2024

ENHANCING STUDENTS' USER EXPERIENCE WITH A CODE CRITIQUER

Laura E. Albrant
Michigan Technological University, lealbran@mtu.edu

ENHANCING STUDENTS' USER EXPERIENCE WITH A CODE CRITIQUER

By

Laura E. Albrant

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Applied Cognitive Science and Human Factors

MICHIGAN TECHNOLOGICAL UNIVERSITY

2024

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Applied Cognitive Science and Human Factors.

Department of Psychology & Human Factors

Thesis Co-advisor:	*Dr. Leo C. Ureel II*
Thesis Co-advisor:	*Dr. Shane Mueller*
Committee Member:	*Dr. Laura E. Brown*
Committee Member:	*Dr. Michelle Jarvie-Eggart*
Department Chair:	*Dr. Kelly Steelman*

Dedication

To K. S. & the rest of my friends

Thank you for pushing me to this path. Love y'all!

Contents

List of Figures

List of Tables

Abstract

This thesis explores the role of human factors in the realm of code critiquers and students' experiences with them. Across three studies, the work utilized Design Thinking to improve the user experience of WebTA for introductory engineering students learning MATLAB. The first two studies gathered observational and interview data to empathize, define, and ideate a new user interface (UI). Said UI was prototyped and then tested with the third study. Overall, the surveys' data suggests that most students found the new design to be more appealing, useful, and purposeful; however, there is still plenty of room for improvement. Additionally, analysis of student behavior while using WebTA is very suggestive of enhancing students' learning. Future work will include more robust usability testing, validation & further prototyping of interactive help documentation, and the creation process of a dashboard for professors to employ as an aid to their pedagogy.

Chapter 1

Introduction

Human Factors/Ergonomics (HFE) and its related principles are the missing piece(s) to modern code critiquers. HFE is a melting pot of various disciplines (e.g. psychology, artificial intelligence, philosophy, engineering, etc.) and an autonomous science. HFE operates solely in and around the link between humans and technology. Code critiquers are software that analyze computer programs and output a report, typically on errors. Code Critiquers date back almost as far as coding itself. As soon as computer scientists found a way to command a computer, they commanded it to help them fix their work. My research is the marriage between the two.

The goal of this research is to upgrade the user experience of students' while using a code critiquer. This is done by revamping the user interface and implementing helpful

new features through the process of design thinking. This UI was then compared to the original design on its appeal, usefulness, and perceived fulfillment of purpose.

In this chapter, I will outline the background knowledge in which my research rests atop. Starting with more the simultaneously more broad and more pedantic topic of code critiquers. Then, I will detail how WebTA, the software that this research helps develop and hone, was borne and currently functions. In this thesis, I will discuss my contributions to improving students' user experiences while using WebTA, particularly in regards to its user interface. Chapter 2 outlines the typical user story, or use case, of WebTA. Chapter 3 dives into the theory and rationale behind the work. Chapter 4 examines the work done in the first/pilot study. Chapter 5 demonstrates the improvements made with the re-imagined UI. Chapter 6 deliberates on the larger, second study that evaluates the new design. Chapter 7 reviews the work overall, its implications, and its limitations.

1.1 Code Critiquers & Related Terms

Over time, the terminology relating to code critiquers has shifted and can quickly seem muddling. The major terms that relate to code critiquers and are important to this topic include: Automated Feedback System, debuggers, linters, autograders, and Intelligent Tutoring System(s). The best way to understand how all of these terms

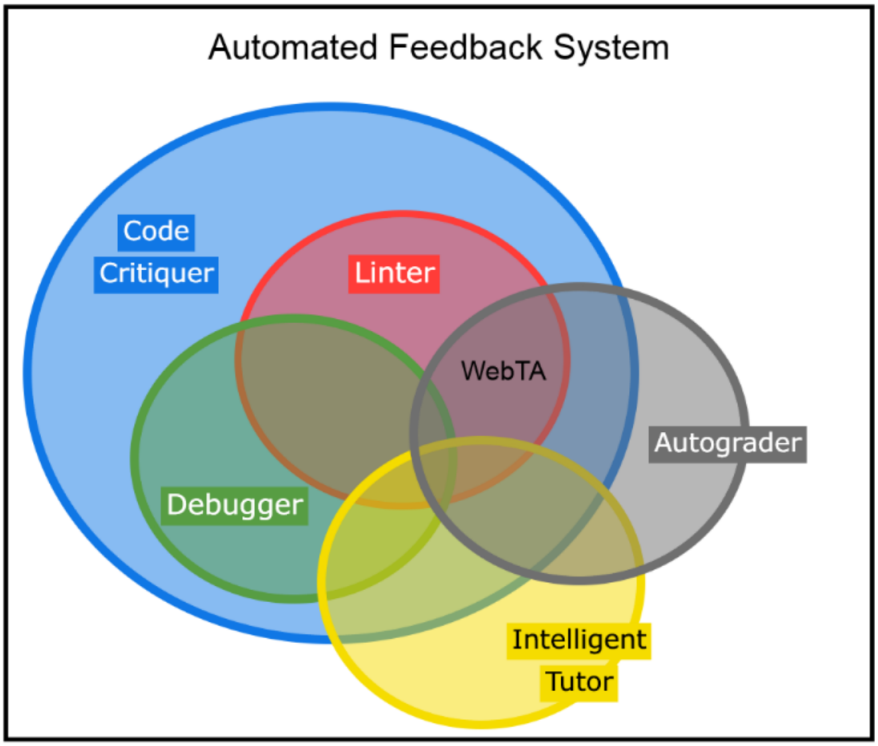

Figure 1.1: This venndiagram shows the relationship between terms relating to Automated Feedback Systems, code critiquers, and WebTA.

connect is shown in Figure 1.1. Automated Feedback System (AFS) is a superordinate term for every other term present. A code critiquer is a special type of AFS. Both debugger and linter are types of, and/or features of, a code critiquer. An autograder is a type of AFS as well as a potential feature of a code critiquer. An Intelligent Tutor System (ITS) is a type of AFS and a possible feature of a code critiquer. This makes WebTA, the focus of this paper, an Automated Feedback System, code critiquer, linter, and autograder.

1.1.1 Automated Feedback Systems (AFS)

The modern code critiquer stems from a category of systems that provide autonomous responses (e.g. assessment or commentary) to a learners' work, which are aptly named Automated Feedback Systems (AFS). AFS' exist and vary across numerous fields such as computer science & all of its subfields, mathematics, public-speaking, and even dance (Deeva et al., 2020). A code critiquer is a specialized AFS that only analyzes and assists with computer programs, or code.

In a comprehensive literature review of published AFS' from 2008 through 2019 (Deeva et al., 2020), researchers classified these systems based on a series of multi-faceted dimensions: 1) architecture, 2) feedback, 3) educational context, 4) evaluation, and 5) terminology. Architecture refers to an AFS' model of its domain, any incorporation of expert knowledge, the type of data collected from students (e.g. in-system behavior, open answer, etc.), the model for generating feedback, and how the system is implemented. Feedback refers to the adaptiveness, timing, level of user control, and purpose of the assistance the system provides a learner. Education context includes details on the domain, level of education, and setting (e.g. blended learning). Evaluation connects the method of testing (e.g. student surveys), group size, and statistical analysis techniques (e.g. ANOVA). Terminology simply alludes to the words and phrases the researchers observed authors using to describe the various systems.

For example, the most common phrase found was "Intelligent Tutoring System", in which 35% of the papers within the review had in their abstract/introduction (Deeva et al., 2020). Since code critiquers are a type of AFS, these classifications are able to be applied.

1.1.2 Debuggers & Linters

Some of the earliest code critiquers were debuggers, which are programs to help find errors within code. The 1980's debugger, LAURA, is one such archetype (Adam and Laurent, 1980). LAURA uses a graph model of a programming assignment requirements to help students find semantic errors. Many debuggers execute the code step-by-step with manually-set stops called breakpoints. It is a dynamic process. This process of debugging (i.e. removing "bugs" from code) is often used to understand discrepancies in logic and diagnose the exact point that a program may be incorrect.

Linters do not run the code. Instead, they analyze the source code as if it is text to find errors and stylistic mistakes. It is a static process. This type of program can aid in improving the readability of code on top of the detection of problems (Tómasdóttir et al., 2017).

1.1.3 Autograders

Autograders are widely utilized both in and out of computer science education. They are software that systematically provides grades to students' work based on some programmatic rubric. Autograding computer code often involves assigning point values on output correctness, time efficiency, and/or code style. The strictness of a rubric, as with any field, usually depends on the level of expertise (or lack thereof) expected from the students.

1.1.4 Intelligent Tutor Systems

Intelligent tutors, or Intelligent Tutor Systems (ITS), are dynamic, adaptive, and often tailored. ITS' contain a model representing the student's behavior while engaged in learning. This student/learner model is able to adjust to the student's needs in order to better support a student's learning journey (Woolf, 2010).

