Examining Student Resolutions of Automated Critiques

Laura Albrant, Michigan Technological University

After completing a bachelor's degree in computer science, Laura Albrant decided to challenge how she viewed software development, by switching departments. Currently working towards a PhD in Applied Cognitive Science & Human Factors at Michigan Technological University, Laura pursues interests on both sides of the fence through education research.

Dr. Michelle E Jarvie-Eggart P.E., Michigan Technological University

Dr. Jarvie-Eggart is a registered professional engineer with over a decade of experience as an environmental engineer. She is an Assistant Professor of Engineering Fundamentals at Michigan Technological University. Her research interests include technology adoption, problem based and service learning, and sustainability.

Dr. Leo C. Ureel II, Michigan Technological University

Leo C. Ureel II is an Assistant Professor in Computer Science and in Cognitive and Learning Sciences at Michigan Technological University. He has worked extensively in the field of educational software development. His research interests include intelligent learning environments, computer science education, and Artificial Intelligence

Examining Student Resolutions of Automated Critiques

Abstract

This study examines the submission data of students in an introductory engineering courses who modify their code in response to automated critique and how this process evolves over multiple submissions. Our research is grounded in data collected from students in Engineering Fundamentals (EF) courses who used our automated feedback tool, automated feedback tool, WebTA. This tool provides instant feedback on code syntax, logic, and style, tailored for first-year college students. We analyze student interactions over four semesters, examining the effectiveness of EF students' abilities to resolve critiques. This comparative analysis highlights differences in problem-solving strategies, engagement levels, and learning approaches in engineering fundamentals students. The data gathered from four semesters of engineering fundamentals courses, includes detailed logs of each submission, such as submission times, errors identified, critique counts, and resolution times. By analyzing patterns across different submissions, we tracked how student learning evolved over time and varied across disciplines. This approach allowed us to identify effective strategies in automated feedback design that cater to the diverse needs of learners from different educational backgrounds.

1 Introduction

Providing *rich, timely feedback to the students when they are learning* is a significant challenge in large classrooms. This is especially true in first-year engineering classrooms where students are tasked with learning the syntax and understanding the semantics of a programming language, such as MATLAB. The feedback provided by the MATLAB environment is often opaque and geared towards experts, not beginning programmers. Most large introductory programming courses incorporate autograders into their teaching and learning practice, but here again the feedback provide is often not rich and may just provide binary (correct/incorrect) or number of points scored feedback.

The ability to provide timely and effective feedback is taxed by the onerous time commitment required and compounded with large course sizes. *Instructors cannot critique every student's code in large classroom settings. Furthermore, instructors are unavailable during late-night study sessions (or at any particular time of day an individual student needs that immediate feedback) to help students develop good coding practices.*

We have developed an automated *code critiquer* called WebTA that serves as our research platform as we investigate ways to help students learn to code [1, 2]. This study reviews WebTA's learnability through the students' behavior as logged by the system.

2 Background

2.1 Patterns & Antipatterns

2.1.1 Patterns

Design patterns represent fundamental principles of good software design, applicable across various programming styles. Patterns represent well-structured algorithms, modular function design, and consistent data handling. They promote readability, maintainability, and reusability, even within the constraints of script-based development [3].

Pattern: The Parameterized Function Pattern

For example, the *Parameterized Function* (Listing 1) pattern involves encapsulating a specific computation or algorithm within a function that accepts input parameters. This pattern promotes modularity, reusability, and clarity, allowing for easy modification and adaptation of the function to different scenarios.

Consider a scenario where we need to compute the numerical derivative of a function using a finite difference method.

Listing 1: The Parameterized Function Pattern

```
% Pattern: Parameterized Function
function derivative = computeDerivative(func, x, h)
  % Computes the numerical derivative of func at x
  % using a central difference method.
  % func: Function handle.
  % x: Point at which to compute the derivative.
  % h: Step size.
  derivative = (func(x + h) - func(x - h)) / (2 * h);
end

% Example usage:
f = @(x) sin(x);
x_val = pi / 4;
step_size = 0.001;
deriv_val = computeDerivative(f, x_val, step_size);
disp(["Derivative at x = " num2str(x_val) ": " num2str(deriv_val)]);

f2 = @(t) t.^2;
t_val = 2;
step_size2 = 0.00001;
deriv_val2 = computeDerivative(f2,t_val, step_size2);
disp(["Derivative at t = " num2str(t_val) ": " num2str(deriv_val2)]);
```

In Listing 1, the ‘computeDerivative’ function encapsulates the derivative computation, accepting the function handle, the point of evaluation, and the step size as parameters. This positive pattern promotes reusability, as the function can be applied to different functions and parameter values without modification. It also improves clarity, as the function's purpose and inputs are explicitly defined.

