	95.509615
std	3.473669	13.189147	10.628143	12.107189	6.954771
min	1.000000	0.000000	5.000000	36.000000	71.000000
25%	2.000000	4.000000	9.000000	61.000000	94.750000
50%	3.000000	9.000000	14.000000	71.000000	99.500000
75%	6.000000	15.000000	20.250000	80.250000	100.000000
max	17.000000	75.000000	58.000000	93.000000	100.000000

Figure 17: Summary of Student Behavior Insight metrics from identified student cases.

We believe these students are ones who would benefit from a structured AI assistant that does not provide the complete answer but instead hints towards the solution as they have shown the desire to solve the problem on their own before resorting to outside assistance. These students would also benefit from early intervention to ensure fundamental skills are learned prior to more advanced subjects that rely on a strong foundation of learning. Without that foundation students tend to turn towards external sources as seen in figure 18.

Discussion

One of the key skills emphasized in a computer science education is the ability to persist through challenging problems, known as productive struggle. This essential aspect of problem-solving is now often bypassed because of the widespread availability of AI assistance. When students encounter obstacles in their development process, they can easily find answers online, a convenience that wasn't always guaranteed before AI became prevalent. Given the pressures of deadlines, other coursework, and grades, it's understandable that some students turn to AI for help. However, this reliance on AI may impede their ability to tackle more complex coursework or real-world problems where AI solutions aren't always readily available. This trend is evident in the increased amount of copied content as lab assignments become more difficult as shown in Figure 18.

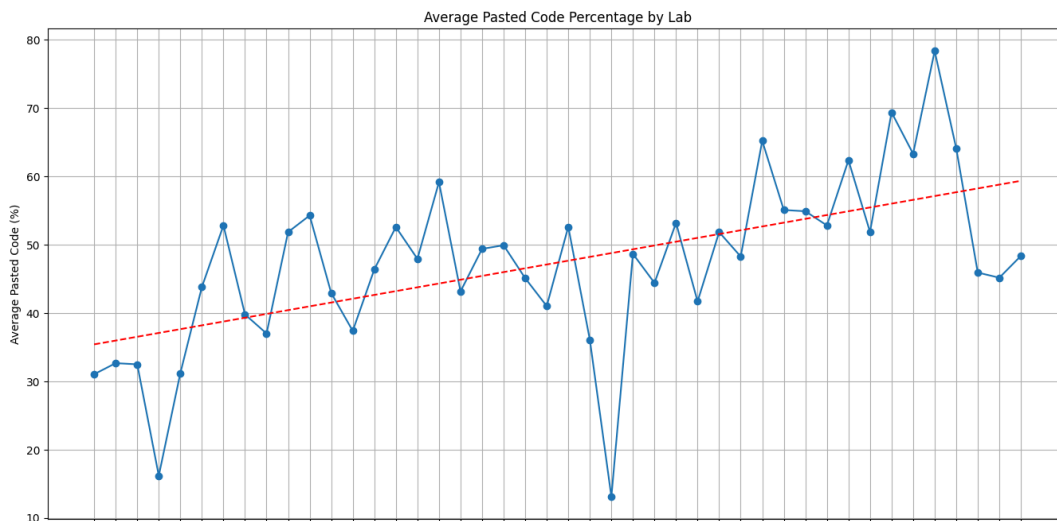


Figure 18. A chart showing the general increase in pasted code across the attempted labs in increasing difficulty.

When using plagiarism detection tools, communicating about the use and purpose of the tools is important for maintaining a positive learning environment [29], supplementing discussion about academic integrity in general [30]. Students are more likely to comply with AI use policies when instructors clearly indicate acceptable usage and declaration of AI, as well as consequences of non accepted use [8, 31-32].

Over-emphasis on punishment and overly harsh penalties should be avoided, as research shows that this leads to more students not declaring AI use and plagiarizing [32], while undermining a conducive, positive learning environment [21, 33]. On the other hand, overly lenient policies lead to normalized non accepted AI use without declaration [34]. Stone [35] suggests fostering shared responsibility for academic integrity with students, employing an approach where AI is showcased as a learning enhancement tool, with proper use made clear through examples and assignments.