1.1.5 Code Critiquers

The term code critiquer is a slightly less broad designation compared to its parent, AFS. A code critiquer is a special type of AFS that only analyzes and comments on code, be that high-level, low-level, or even blocky code (Greifenstein et al., 2021, Von Wangenheim et al., 2018, Wang et al., 2021) (i.e. code constructed with GUI elements, or blocks). As such, code critiquer functionality varies greatly. Some (auto)grade, some debug, some lint, and others incorporate intelligent tutoring. Each encompasses its own distinct characteristics. Caesar at Massachusetts Institute of Technology has a heavy social aspect that strongly encourages students to peer review code (Tang, 2011). LAURA (Adam and Laurent, 1980), the debugger, is also a code critiquer. Researchers at Queensland University of Technology proposed a code analysis framework with a special "fill in the gap" type of programming assignment (Truong et al., 2005). The subject of this paper, WebTA, is equally distinct.

1.2 WebTA

1.2.1 Progress Thus Far

Borne in 2020, WebTA originated as a Java code critiquer with linting and auto-grading capabilities. It was developed with the purpose of assisting students within an introduction to Java and programming course outside of the classroom (Ureel II, 2020). To this day, WebTA continues to grow more detailed and extensive. Foundational steps have been made towards WebTA becoming a grouped system of code critiquers under one user interface (Ureel II, 2021). The beginnings of a Python critiquer are underway (Albrant et al., 2023c). Most prevalent, a MATLAB critiquer has been introduced to an introductory engineering course and thoroughly examined (Albrant et al., 2023b, Benjamin et al., 2024, Ureel II et al., 2022).

1.2.2 How WebTA Functions (For MATLAB)

1.2.2.1 Antipatterns, Good Patterns, & a Traffic Light

WebTA contains a database filled with antipatterns and good patterns. Antipatterns, in this context, are common mistakes found in novice programmer's code (Ureel II, 2020). These mistakes vary in severity of effect, importance, and programming language. For instance, stylistic antipatterns, like Crammed/Crowded Operators (i.e. missing spaces around operators like '=' and '+'), are labeled as "Warning" by WebTA. It does not affect how the code runs, but instead affects the code's readability. It also exists for Java, MATLAB, Python, and arguably every other high-level text-based language. Additionally, a semantic antipattern like Replacing Pi (i.e. using an unnecessary extra variable to contain the value of the number pi) can poorly affect both the efficiency and memory within a program. Thus, it receives the label of "Critical" by WebTA. This antipattern is most noticed in MATLAB but could be present in Python and Java as well.

The counterpart of an antipattern is called a good pattern. This is coding behavior that should be reinforced for novice programmers Albrant et al. (2023a). A humorously and anecdotally rare example of a good pattern is called Correct Comment Block, where a student includes and properly formats their header comment block(s)

at the top of their source file with the required details.

To annotate the labels, WebTA uses a traffic light system of green, yellow, and red, paired with the labels associated with its known patterns. Good patterns are green. Warning antipatterns are yellow. Critical antipatterns are red.

During the course of this research, MATLAB WebTA had limited functionality, at least in comparison to its predecessor. The Java WebTA can detect a wide variety of patterns: style, syntax, logic, semantic, run-time & compilation errors, inefficiency, etc.. The MATLAB WebTA, in its infancy state and due to the closed nature of the language, could only detect syntax and style patterns using various forms of static analysis.

1.2.2.2 Syntax Checking

In order to check MATLAB syntax, WebTA utilizes McGill's MATLAB Parser, McLab (Casey et al., 2010). This allows the critiquer to detect syntax errors like Unexpected Token. This error is has a large list of causes like mistyping, undeclared variables, etc.. An example of this is in Figure 1.2, which demonstrates what a student would see when this error is found in their code.

Figure 1.2: WebTA's feedback for an anecdotally common syntax error called Unexpected Token.

1.2.2.3 Regex Based Static Analysis

Regular expressions, or regex, are a common tool for text analysis in order to search and adjust. With the use of specific characters, regex can communicate complex text-based patterns. One pattern that WebTA uses a regular expression to find is Correct Comment Block. Students are required to put their program name, program description, their own name, class section, and team number.

```
% Program Name: myprogram.m
% Program Description: What my program does and how and why.
% Name: Jane Doe
% Section: L00
% Team: 03
```

The regular expression used to look for and check for this header comment block is:

```
(?m)\A\s*%\s+Program Name:\s*(.+)\s*
%\s+Program Description:\s*(% .*\s+)+\s*
%\s+Name:\s*(([^\s]+\s)+)\([^)]*\)\s*
%\s+Section:\s*(\w+)\s*
%\s+Team:\s*(\w+)\s*
```

Even split up for a line-to-line comparison, that regular expression is dreadful to read. Regex is inherently difficult for humans, even experts, to read. However, WebTA happens to excel with it.

1.2.2.4 AST Based Static Analysis

Using a tree-like data structure, an Abstract Syntax Tree (AST) is a method of representing code and its structure. WebTA utilizes an AST while linting as part of its search for antipatterns and good patterns. The use of an AST allows us to detect patterns that regular static analysis would be unable to find. The following code exerts an excellent example. The code below will likely have a logic error due to the floating-point number comparison.

```
a = 12.42
b = 16.000008
while (a < b)
    foo()
```

Corrected code would look something like this.

```
a = 12.42
b = 16.000008
```

```
epsilon = .05

while (abs(a - b) < epsilon)

    foo()
```

1.2.3 Linter

Linting is the process of scanning code to find errors, style crimes, and other text-based problems. The term itself is derived from the very first linter, called Lint (Johnson, 1977). WebTA's linting finds patterns like the antipattern Crammed Operators. An example of this in MATLAB may look like:

```
    for k=2:length(t):

        rmean=[rmean(t(k)+t(k-2))/2]

    end
```

A fixed version of the same code might look like:

```
    for k = 2:length(t):

        rmean = [rmean(t(k) + t(k-2)) / 2]

    end
```

Chapter 2

Extended Example

2.1 User Work Flow

The expected, or typical, user work flow of students is as follows:

1. Student logs in.
2. Student selects course.
3. Student selects assignment
4. Student either selects previous critique to view or submits more code file(s) to receive a new critique.

5. Student reviews feedback.

6. Student changes code (outside of WebTA).

7. Student resubmits to WebTA.

8. Repeat until student is content with results.

Suppose for step four, the student submits the following MATLAB code as a ".m" file:

```
fprintf(Hello World!)
```

This code is an example the antipattern "Missing Quotations (fprintf)". In MATLAB, unless a variable is involved, the text within an fprintf call must also have single or double quotations around them.

The student would receive the following feedback:

> Missing quotes around the format string.
>
> See https://www.mathworks.com/help/matlab/ref/fprintf.html.
>
> What are some good ways to remember to put quotes around the fprintf format string?

The student should then fix and resubmit the following code:

```
fprintf("Hello World!")
```

The expectation is that students will be content with their code upon receiving a green light from WebTA and correct output based on their assignment's rubric. Figure 2.1 is a more visual and detailed way to view this process. From the Start node to the Assignment Details node, the flow is straight forward and equates to the first four steps highlighted previously. The more complex part of the flow occurs after a student selects or submits a critique. From there, students can review the details that WebTA provides. The student can review the submission details (e.g. date, time, etc.), use the critique table and feedback messages to fix antipatterns in their code, and/or download the source files of that submission. Lastly, they can submit again to repeat the reviewing process until they are satisfied with the results.