2.1.2 Antipatterns

Antipatterns are commonly observed problem solutions that result in negative consequences [4]. They often arise from a lack of structure, excessive code duplication, and reliance on implicit assumptions. These practices lead to fragile code that is hard to understand, modify, and debug. The absence of clear interfaces and effective data flow management worsens these issues, making it difficult to grasp the program's behavior.

Antipatterns are code structures that appear sound, and may indeed work well in limited contexts, but generally produce poor results [4]. Andrew Koenig illustrated them as, "An Antipattern is just like a pattern, except that instead of a solution it presents something that looks superficially like a solution, but isn't one" [5].

While we are concerned with Antipatterns in general, we are more concerned with *novice Antipatterns*; i.e., poorly conceived or erroneous code structures commonly created by novice programmers. Often, these novice Antipatterns represent bad solutions that would never be seen in expert code. For this reason, tools designed to assist expert programmers rarely provide good feedback on these kinds of mistakes.

Antipattern: The Magic Number Antipattern

For example, consider the antipattern where magic numbers are used directly within the code, without clear documentation or encapsulation (Listing 2).

Listing 2: The Magic Numbers Antipattern

```
% Antipattern: Magic Numbers
f_magic = @(x) sin(x);
x_magic = pi / 4;
h_magic = 0.001;
deriv_magic = (f_magic(x_magic + h_magic) - f_magic(x_magic - h_magic)) / (2 * h_magic);
disp(["Derivative at x = " num2str(x_magic) ": " num2str(deriv_magic)]);

f2_magic = @(t) t.^2;
t_magic = 2;
h2_magic = 0.00001;
deriv2_magic = (f2_magic(t_magic + h2_magic) - f2_magic(t_magic - h2_magic)) / (2 * h2_magic);
disp(["Derivative at t = " num2str(t_magic) ": " num2str(deriv2_magic)]);
```

In Listing 2, the constants '0.001' and '0.00001' are used directly within the computation, without clear documentation or encapsulation. Among the potential problems caused by this are:

- *Lack of Clarity:* The purpose of the magic numbers is unclear, making it difficult to understand the code.
- *Reduced Maintainability:* If the values of the magic numbers need to be changed, they must be manually updated in multiple locations, increasing the risk of errors.
- *Reduced Reusability:* The code is tightly coupled to the specific values of the magic numbers, making it difficult to reuse in different contexts.
- *Increased Error Proneness:* If a magic number is mistyped or incorrectly updated, it can lead to subtle errors that are difficult to detect.

The Magic Number antipattern leads to brittle, unmaintainable code that is prone to errors. It violates fundamental principles of software engineering, such as clarity and maintainability.

The Parameterized Function pattern and the Magic Number antipattern illustrate the impact of design choices on programs. By adhering to good patterns, developers can create modular, reusable, and clear code. Conversely, when novices implement antipatterns, such as the Magic Number antipattern, it leads to technical debt, reduced maintainability, and increased complexity.

In the context of introductory computing, cultivating an awareness of patterns and a critical eye for antipatterns are good habits for writing sustainable and reproducible code. Application of sound design principles can significantly improve the quality and lclarity of student programs.

2.2 WebTA Platform

WebTA is a web-based, interactive tool that facilitates learning through automatic critique of student source code. WebTA is comprised of several modules, data sources, components, and interaction modes (Figure 1). Students submit code via a website. Their code is stored in a database and passed in sequence to a compiler, a test stand, and a critiquer. The critiquer module combines static analysis with an Abstract Syntax Tree (AST) to identify patterns and antipatterns in student code. Critique results are returned to the student immediately and stored for instructor review.