With the information we are getting from Student Behavior Insights we want to begin to see if broader patterns across the class can be identified. Currently we only display the “Outliers” to the instructor but in the future we would like the same metrics to be available across the entire class. With this we can begin to try and identify trends that span across multiple labs.

Future Work

Moving forward we plan to expand our behavioral insights into other sections of the interactive textbook. This would include monitoring behavior within smaller activities that the students are expected to complete during their reading. The primary goal for our long term work is to identify and report to instructors whether classes are struggling with learning objectives before it reaches a point where the students are expected to use that knowledge to complete more difficult

assignments or tests. Instructors could use that knowledge to modify their teaching plans before it becomes too disruptive to go back and review foundational learning.

References

- [1] Marsan, C.D., 2010. Why computer science students cheat. PC World, April.
- [2] Bidgood, J. and Merrill, J.B., 2017. As computer coding classes swell, so does cheating. New York Times, 6.
- [3] Ragab, Y. One-sixth of computer science class caught cheating. Daily Illini, Mar 2018.
- [4] Medium.com blog post (anonymous). How UCLA Admins Could Stop The GitHub Cheating and Let Us Get Back To Learning CS. Dec 2017.
- [5] Welding, L. 2023. "Half of College Students Say Using AI on Schoolwork Is Cheating or Plagiarism." <https://www.bestcolleges.com/research/college-students-ai-tools-survey/>.
- [6] Abbas, M., F. A. Jam, and T. I. Khan. 2024. "Is It Harmful or Helpful? Examining the Causes and Consequences of Generative AI Usage among University Students." *International Journal of Educational Technology in Higher Education* 21 (1): 10. doi:10.1186/s41239-024-00444-7.
- [7] Hutson, J. (Ed.). (2024). *The Rise of AI in Academic Inquiry*. IGI Global.
- [8] Luo, J. (2024). A critical review of GenAI policies in higher education assessment: A call to reconsider the "originality" of students' work. *Assessment & Evaluation in Higher Education*, 1-14.
- [9] Cotton, D. R., Cotton, P. A., & Shipway, J. R. (2023). Chatting and cheating: Ensuring academic integrity in the era of ChatGPT. *Innovations in Education and Teaching International*, 1-12.
- [10] C. A. Gao, F. M. Howard et al., "Comparing scientific abstracts generated by chatgpt to original abstracts using an artificial intelligence output detector, plagiarism detector, and blinded human reviewers," 2022. [Online]. Available: <https://doi.org/10.1101/2022.12.23.521610>
- [11] J. Qadir, "Engineering education in the era of chatgpt: Promise and pitfalls of generative ai for education," 2022. <https://doi.org/10.36227/techrxiv.21789434.v1>
- [12] Denny, P., Kumar, V., and Giacaman, N. Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (New York, NY, USA, 2023), SIGCSE 2023, Association for Computing Machinery, p. 1136–1142.
- [13] Finnie-Ansley, J., Denny, P., Becker, B. A., Lexton-Reilly A., and Prather, J. The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference* (New York, NY, USA, 2022), ACE '22, Association for Computing Machinery, p. 10–19.