2.2 WebTA's Original Design

Figures 2.2 through 2.5 provide exemplar screenshots of WebTA's original design with a focus on specific features. Figure 2.2 displays the very first page students will see after successfully logging in: a grid-like list of the courses available for them to see. Figure 2.3 shows an example of a course's main page, which contains a list of the existing assignments and their due dates. With Figure 2.4, a student sees the assignment title, description, a file selector, and a submit/critique button. Figure

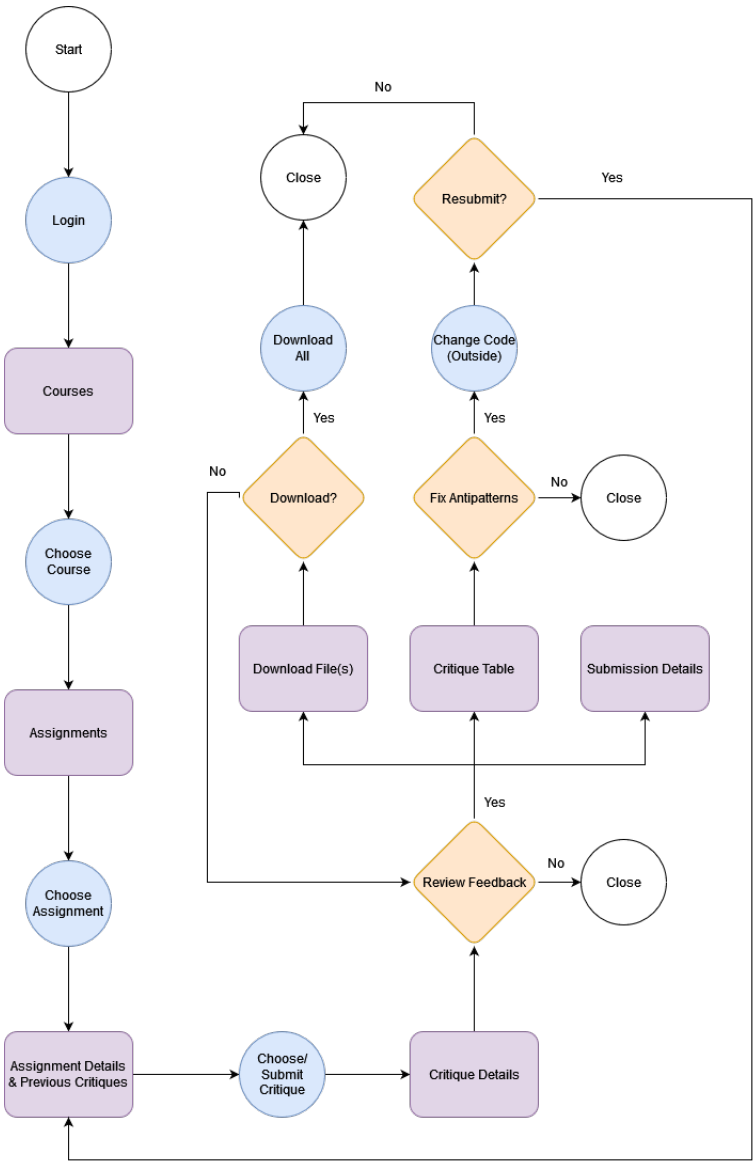

Figure 2.1: This figure displays a typical user work flow of students using WebTA.

2.5 showcases the same page as the previous screen shot, but now with the ability to review previous critiques that WebTA has stored.

Figure 2.6 displays the overall layout of what is called a "critique page". Every critique page has four primary functions: show the submission's details (Figure 2.7),

Figure 2.2: This figure is a screenshot of the first page students see after logging in. It lists each course they have access to.

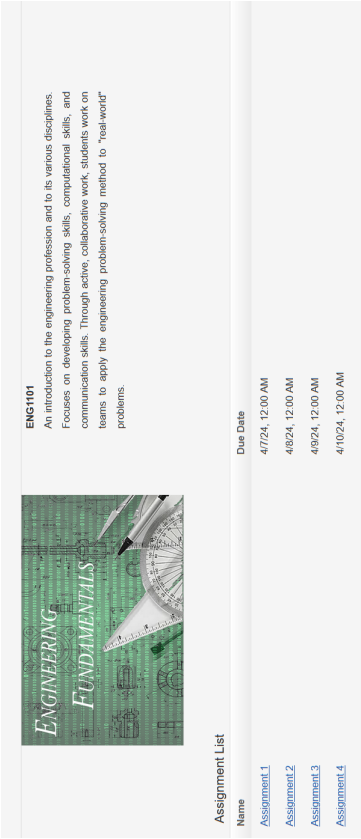

Figure 2.3: After selecting a course, students will see this web-page.

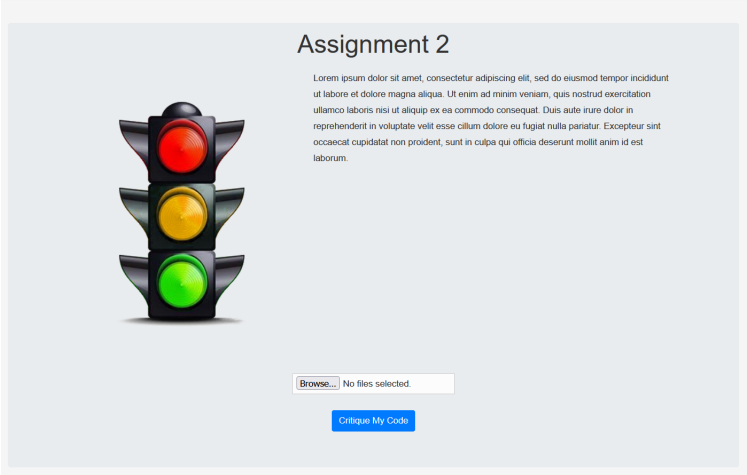

Figure 2.4: After selecting an assignment, the student will be met with a web-page like this.

Figure 2.5: This is what the same web-page as Figure 2.4 but after a student has submitted to WebTA at least once.

download the source files (Figure 2.7), and provide both a general summary (Figure 2.8) as well as a line-by-line breakdown (Figure 2.9) of what WebTA found in the code. Figure 2.10 presents and example of the line-by-line breakdown can seem quite overwhelming.

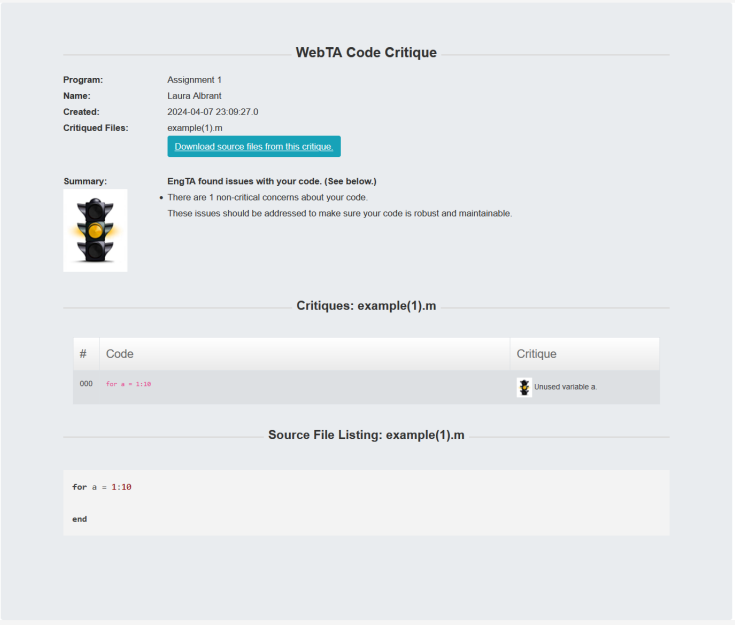

Figure 2.6: After submitting code, or selecting a previous critique from the list, the students will see this web-page.

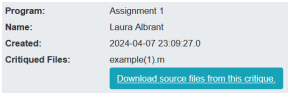

Figure 2.7: This is a closer look at the feature of the critique page which lists details about the submission as well as the button to allow students to download all of the files within the critique.

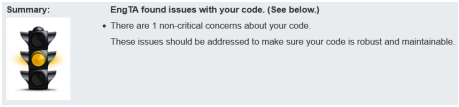

Figure 2.8: This is a closer look at the feature which provides a general summary on what WebTA found.

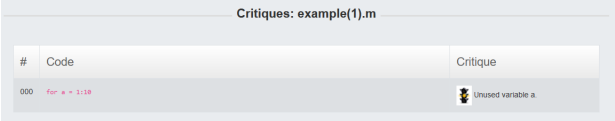

Figure 2.9: This is a closer look at the feature which displays antipatterns and good patterns found on a line by line basis.