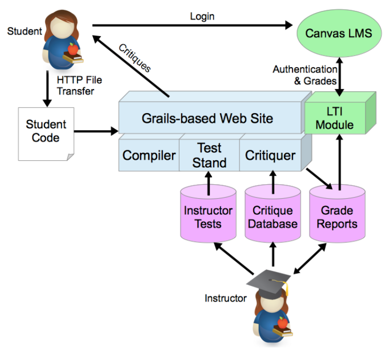


Figure 1: WebTA Architecture

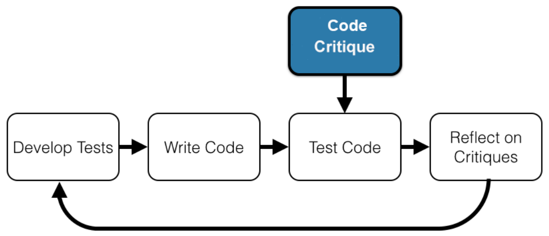
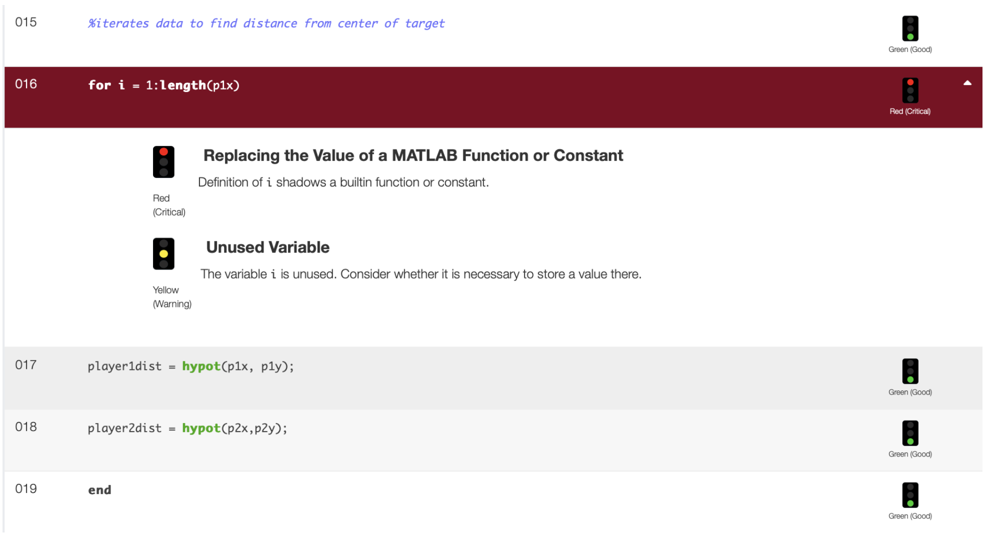


Figure 2: WebTA Integrated Development Cycle

Students are provided with immediate feedback in the form of line-by-line code critiques (Figure 3). Students receive the benefits of cognitive apprenticeship through the feedback they receive in the tool. This facilitates tight, productive cycles of inquiry, critique and learning (Figure 2).



```
%iterates data to find distance from center of target
for i = 1:length(p1x)
```

Replacing the Value of a MATLAB Function or Constant

Definition of i shadows a builtin function or constant.

Unused Variable

The variable i is unused. Consider whether it is necessary to store a value there.

```
player1dist = hypot(p1x, p1y);
player2dist = hypot(p2x,p2y);
end
```

Figure 3: WebTA code critiques

3 Research Framework

3.1 Cognitive Apprenticeship

The Cognitive Apprenticeship model [6, 7] is a constructivist approach to learning that focuses on teaching concepts and practices utilized by experts to solve problems in realistic environments. It has special relevance in the context of novice programming because it emphasizes making implicit processes explicit to the learner.

Cognitive Apprenticeship posits six key teaching methods:

1. **Modeling:** The expert demonstrates the target skill, making their thought processes explicit through verbalization and other means.
2. **Coaching:** The expert provides guidance and support as the learner attempts to perform the skill, offering feedback and suggestions.
3. **Scaffolding:** The expert provides temporary support structures that enable the learner to perform tasks that would otherwise be beyond their capabilities.
4. **Articulation:** The learner is encouraged to articulate their understanding and reasoning, making their thought processes explicit.
5. **Reflection:** The learner is encouraged to compare their performance with that of the expert, identifying areas for improvement.
6. **Exploration:** The learner is encouraged to independently explore new problems and apply their skills in novel contexts.

These methods promote the development of metacognitive skills, enabling learners to become self-directed and independent problem-solvers. Cognitive Apprenticeship emphasizes the importance of situated learning, where skills are learned in the context of authentic tasks and problems.

3.2 The Kolb Learning Cycle

David A. Kolb's experiential learning theory [8] frames learning as a cyclical process involving four distinct stages: concrete experience, reflective observation, abstract conceptualization, and active experimentation. Kolb's model emphasizes the importance of experience as the foundation of learning, suggesting that knowledge is constructed through the transformation of experience.

- **Concrete Experience:** This stage involves engaging in a new experience or situation. This could be a practical task, a problem-solving scenario, or a social interaction.
- **Reflective Observation:** After an experience, the learner reflects on their observations and considers different perspectives and interpretations.
- **Abstract Conceptualization:** Based on the reflections, the student forms abstract concepts and generalizations, developing theories and models to explain their observations.
- **Active Experimentation:** Students apply new concepts and theories to new situations, testing their validity and refining their understanding.

The Kolb Learning Cycle is a continuous, iterative process that closely aligns with the program development cycle (Figure 2). Students can enter the cycle at any stage. The cyclical nature of learning allows for continuous improvement and adaptation as learners refine their understanding through repeated cycles of experience and reflection.

4 Methods

Across four semesters, we collected the data on every submission students made on WebTA: 1) the date and time of submission, 2) details of every pattern found within the code submitted, 3) the overall status (i.e. highest severity of the patterns found), 4) the identifier for who submitted, 5) and the assignment the submission is for. This data was compiled to inspect, compare, and contrast students' behaviors while using WebTA across all four semesters. The analysis' results suggest that student behavior remained relatively similar.