- [14] Reeves, B., Sarsa, S., Prather, J., Denny, P., Becker, B. A., Hellas, A., Kimmel, B., Powell, G., and Leinonen, J. Evaluating the performance of code generation models for solving parsons problems with small prompt variations. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (New York, NY, USA, 2023), ITiCSE 2023, Association for Computing Machinery, p. 299–305.
- [15] Savelka, J., Agarwal, A., Bogart, C., Song, Y., and Sakr, M. Can generative pre-trained transformers (gpt) pass assessments in higher education programming courses? In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (New York, NY, USA, 2023), ITiCSE 2023, Association for Computing Machinery, p. 117–123.
- [16] Wermelinger, M. Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (New York, NY, USA, 2023), SIGCSE 2023, Association for Computing Machinery, p. 172–178.
- [17] Albluwi I. Plagiarism in programming assessments: a systematic review. *ACM Transactions on Computing Education (TOCE)*. 2019 Dec 9;20(1):1-28.
- [18] Schleimer, S., Wilkerson, D.S. and Aiken, A., 2003, June. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (pp. 76-85).
- [19] Prechelt, L., Malpohl, G. and Philippsen, M., 2002. Finding plagiarisms among a set of programs with JPlag. *J. Univers. Comput. Sci.*, 8(11), p.1016.
- [20] Hage, J., Rademaker, P. and Van Vugt, N., 2010. A comparison of plagiarism detection tools. *Utrecht University. Utrecht, The Netherlands*, 28(1).
- [21] Aalborg, A., & Egeltoft, T. (2023). Exploring the use of plagiarism detection software in a higher education context: A critical review of the literature. *International Journal of Educational Integrity*, 20(1), 1-18.
- [22] Zepeda-Hernández, S., Villaseñor-Zepeda, C. A., & Márquez-Montiel, J. (2022). A systematic review of academic integrity in online learning environments. *International Journal of Educational Technology in Higher Education*, 19(1), 1-22.
- [23] Nguyen, F. T. (2023). Student perceptions of academic integrity tools: A survey study. *Journal of Educational Technology Development and Exchange (JETDE)*, 16(2), 181-198.
- [24] Y. Zhang, “Academic cheating as planned behavior: The effects of perceived behavioral control and individualism-collectivism orientations,” *Higher Educ.*, vol. 87, no. 3, pp. 567–590, Mar. 2024.
- [25] Tiong, L. C. O., Lee, Y., Lim, K. L., & Lee, H. J. (2024). Advancing Online Assessment Integrity: Integrated Misconduct Detection via Internet Protocol Analysis and Behavioural Classification. *IEEE Access*.
- [26] Balida, D. A., Navarro, J., Gonzaga, E. B., Gapoy-Landicho, W. C., & Collado, M. F. (2024). The Impact of the EduIntegrity Suite on Academic Integrity: A Qualitative Study. In *Proceedings of The 23rd European Conference on e-Learning. Academic Conferences International*.

- [27] C. L. Gordon, R. Lysecky and F. Vahid, "Less Is More: Students Skim Lengthy Online Textbooks," in IEEE Transactions on Education, vol. 66, no. 2, pp. 123-129, April 2023, doi: 10.1109/TE.2022.3199651.
- [28] Leung, C. Y. D., & Mazzone, J., & Kazakou, E., & Gordon, C., & Edgcomb, A. D., & Rajasekhar, Y. (2024, June), A Powerful Labs Environment for Computer Science Courses Paper presented at 2024 ASEE Annual Conference & Exposition, Portland, Oregon. 10.18260/1-2--46475
- [29] Nguyen, A., Ngo, H. N., Hong, Y., Dang, B., & Nguyen, B. P. T. (2023). Ethical principles for artificial intelligence in education. *Education and Information Technologies*, 28(4), 4221-4241.
- [30] Simon, Sheard, J., Morgan, M., Petersen, A., Settle, A., & Sinclair, J. (2018, January). Informing students about academic integrity in programming. In *Proceedings of the 20th Australasian Computing Education Conference* (pp. 113-122).
- [31] Carless, D. 2013. "Trust and Its Role in Facilitating Dialogic Feedback." In *Feedback in Higher and Professional Education: Understanding It and Doing It Well*, edited by David Boud and Elizabeth Molloy, 90–103. London: Routledge.
- [32] Tan, C., M. M. Alhammad, and M. Stelmaszak. 2024. "Teachers' Perceptions of Students' Use of Generative AI in Summative Assessments at Higher Education Institutions: An Exploratory Study." Paper presented at the UK Academy for Information Systems Conference Proceedings 2024. https://aisel.aisnet.org/ukais2024/19?utm_source=aisel.aisnet.org/ukais2024/19&utm_medium=PDF&utm_campaign=PDFCoverPages.
- [33] Callaghan, D. E., Graff, M. G., & Davies, J. (2013). Revealing all: misleading self-disclosure rates in laboratory-based online research. *Cyberpsychology, Behavior, and Social Networking*, 16(9), 690-694.
- [34] Simkin, M. G., & McLeod, A. (2010). Why do college students cheat?. *Journal of business ethics*, 94, 441-453.
- [35] Stone, A. (2023). Student perceptions of academic integrity: a qualitative study of understanding, consequences, and impact. *Journal of Academic Ethics*, 21(3), 357-375.