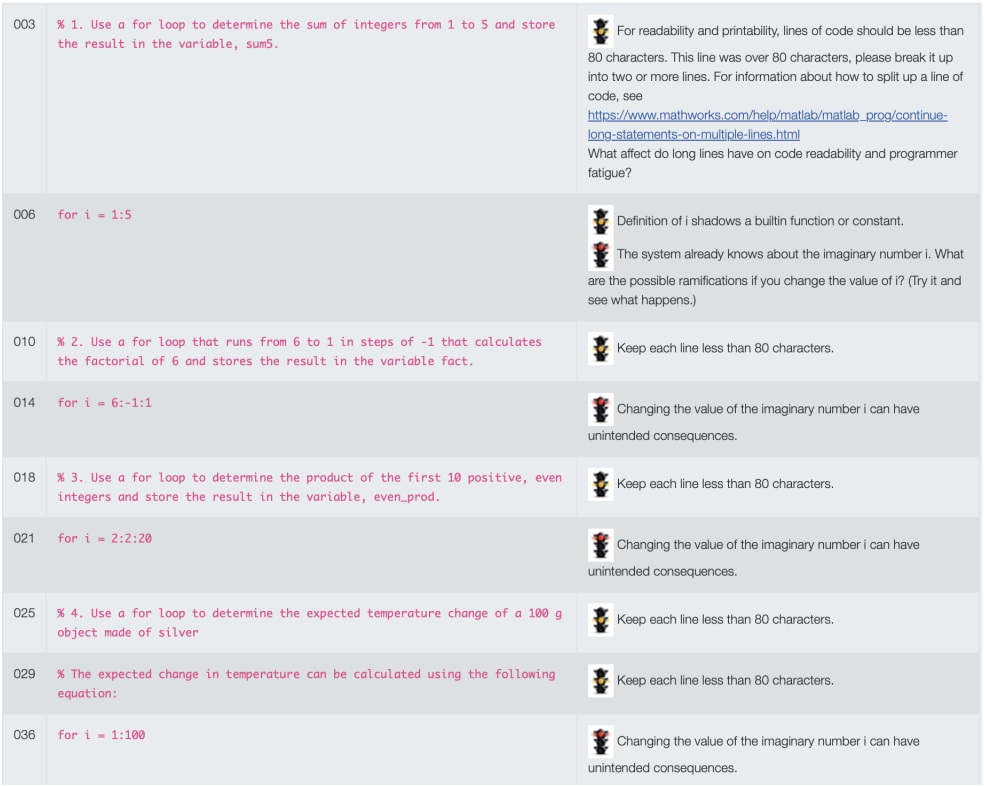

Figure 2.10: This is an additional example of Figure 2.9 to demonstrate how quickly the critique table can become overwhelming to look at.

Chapter 3

Theory & Rationale

3.1 Error Messages & User Interfaces: The art of showing a student ways to improve

There is a clear art to teaching. Every teacher/instructor does their own interpretation of what some believe is best for their students. Code critiquers are, and should be, no different. This is why I take how WebTA displays and describes the students' patterns with great concern and care.

Due to the complexity and dynamic nature of WebTA, a blend of three human factors, or ergonomic, viewpoints were/are required to properly analyze its user experience:

corrective, preventive, and prospective ergonomics. Corrective ergonomics focuses on "the problem to correct" (Robert and Brangier, 2009). In terms of WebTA, the 'problem' is the presence of antipatterns in students' programs. Preventive ergonomics focuses on the system's design, usually about design choices that minimize the chance of human error by accounting for human ability, or lack thereof (Robert and Brangier, 2009). This flavor of ergonomics would try to pinpoint and resolve WebTA's flaws in its user interface. Finally, prospective ergonomics is defined as "[an attempt] to anticipate human needs and activities to create new artifacts that will be useful and provide positive user experience" (Robert and Brangier, 2009). This is a framework to analyze a learning environment, training process, in our case, a learning system.

The first problem space I noticed, either with common sense or insight from the pilot study (Albrant et al., 2023b), was the antipattern feedback WebTA was giving. Two examples of the original design's feedback can be found in Figure 2.9 and Figure 2.10. These messages can be described as "functional but lacking". They deliver a message that contains a hint to a solution and a link to why this kind of pattern is an antipattern. However, they don't inform the student *why it should matter to them.*

While attempting to resolve this, I quickly realized that WebTA's antipattern feedback functions just like error messages in code (Albrant et al., 2023a). Good error messages are, among a few other things, relevant, actionable, user-centered, and courteous (Oshana and Kraeling, 2013). Thus, one of my first steps of this redesign was

to alter the wording for our known anitpatterns and this remains an ongoing process (Albrant et al., 2023a). The following list encapsulates the mental schema I use to adjust each critique message for antipatterns:

1. Five sentences or less
2. Antipattern named accurately & consistently
3. Student-centered & courteous antipattern description
4. Explanation as to why the antipattern is relevant to the students' learning
5. Actionable hint to solve the antipattern

I set the maximum number of sentences a feedback message can have to five in order to minimize cognitive load. Additionally, seven plus or minus two is the commonly held "magic number" for how many items of information a person can hold in their working memory (Baddeley et al., 1994). As the lower end of the magic number, five seems like a good limit to use. Next, every pattern has a name in WebTA's database. For the sake of easier recall and communication, I find it is important for every antipattern to have an accurate and consistent name; this is very similar to naming variables while programming. When describing an antipattern to the student, the description should be centered on their learning, or point-of-view, as well as courteous to them. The description should never place blame or shame onto the student, purposefully or not. Then, the message should contain a brief reason why WebTA searches and

pulls focus to the given antipattern. This is intended to inform the student on the relevancy to validate the learning in their mind. Lastly, WebTA is not intended to grade the students or give them the answers to their problems. It's primary purpose is to gently guide them to best practices and debugging help in the recursive process of programming. Therefore, the feedback message should give a hint on how to solve the antipattern, not directly give the corrected code.

One example of this is with the Unused Variable antipattern (Fig. 2.9). Previously the feedback/critique message was simply:

> Unused variable a.

The new message is:

> **Unused Variable** (a): There is a variable called *a* that has not been used, yet. While having unused variables in your code will not hurt any functionality, it can affect readability and efficiency. It is best to remove it or put it to use.

As for the user interface, Figures 2.6 through 2.10 exhibit the overall look of WebTA's original design as well as highlight specific features that have been tackled through this redesign. Explicitly, WebTA has four main features within its critique details page: submission details (Figure 2.7), summary (Figure 2.8), critique table (Figure

2.9), and downloading critiqued files (Figure 2.7). Overall, the original UI of WebTA can be described in a similar way as its error messages: functional with lots of room for improvement, especially in terms of readability/accessibility and cognitive overload. When it comes to readability and accessibility within the context of code critiquers, the topic is not as thoroughly researched as it should be. However, Fathauer and Rao 2019 do share their experiences and general tips on how to be more accessible within an automated feedback system. Through this, I was able to complete small steps in the hopes of having a sizable impact, with the intention of making much larger progress in the future. For example, efforts were made to use semantic HTML in the website itself to improve usability with a screen reader. According to Fred Paas, cognitive overload is when working-memory becomes "overloaded" due to its limited capacity and lifespan (Paas et al., 2010). There are three different types: intrinsic, extraneous, and germane. Intrinsic cognitive load refers to the overloading caused by the complexity of a topic being learned. Extraneous load relates to the external aspect of the lacking nature of instructional materials while learning. Germane load is the working memory resources, or lack thereof, to combat intrinsic load. Knowing this, is it clear how WebTA's original design contributes to cognitive overload; particularly intrinsic as programming is a complex subject and extraneous as the original design required improvement. Consulting Figure 2.10, when WebTA has spotted a large quantity of antipatterns within the code, this design is quite difficult to visually parse. A part of the new design involved chunking much of the critical information

and allowing the user to hide said information until needed.

3.2 Research Question

The overarching question that leads me to this research is: *RQ: How can I improve WebTA for students?* More accurately, I want to identify the exact areas of WebTA that could cause students to not enjoy the application, not find it helpful, not fulfill an intended purpose, or not find it relevant to them. To attempt to answer this, the process of Design Thinking (Koh et al., 2015, Razzouk and Shute, 2012) was implementedT the first course of action was to listen/observe the students and instructors using WebTA (i.e. empathize and define). This was during the first of three studies (Albrant et al., 2023c). Interviews with instructors for the course (i.e. Study 2) further defined the problem space and aided in the ideation and prototyping phases (Benjamin et al., 2024). That insight, combined with my knowledge of human factors principles and research, I

1. redesigned the user interface of WebTA with the goals of minimizing cognitive overload and improving its aesthetic appeal as well as tested the new design through the opinions of the students (Study 3),

2. will adjust the wording of feedback to students to be actionable, relevant, and courteous (Oshana and Kraeling, 2013), and,

3. will incorporate extensive help documentation with a section for each known (anti)pattern outlining its name, definition, why its a part of WebTA, and a fix-it-yourself example.

Chapter 4

Study 1

This paper was initially published in the 2023 conference proceedings of Frontiers in Education, or FIE (Albrant et al., 2023b). Assignments S17 and S22 are referred to as assignments B and C in future chapters.

During this introduction, the MATLAB WebTA's functionality did not extend past static code-as-text analysis. Thus, it did not contain the functionality to catch every type of antipattern, only some stylistic or syntactic ones that were previously detected and stored within a database as regular expressions. One example, from MATLAB's style guide (Johnson, 2024), is commonly referred to as crowded operators. It is a stylistic antipattern where a programmer does not put enough spacing around an operator of any type.

4.1 Problem Statement

FYE students often struggle with writing clean and effective code due to their limited knowledge of coding conventions, best practices, and common programming pitfalls. As a result, their code may be difficult to read, understand, and maintain. Instructors face challenges in providing timely and personalized feedback to large numbers of students to help them improve their coding skills. Additionally, FYE instructors most commonly have backgrounds in engineering education. Although they have taken a programming class or two, they are not computer scientists and are not trained in CS education. Bridging the gap between computer science and engineering education, there is a need for an automated tool that can analyze students' MATLAB code, identify antipatterns, provide immediate feedback, and suggest related improvements. This paper addresses the problem by presenting the development and testing of WebTA, a code critiquer specifically designed for FYE students using MATLAB, aiming to enhance their coding skills and better prepare them for future engineering courses and careers.