4.1 Participants and Setting

The study included undergraduate students enrolled in engineering fundamentals courses at an R1 public university. Over four academic semesters, data were collected from 898 students who used WebTA as part of their regular coursework. (Spring 2023: 68 students, Fall 2023: 153 students, Spring 2024: 62 students, Fall 2024: 615 students) Fall 2024 saw the largest number of students using our software as we deployed to all sections of the course. The EF courses utilized MATLAB to introduce engineering problem-solving through programming.

4.2 Data Collection

WebTA automatically logged every student submission, capturing a range of data points for each:

- **Submission Metadata:** Date and time of submission, student identifier, and assignment details.

- **Code Analysis:** Details of every pattern and antipattern detected within the submitted code, providing insights into the types of errors students made and their frequency.
- **Performance Indicators:** Overall status of each submission, categorized by the highest severity of errors, allowing for the assessment of student progress and error resolution over time.
- **Feedback Interaction:** Data on how students responded to specific critiques, including the time taken to resolve each and the sequence of changes made.

5 Results

For the following analysis, the three intervention assignments for the Spring 2023 will be called A, B, and C. The three assignments for Fall 2023 will be D, E, and F. The three assignments for Spring 2024 will be G, H, and I. Finally, the three assignments from Fall 2024 are referred to as J, K, and L. Table 1 shows a general overview of the submissions made by students across the assignments/semesters.

Submissions By Assignment					
Assignment	Total	Min	Max	Median	Mean
Spring 2023					
A	21	0	7	0	0.32
B	282	0	12	4	4.34
C	172	0	10	2	2.65
Fall 2023					
D	262	0	10	1	1.74
E	107	0	6	0	0.71
F	441	0	17	2	2.92
Spring 2024					
G	138	0	8	1	2.30
H	93	0	9	1	1.55
I	91	0	9	1	1.52
Fall 2024					
J	1732	0	26	2	2.83
K	828	0	20	2	1.35
L	805	0	18	1	1.31

Table 1: A breakdown of submissions for each assignment.

5.1 Spring 2023

As table 1 shows, the Spring 2023 semester has an assignment (A) with a tenth to a fourth of the submissions compared to the other assignments. This is due to an unexpected time constraint during the class period, which led to only a handful of students submitting to WebTA. In tandem, the low submission count can cause the assignment to seem like an outlier in weird instances. For example, assignment A is the only assignments with a mean number of submissions per student be less than 0.5.

The number of patterns found in students' submissions over time, color-coded by assignment is graphed in Figure 4. The thick lines show the mean number of patterns found for a given submission number per assignment. Table 2 shows the results from a simple linear regression model on the thicker, mean lines from Figure 4. This table reveals with good confidence ($p < 0.007$) that the slope of assignments A, B, and C are slightly negative. It is also clear that assignment B had some outlier students who received a hefty number of found patterns and made that their last submission.

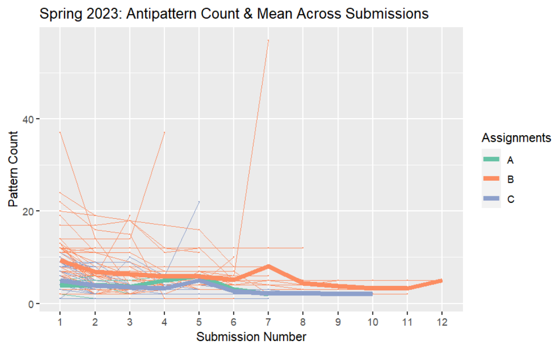


Figure 4: Spring 2023 Pattern Count and Mean Across Submissions

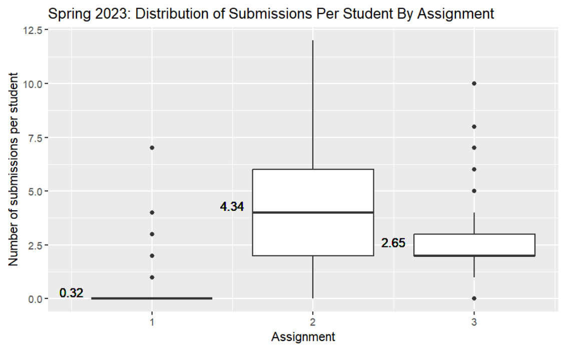


Figure 5: Spring 2023 Distribution of Submissions

In Figure 4, the data is from the Spring 2023 semester. Each thin line on this graph represents the number of patterns found across a student's submission(s) for a given assignment. The X-axis of submission number acts as time. The colors differentiate which assignment and the thick lines display the average across students at that submission/attempt number.

In Figure 5, the data is from the Spring 2023 semester. This boxplot shows the distribution of the number of attempts/submissions that students made per assignment.

Linear Regression of Mean Per Assignment		
Assignment	Estimated Coefficient	p-value
A	-0.5039	$p = 0.004$
B	-0.1665	$p = 0.006$
C	-0.5061	$p < 0.001$

Table 2: The estimated coefficients and p-values of linear regression on the mean lines from Figure 4.

The distribution of the number of submissions made by students is shown in Figure 5. The closest assignment to a normal distribution is assignment B. However, all three assignments are skewed towards zero submissions.

The ratio of a pattern's presence over students' submissions (time) is shown in Figure 6. For example, in the X number of students that reached 12 submissions (in all three assignments) around 50% of those submissions contain the "Correct Comment Block" pattern.

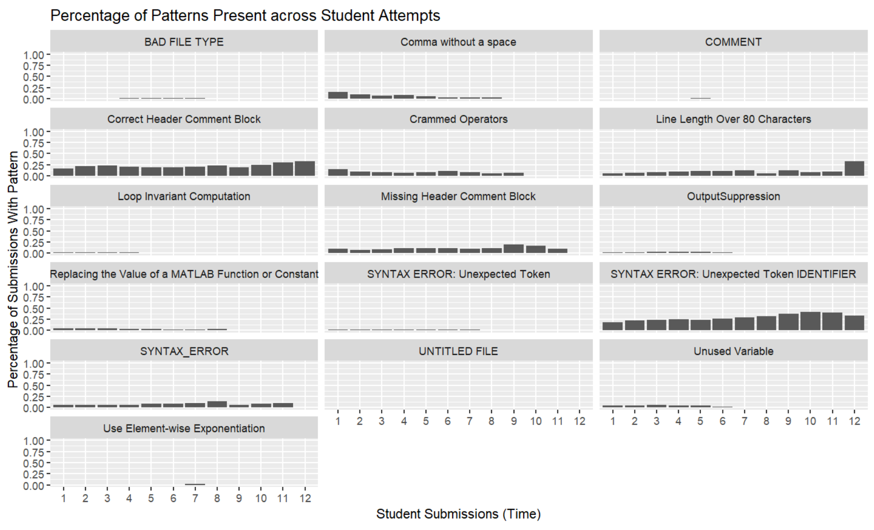


Figure 6: This series of bar charts shows the ratio of students whose code contained the given pattern at the given submission number. For example, in the Spring 2023 semester, no students that made 10 or more submissions had the ”Comma without a Space” antipattern by their 10th submission.

5.2 Fall 2023

The Fall semester(s) is peculiar compared to the Spring semesters as, at Michigan Technological University, there is a significantly higher number of students that join the school in the Fall than the Spring. This means that this semester had a significantly larger class (i.e. more likely to get submission data) than the Spring.

The number of patterns found in students' submissions over time, color-coded by assignment is graphed in Figure 7. The thick lines represent the mean number of patterns found for a given submission number per assignment. The average student has a negative slope or a rather flat slope. Table 3 shows the results from a simple linear regression model on the thicker, mean lines from Figure 7. This table reveals with good confidence ($p < 0.001$) that the slope of assignment E is negative. However, assignments D and F did not contain this same trend.

In Figure 7, from the Fall 2023, each thin line on this graph represents the number of patterns found across a student's submission(s) for a given assignment. The X-axis of submission number acts as time. The colors differentiate which assignment and the thick lines display the average across students at that submission/attempt number.

In Figure 8, from the Fall 2023, the boxplot shows the distribution of the number of attempts/submissions that students made per assignment. The closest assignment to a normal distribution is assignment B. However, all three assignments are skewed towards zero submissions.

Figure 9 observes ratio of a pattern's presence over students' submissions (time). For example, in the X number of students that reached 16 or 17 submissions (across all three assignments) 100% of those submissions contain the “Correct Comment Block” pattern.

In Figure 9, the series of bar charts shows the ratio of students whose code contained the given