4.2 Methods

The section encompasses several important aspects of the project, including the adaptation of WebTA software, beta testing interventions, data collection, evaluation measures, and data analysis. Each subsection provides unique insights into the modifications made, classroom interventions conducted, data collection procedures, evaluation measures utilized, and the subsequent data analysis techniques employed. By following this systematic approach, the effectiveness of WebTA in enhancing students' coding skills and improving the quality of their MATLAB code is evaluated.

4.2.1 Adaptation of WebTA Software

The existing WebTA software, initially designed for computer science students learning Java [6], was modified to work with the MATLAB programming language. The process required understanding the unique requirements of FYE students and identifying any coding conventions, best practices, and common programming pitfalls relevant to MATLAB. The necessary adjustments were preemptively and concurrently made to the code critiquer algorithms to align with MATLAB programming standards. Additionally, observers for the beta testing interventions kept a lookout for any missed or new antipatterns.

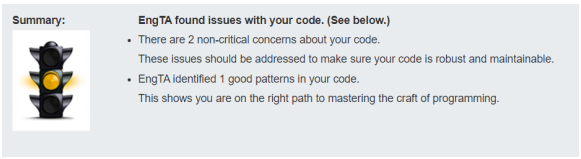

Figure 4.1: Yellow light with syntax error present in the code.

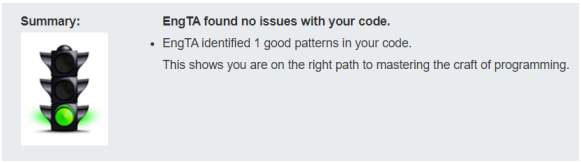

Figure 4.2: Green light with no error present in the code.

4.2.2 Beta Test Interventions

Three beta test interventions were conducted in a classroom setting with a total of 52 FYE students. The students were introduced to WebTA by giving a demo on how to use and navigate through WebTA by the instructors. The WebTA indicates the syntactical and best practices errors with a "Yellow" light, logical or necessary coding requirement errors with a "Red" light, and, code with no presence of error gets a "Green" light. With "Red" and "Yellow" lights, students also receive the necessary corrective actions to be taken to correct the code and progress towards getting a "Green" light. Students were also instructed to submit a screenshot alongside their MATLAB programming assignment for each intervention. The interventions aimed to evaluate the effectiveness of WebTA in enhancing students' coding skills and improving the quality of their MATLAB code.

4.2.3 Data Collection

Feedback generated by WebTA was collected during the beta test interventions to assess the performance and impact of WebTA. The students' MATLAB code submissions were analyzed by WebTA, which identified common mistakes and provided feedback and suggestions for improvement. The feedback generated by WebTA, including the identified antipatterns and recommended code modifications, was recorded for each student.

4.2.4 Evaluation Measures

Various evaluation measures were employed to assess the effectiveness of WebTA. The accuracy of WebTA in detecting coding errors and providing relevant feedback was evaluated by comparing its feedback with manual code reviews conducted by instructors. The improvement in students' coding skills was measured by analyzing the changes in the quality and readability of their MATLAB code submissions, labeled by a traffic light system of severity. Furthermore, student surveys and feedback were collected to gauge their perception and experience of using WebTA and its impact on their learning process.

4.2.5 Data Analysis

The collected data, including feedback from WebTA, instructors' manual code reviews, and student surveys, were analyzed to derive meaningful insights. Statistical analysis techniques were applied to examine the effectiveness of WebTA in identifying coding errors, improving code quality, and enhancing students' coding skills. The results were summarized and presented, highlighting the strengths and limitations of WebTA and providing recommendations for its further improvement. The combination of software adaptation, beta test interventions, data collection, evaluation measures, and data analysis provides a comprehensive approach to evaluating the performance and impact of WebTA as a code critiquer for FYE students using MATLAB.

4.3 Experimental Results

The data collected by the three beta test interventions indicates improvement in students' coding styles and attention to syntax and programming best practices over time (i.e. from 1st classroom intervention to the 3rd classroom intervention beta test conducted). In the 1st classroom intervention, the majority of students received some feedback on their initial code submission to WebTA. The comparison is done in the

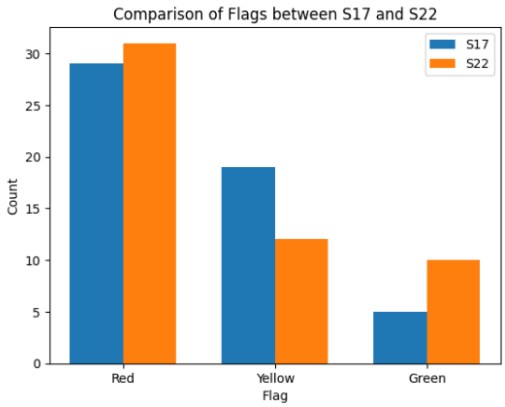

Figure 4.3: Flag comparison in lab intervention with S17 and S22.

second and third lab interventions (i.e. S17 and S22 assignments).

Regarding flags, the authors observed patterns between the two assignments. In S17, the majority of the initial submissions had a "Red" flag, indicating potential issues or challenges. In S22, the number of "Yellow" flags was nearly halved and the number of "Green" flags doubled. These trends most likely display how students became more familiar with WebTA, MATLAB, or both, as many of them Additionally, observations made during both of these class times insinuate that a number of the "Red" flags were due to a strict assignment requirement regarding comments, not the presence of true critical antipatterns. This was noted during both assignments.

When analyzing attempts (i.e. submissions by students in which they get feedback from WebTA, the authors found interesting variations in student efforts). In S17, some students made a high number of attempts, while others had relatively fewer attempts. However, in S22, the attempts were more evenly distributed among the

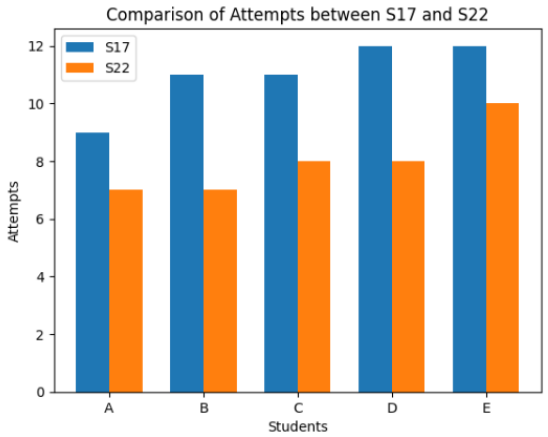

Figure 4.4: Comparison of the number of attempts taken by students for S17 and S22.

students, indicating a more balanced level of engagement.

Overall, these findings suggest that the students demonstrated improvements in their performance and engagement between S17 and S22. The decrease in "Yellow" flags and increased "Green" flags attempts in S22 indicate a positive progression in student performance and dedication to the assignments. These insights can inform future instructional strategies and interventions to enhance student outcomes and promote a more equitable learning environment.

To further evaluate the usability of WebTA, the authors collected surveys and feedback from students. Here are some insights on what the students think about the ease of use and ease of navigation for the WebTA tool.

The majority of the students found the WebTA tool easy to navigate and easy to use (Fig. 4.6 and Fig. 4.5). The most important factor from the survey "What did

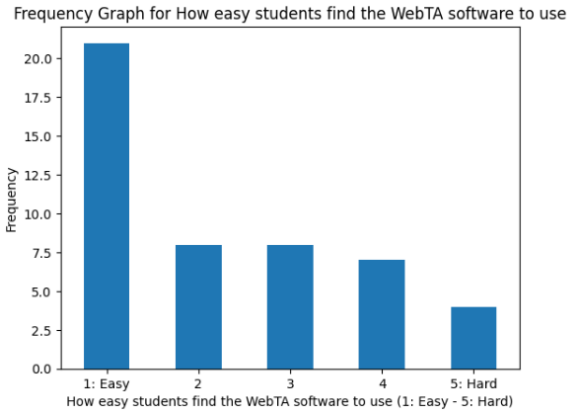

Figure 4.5: Ease of use of the WebTA.