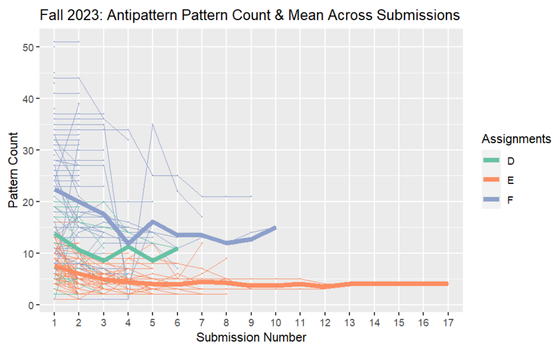


Figure 7: Fall 2023 Count & Mean Across Submissions

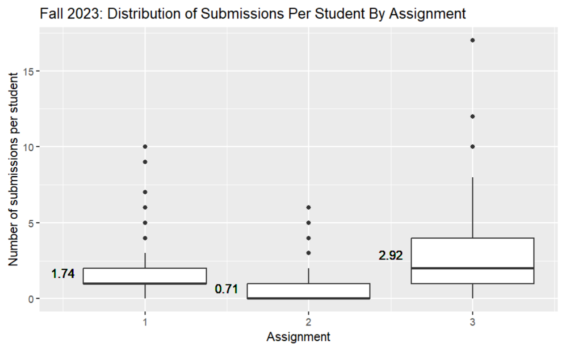


Figure 8: Fall 2023 Distribution of Submissions per Student

Linear Regression of Mean Per Assignment		
Assignment	Estimated Coefficient	p-value
D	-0.1866	p = 0.706
E	-0.6003	$p < 0.001$
F	0.44460	p = 0.105

Table 3: The estimated coefficients and p-values of linear regression on the mean lines from Figure 7.

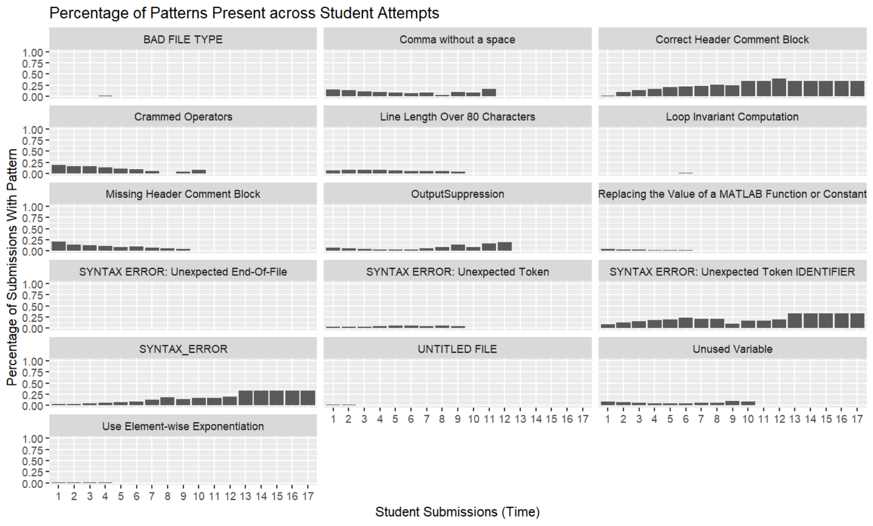


Figure 9: Percentage of Antipatterns across Student Attempts

pattern at the given submission number. For example, in the Fall 2023 semester, no students that made 11 or more submissions had the ”Comma without a Space” antipattern by their 11th submission.

5.3 Spring 2024

Spring 2024 is the semester in which students began using our newly re-designed WebTA. Due to similar class sizes, it is easier to make more direct comparisons and contrasts between this semester and Spring 2023

The number of patterns found in students' submissions over time, color-coded by assignment is graphed in Figure 10. Each thin line on this graph represents the number of patterns found across a student's submission(s) for a given assignment. The X-axis of submission number acts as time. The colors differentiate which assignment and the thick lines display the average across students at that submission/attempt number. The thick lines show the mean number of patterns found for a given submission number per assignment. Table 4 shows the results from a simple linear regression model on the thicker, mean lines from Figure 10. This table reveals with good confidence ($p <= 0.005$) that the slope of assignments G and I are negative.

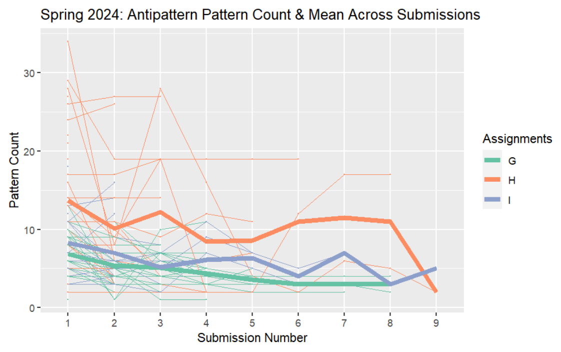


Figure 10: Spring 2024 Count and Mean Across Submissions

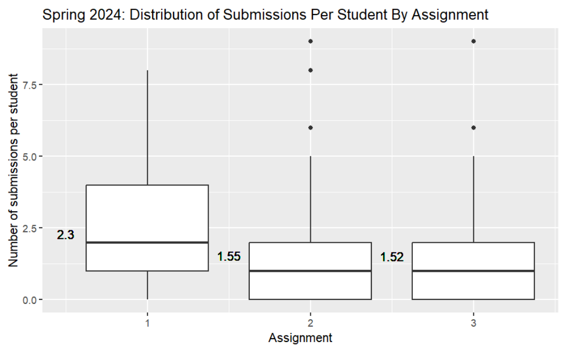


Figure 11: Spring 2024 Distribution of Submissions

The distribution of the number of attempts/submissions that students made per assignment (Figure 11). All three assignments have an average number of submissions per student that is > 1. This means that the average student submitted at least one time for all three assignments.

Linear Regression of Mean Per Assignment		
Assignment	Estimated Coefficient	p-value
G	-1.0159	$p < 0.001$
H	-0.0828	$p = 0.698$
I	-0.6574	$p = 0.005$

Table 4: The estimated coefficients and p-values of linear regression on the mean lines from Figure 10.

Figure 12 observes ratio of a pattern's presence over students' submissions (time). For example, in the X number of students that reached 9 submissions (across all three assignments) none of those submissions contain the "Correct Comment Block" pattern.

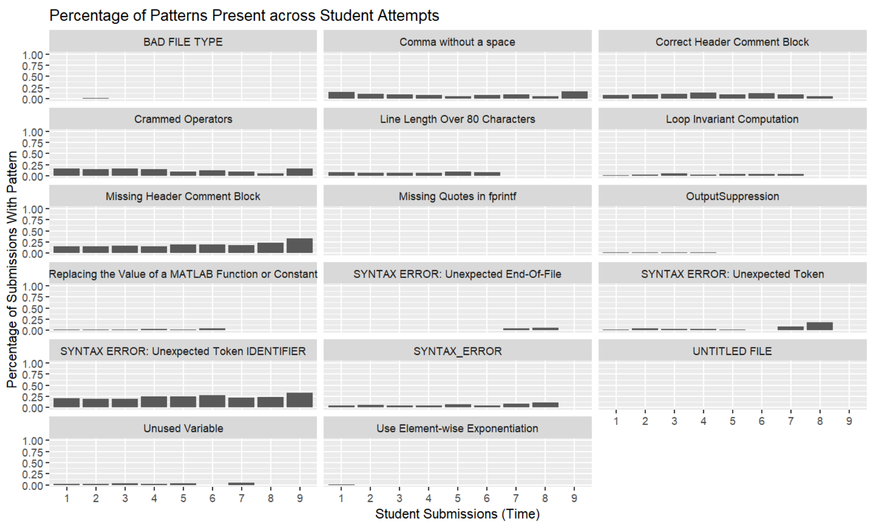
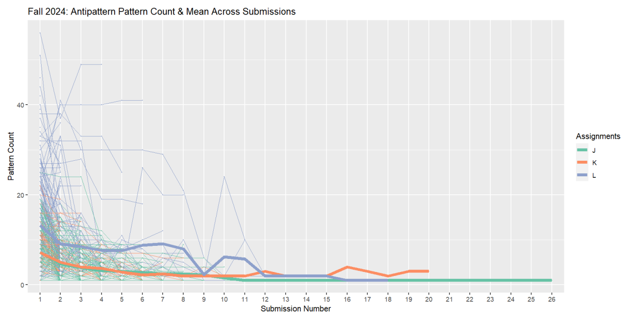


Figure 12: This series of bar charts shows the ratio of students whose code contained the given pattern at the given submission number. For example, in the Spring 2024 semester, no students that made 7 or more submissions had the ”Line Length Over 80 Characters” antipattern by their 7th submission.

5.4 Fall 2024

This Fall semester(s) contained the highest number of students than any of the previous semesters. It resulted in a much larger number of submissions for each assignment.

The number of patterns found in students’ submissions over time, color-coded by assignment is graphed in Figure 13. The thick lines represent the mean number of patterns found for a given submission number per assignment. Table 5 shows the results from a simple linear regression model on the thicker, mean lines from Figure 13. This table reveals with good confidence ($p <= 0.004$) that the slope of assignments J and K are slightly negative.



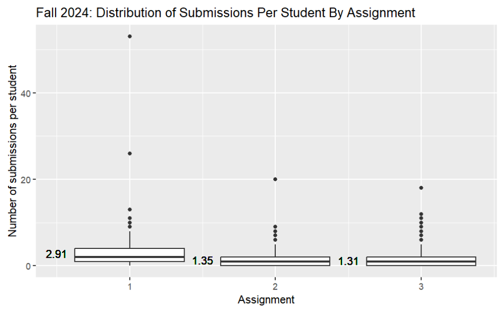


Figure 13: Fall 2024 Count and Mean Across Submissions

Figure 14: Fall 2024 Distribution of Submissions

In Figure 13, from the Fall 2024 semester, each thin line on this graph represents the number of patterns found across a student’s submission(s) for a given assignment. The X-axis of submission number acts as time. The colors differentiate which assignment and the thick lines display the average across students at that submission/attempt number.