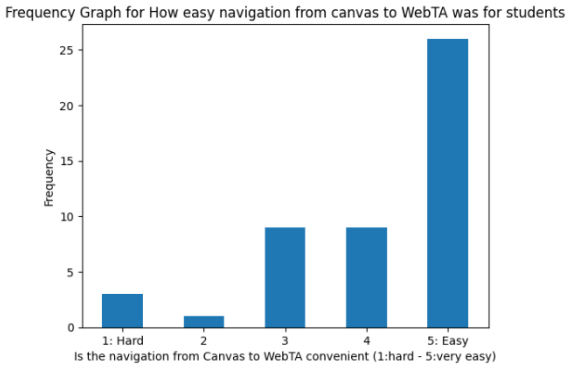

Figure 4.6: Ease of Navigation.

students learn with the help of the WebTA" or was the tool helpful for them was thoroughly analyzed and to get the standard responses and compare for meaningful insights on the usefulness of WebTA (Fig. 4.7).

It is evident that the majority of students found WebTA helpful with understanding or rectifying the syntactical and/or formatting errors with MATLAB programming. The contribution to category "Nothing" says that students either got a "Green" flag in the first go or they knew about how to resolve the error without the help of WebTA. This category holds the second highest importance because the graph represents survey

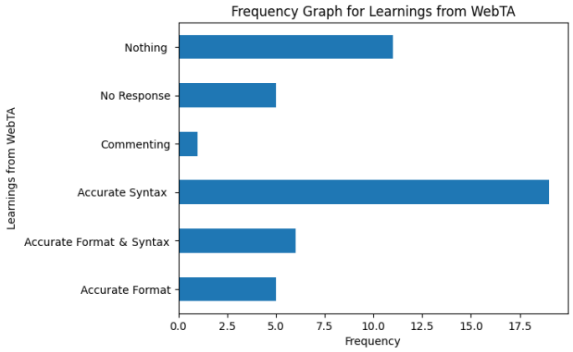

Figure 4.7: Categorical representation of what students learned (self-reported) from WebTA.

data collected from all three beta classroom interventions combined, demonstrating that as students improved, they received positive outcomes in subsequent interventions.

Overall, the initial data analysis and the experimental results show the great potential of the WebTA tool in helping students learn and improve coding standards.

4.4 Conclusion and Discussion

The data collected from the three beta test interventions heavily indicate a great improvement in students' coding styles and adherence to syntax and programming best practices over time. In the initial intervention, a majority of students received feedback on their initial code, with 95% of students receiving "Yellow" or "Red" flags indicating areas for improvement. However, with increased familiarity and experience with the WebTA tool, the performance improved in subsequent interventions, with

an increase in the number of students receiving the "Green" flags on their initial code submission. This demonstrates the positive impact of WebTA on the coding style of engineering students with no prior coding background. Surveys and feedback from students further support the usability and effectiveness of WebTA. The majority of students found the tool easy to navigate and use, indicating its user-friendly nature. Analysis of the survey responses confirmed that students perceived WebTA as a valuable tool in identifying and rectifying coding errors, with the majority attributing their learning and improvement to its assistance. The results and insights obtained from the data analysis and experimental outcomes affirm the great potential of the WebTA tool in helping students learn and enhance their coding skills. The significant improvement observed in students' coding styles, coupled with positive feedback on the tool's usability and educational value, supports its integration into the curriculum for FYE students using MATLAB. Further discussions can revolve around the implications of these findings for enhancing coding education, addressing any limitations of the study, and exploring avenues for future research and improvements to WebTA.

The future scope of this project extends to exploring how the WebTA tool can assist instructors in obtaining clean and effective code from novice programmers. By collecting feedback from instructors, valuable insights can be gained to further enhance WebTA's usability and effectiveness from an instructor's perspective. This would involve incorporating features and functionalities that specifically cater to the needs and preferences of instructors, making the tool more instructor-friendly. By aligning the

tool with instructors' requirements, it can better support their teaching process and streamline the code evaluation and feedback process. Additionally, future research could delve into evaluating the long-term impact of WebTA on students' coding skills beyond the first-year level, as well as investigating its potential application in other programming languages and disciplines. These endeavors would contribute to the continuous improvement and refinement of WebTA as a valuable tool for both students and instructors in the realm of coding education. Furthermore, to gain more comprehensive insights and evaluate the long-term impact of the WebTA tool, the experiment will be extended to cover a full semester. This extended duration will allow for a more in-depth analysis of students' coding skills, progress, and the effectiveness of WebTA over an extended period. Continuing the experiment for a full semester enables the assessment of WebTA's sustained impact on students' coding abilities, identification of challenges, and a comprehensive understanding of the benefits and limitations of WebTA in supporting coding education for FYE students using MATLAB.

Chapter 5

WebTA's Re-Imagined Design

Screenshots of the new design of WebTA are found in Figures 5.1 through 5.9. The new design was aimed to improve five main features of WebTA: 1) assignment instructions & submission, 2) previous submissions/critiques, 3) critique details, 4) critique summary, 5) and critique table.

5.1 Feature 1: Assignment Instructions & Submission

This feature (Figure 2.4) is found when a student selects an assignment from the list on their course's main page. The new design (see Figure 5.1) has four primary

Figure 5.1: After selecting an assignment, the student will be met with a web-page like this.

changes: 1) a "submit" button near the name of the assignment that, when clicked, will automatically scroll them to the actual submit/critique button, 2) the removal of the large traffic light image, 3) a more blatant (bright blue) design to the critique button, and 4) the addition of displaying the assignment's due date just below its title.

5.2 Feature 2: Previous Submissions/Critiques

This feature (Figure 2.5) is found when a student selects an assignment from the list on their course's main page that they have already submitted to at least once. The

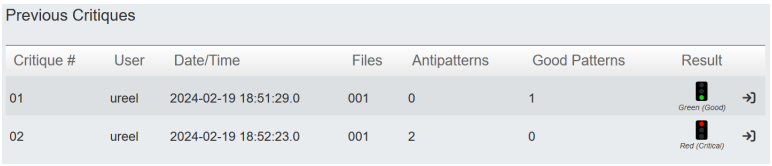

Figure 5.2: After a student has submitted to WebTA at least once, this table appears below the "Critique" button.

new design (Figure 5.2) revised the original's condensed list of links to be a table that is more readable and provides a bit more information about each critique in the table.

5.3 Feature 3: Critique/Submission Details

This feature is contained within the individual critique pages (Figure 2.7). The new design groups the critique page into three different parts, as opposed to one, typically long page (Figure 5.3). The details of the critique (e.g. user, submission time, assignment name, etc.) mostly remain in a similar place, except the assignment's name, which was moved above as the page's title. Additionally, a larger traffic light icon was placed in line with the details labeled "Overall Result" to bring that to a students' attention sooner. Lastly, a comment box was added to allow for students to provide feedback directly to researchers.

The ability to download files, another aspect of this feature, was moved to its own section (Figure 5.4). Moreover, the ability to download a subset of source files, as

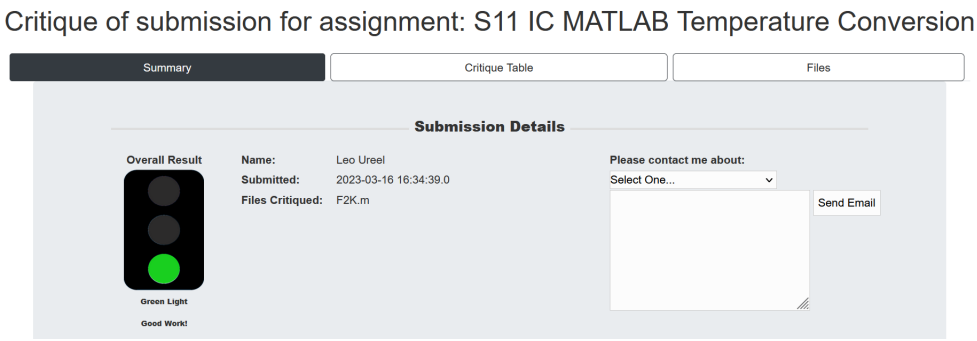

Figure 5.3: The ability to download source files has its own section as well as the ability to download select files, instead of all or one.

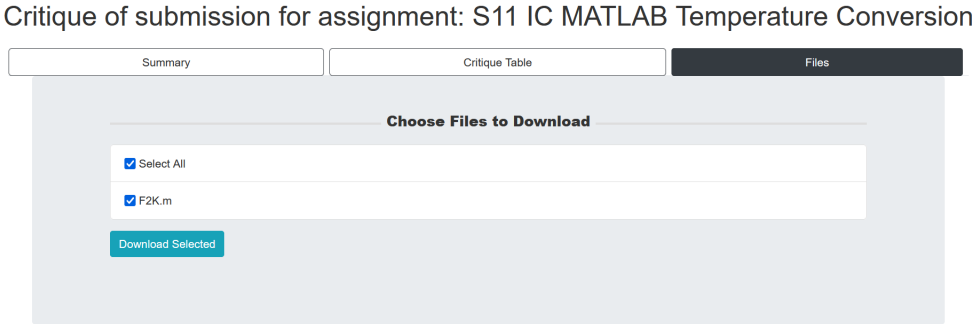

Figure 5.4: The ability to download source files has its own section as well as the ability to download select files, instead of all or one.

opposed to all or one at a time, was added through the use of a list of check boxes.

5.4 Feature 4: Critique Summary

This feature is contained within the individual critique pages and placed just below the critique details section (Figure 5.5). This new design aims to condense the original's bullet point list with multiple sentences to a table with numbers and their explanations separated by columns. The goal of this is for students, after they understand what the traffic light color represents, can more easily ignore the lights'

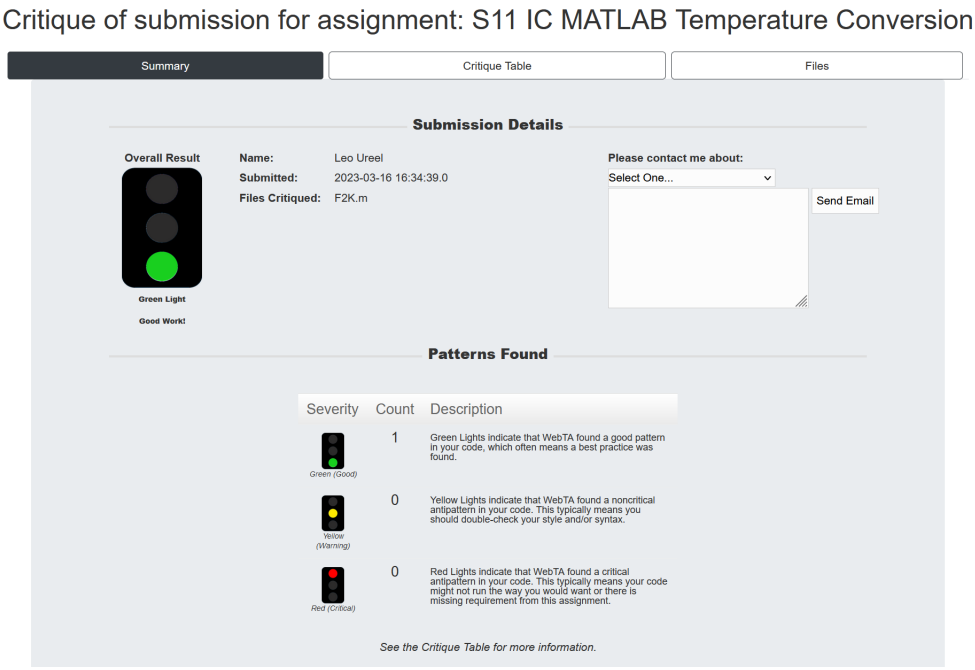

Figure 5.5: After submitting code, or selecting a previous critique from the list, the students will see this web-page.

descriptions as it becomes unnecessary for them to read. In turn, they can focus on the numbers more.

5.5 Feature 5: Critique Table

This line by line breakdown of the students code received an upgrade by minimizing the amount of information shown at first (Figure 5.6). Instead of listing all of the found patterns in the rightmost column of the table for every line, the details for every pattern found gets effectively hidden. Every line of code gets a traffic light in the right most column. A green light can indicate the absence of any antipatterns

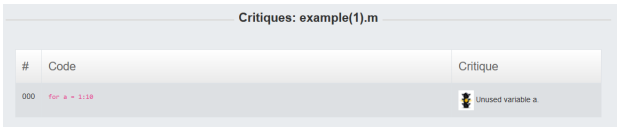

Figure 5.6: This is a closer look at the feature which displays antipatterns and good patterns found on a line by line basis.

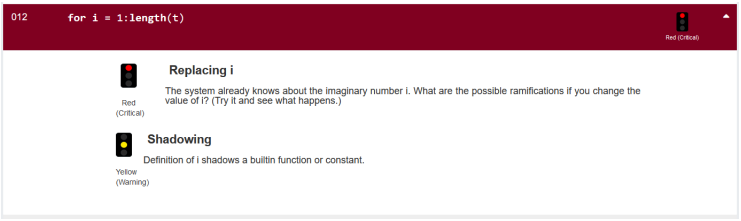

Figure 5.7: When a student clicks on a row that contains a critical antipattern, the row will expand in this fashion.

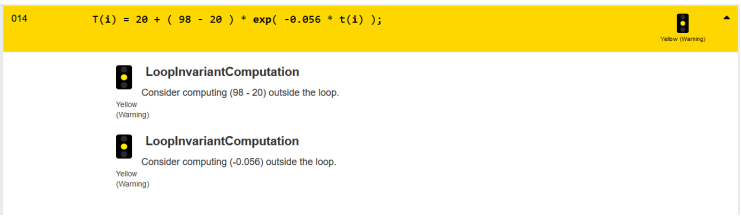

Figure 5.8: When a student clicks on a row that contains a warning antipattern, the row will expand in this fashion.

or the presence of a good pattern. The lines of code with at least one pattern gain a "down caret" to the right of its traffic light icon, which universally indicates the container can be expanded. Upon expansion, that row will be highlighted in a color that matches its overall status/light. A list of all patterns present gets revealed below the now highlighted row. Examples of expanded rows are within Figures 5.7 through 5.9. This feature's redesign is the key area for reducing cognitive load for the students.

Figure 5.9: When a student clicks on a row that contains a good pattern, the row will expand in this fashion.

Chapter 6

Study 2

6.1 Introduction

With this first iteration of design thinking, this second study's purpose is to test the new user interface. DT's test phase involves "getting feedback, modifying the design, reevaluating it, and making a decision to accept or reject the modeled idea" (Razzouk and Shute, 2012). This study outlines the feedback and evaluation portions, then discusses future modifications and the decision of accepting/rejecting certain features. The goal of this evaluation is to determine if the new design's aesthetic appeal, perceived sense of usefulness and purposefulness is better to students in comparison the original design as well as formally review WebTA's overall usability and

user experience as well as its learnability.

6.2 Methods

6.2.1 Training

To close a gap in antipattern knowledge, I designed a training session based in the Interactive, Constructive, Active, and Passive (ICAP) framework that was carried out before WebTA's integration for the semester of Spring 2024. The training was conceived to be open-ended and encourage a two-way flow of information between us researchers with the professor and teaching assistants for the course Benjamin et al. (2024). The instructors learn how best to utilize patterned language and WebTA as a tool. Researchers record the session to analyze and receive an even better understanding of instructors' perceptions of the project and ways in which WebTA can improve.

6.2.2 Survey Instruments

To evaluate the students' opinion on the redesign and general usage of WebTA, two surveys were distributed to the students of ENG1101. The first survey uses A/B

testing to directly compare the old design to the new. It provides screenshots of the above features in old design and my design of WebTA. For each feature presented, three questions are asked. 1) Which of the two interfaces do you find to be more appealing? 2) Which of the two interfaces do you find to be more useful? And, 3) Which of the two interfaces do you find to fulfill the described purposes better?. The second survey utilizes questions from the the User Experience Questionnaire, or UEQ (Schrepp et al., 2017). This questionnaire consists of 26 seven point scales with end-points like "attractive" to "not attractive". I hypothesize that the first survey will reveal students finding "Design B" (i.e. the redesign) to be more appealing, more useful, and equally purposeful. The second survey will unveil students' generalized perceived drawbacks of WebTA's user experience in which I can address. By combining a training session with the instructors and revamping the design of WebTA, I aim to improve the students' overall user experiences. In doing so, it can improve their learning and encourage a passion for programming in MATLAB.

6.2.3 Student Behavior Data

Across all three semesters, WebTA collected the following data (and more) on every submission: 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 the three semesters. The analysis' results suggest that student behavior remained relatively similar.

6.3 Results

6.3.1 Survey Instruments

6.3.1.1 A/B Testing

This survey's results are shown in Figure 6.1 through Figure 6.8. Overall, students heavily leaned towards a preference with Design B (i.e. the new UI) for appeal, perceived usefulness, and perceived purpose fulfillment.