Linear Regression of Mean Per Assignment		
Assignment	Estimated Coefficient	p-value
J	-0.1000	$p < 0.001$
K	-0.1460	$p = 0.004$
L	-0.0852	$p = 0.138$

Table 5: The estimated coefficients and p-values of linear regression on the mean lines from Figure 13.

Figure 15 observes ratio of a pattern's presence over students' submissions (time). For example, in the X number of students that reached 25 submissions (across all three assignments) all of those submissions contain the "Missing Header Comment Block" antipattern.

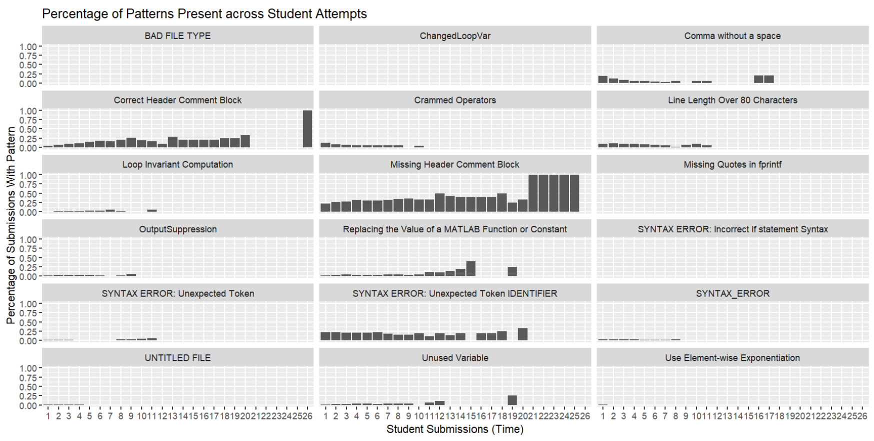


Figure 15: This series of bar charts shows the ratio of students whose code contained the given pattern at the given submission number. For example, in the Fall 2024 semester, all of the 26th submissions contained the "Correct Header Comment Block" good pattern.

6 Discussion/Conclusion

While it is difficult to make broad statements on education-related research, especially pilot data, this study still provides insight into students' resolutions while using WebTA. The key takeaways from this study include:

1. There is evidence of WebTA's learnability with this data as the average student is typically able to decrease the number of antipatterns found in their code. This is a trend found in multiple assignments, across multiple semestersm and across two different UI designs (Figs. 4, 7, 10; Tabs. 2,3,4).
2. There is a limit to how many tries an individual student is willing to do while submitting.
3. It can be inferred that too much feedback at once can cause a student to give up, likely from being overwhelmed and/or frustrated (Fig. 4).

The analysis of student behavior across four semesters is not conclusive, but rather extremely suggestive of WebTA's usefulness to enhancing students' learning while programming. As WebTA aims to help students through the processes of programming and improving their code. By only providing explanation of the antipatterns and hints to solve them, the above evidence of students learning to fix antipatterns is promising for the continued development of WebTA.

7 Future Work

Future plans include: (1) robust usability testing with both students & instructors, (2) validation & further prototyping of interactive help documentation, (3) continuation of rewriting/improving critique messages, (4) amending the lack of easy autonomy for professors to set up their assignments on their own in WebTA, (5) creation of a customizable experience with WebTA's pattern database, complete with a machine-learning recommendation system for which patterns to check for in an assignment, and (6) the creation of a dashboard for professors to employ as an aid to their pedagogy.

Acknowledgments

This work was funded by the National Science Foundation award #2142309. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

This paper is a modified version of Chapter 6 of the author's thesis [9]. We thank the National Science Foundation, the Institute of Computing and Cybersystems, and the Michigan Technological University College of Computing for their support in making this research possible.

References

[1] L. C. Ureel II, *Critiquing Antipatterns In Novice Code*. PhD thesis, Michigan Technological University, Houghton, MI, Aug 2020.

[2] L. C. Ureel II and C. Wallace, "Automated critique of early programming antipatterns," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 738–744, 2019.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software Addison-Wesley*. Addison-Wesley, 1995.

[4] W. H. Brown, R. C. Malveau, H. W. McCormick, and T. J. Mowbray, *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc., 1998.

[5] A. Koenig, "Patterns and antipatterns.," *Journal of Object-oriented Programming*, vol. 8, pp. 46–48, 1995.

[6] A. Collins, "Cognitive apprenticeship," in *Cambridge Handbook of the Learning Sciences* (R. Sawyer, ed.), pp. 47–60, Cambridge University Press, 2006.

[7] A. Collins, “Cognitive apprenticeship and instructional technology,” *Center for the Study of Reading Technical Report; no. 474*, 1989.

[8] D. A. Kolb, *Experiential learning: Experience as the source of learning and development*. FT press, 2014.

[9] L. E. Albrant, “Enhancing students’ user experience with a code critiquer,” Master’s thesis, Michigan Technological University, Houghton, MI, September 2024.