Figures 6.1 through 6.4 possesses four graphs that display the total counts of answers (i.e. "A" for the original design, "B" for the new design, or "They are equal") across all five of the UI features highlighted by the survey and this paper. Figure 6.1 shows the count of the summation of all three qualities (i.e. Appeal, Usefulness, and Purpose) across all five features. With a total possible count of 300 (5 features/Q's x 60 students), the number of all "B" answers for the three qualities per feature is

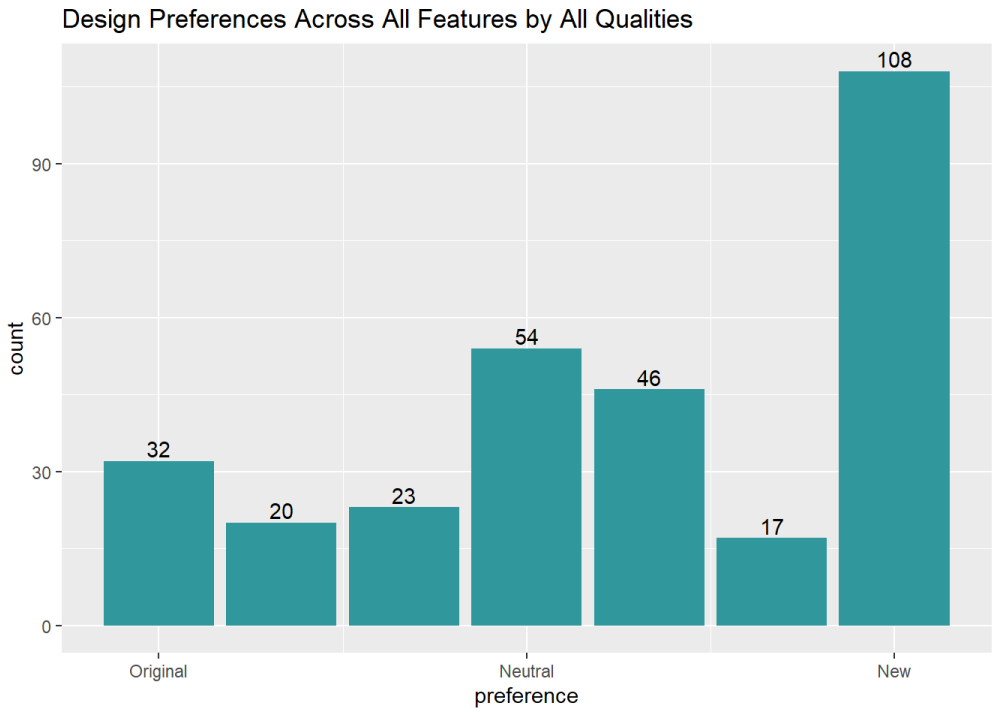

Figure 6.1: This graph shows the count of students' preferences across all five features grouped by the sum of all three qualities (appeal, usefulness, and purpose) asked about. For example, if a student chose Design B/New for all three qualities of Feature 1, that adds to the far right bar, marked "New" in the X-axis.

greater than 90. This implies that the majority of students prefer the new design's version of the five features in terms of appeal, usefulness, and fulfillment of purpose. Figure 6.2 shows that over half (¿ 150) of the responses to questions asking which design is more appealing chose the new design. Figure 6.3 a very similar trend. Although, Figure 6.2 has less neutrality (i.e. "They are equal" answers) than Figure 6.3. Finally, Figure 6.4 reveals that the same proportion of students view the new design as equally fulfilling its features purposes as those who view the new design as fulfilling the features' purposes better than the original design.

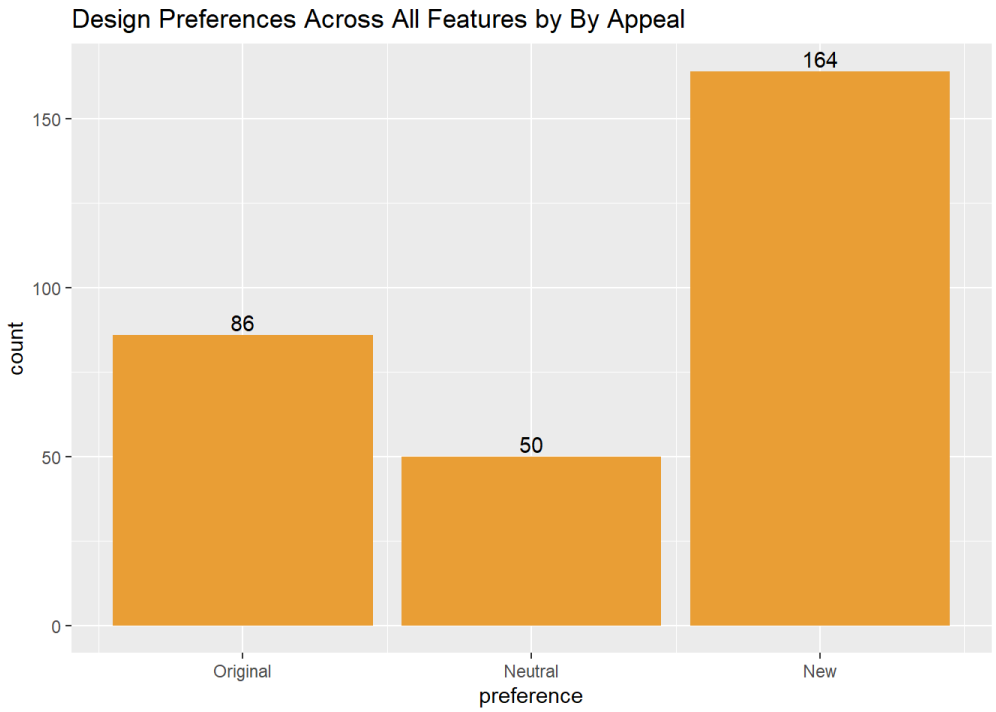

Figure 6.2: This graph shows the count of students' preferences across all five features for the appeal quality. For example, if a student chose Design B/New for all five features on their appeal questions, that adds a count of five to the far right bar, marked "New" in the X-axis.

Figures 6.5 through 6.8 contain four graphs that separate out the distributions of the answers by the individual features. Figure 6.5 reveals that features #2 (Previous Submissions/Critiques) and #5 (Critique Table) are the most neutrally rated across appeal, usefulness, and purpose. Figure 6.6 demonstrates that across all features, students strongly preferred the new design's appearance over the old one. Features #1 (Assignment Instructions & Submission) and #4 (Critique Summary) show the most preference to the original design's appeal. Upon coding the follow-up questions' (i.e. "What about that design is more appealing?") responses, majority of the students who preferred the original design enjoy the image of a traffic light on the assignment

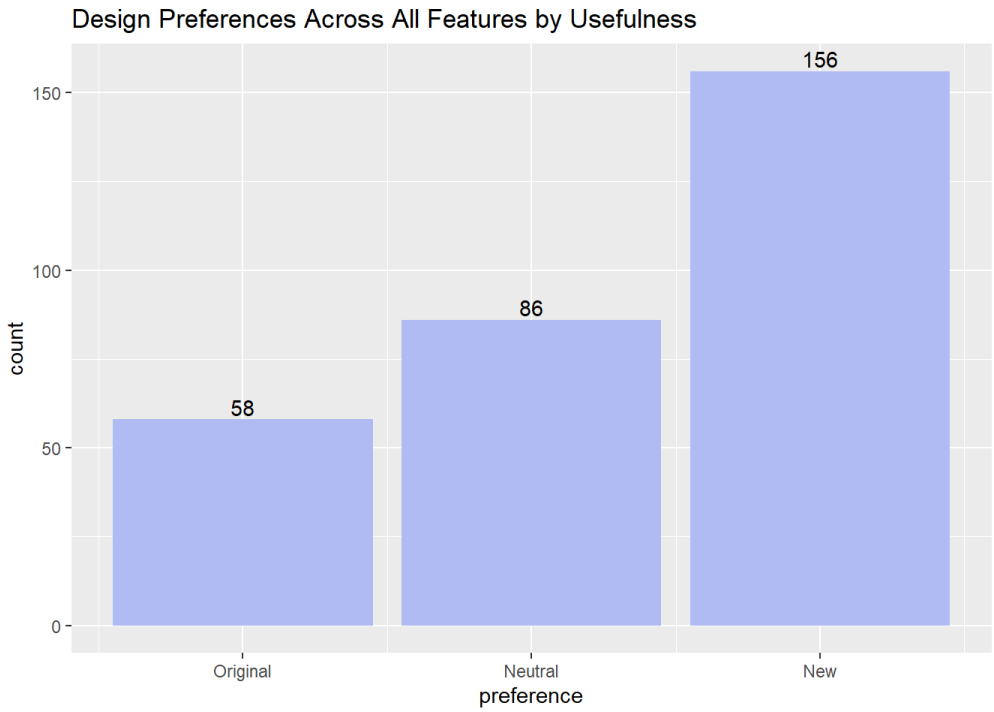

Figure 6.3: This graph shows the count of students' preferences across all five features for the usefulness quality. For example, if a student chose Design B/New for all five features on their usefulness questions, that adds a count of five to the far right bar, marked "New" in the X-axis.

page. Figure 6.7 displays that for the first four features, students overwhelmingly see the new design as more useful. However, feature five (critique table) is perceived as useful as the original. Figure 6.8 shows a similar trend with feature 5 as Figure 6.7. Except, feature #1 joins that trend of equality/neutrality as well.

Comprehensively, the survey's results strongly indicate that, on average, students view the new design to be more appealing, more useful, and better fulfills its purpose than the original design.

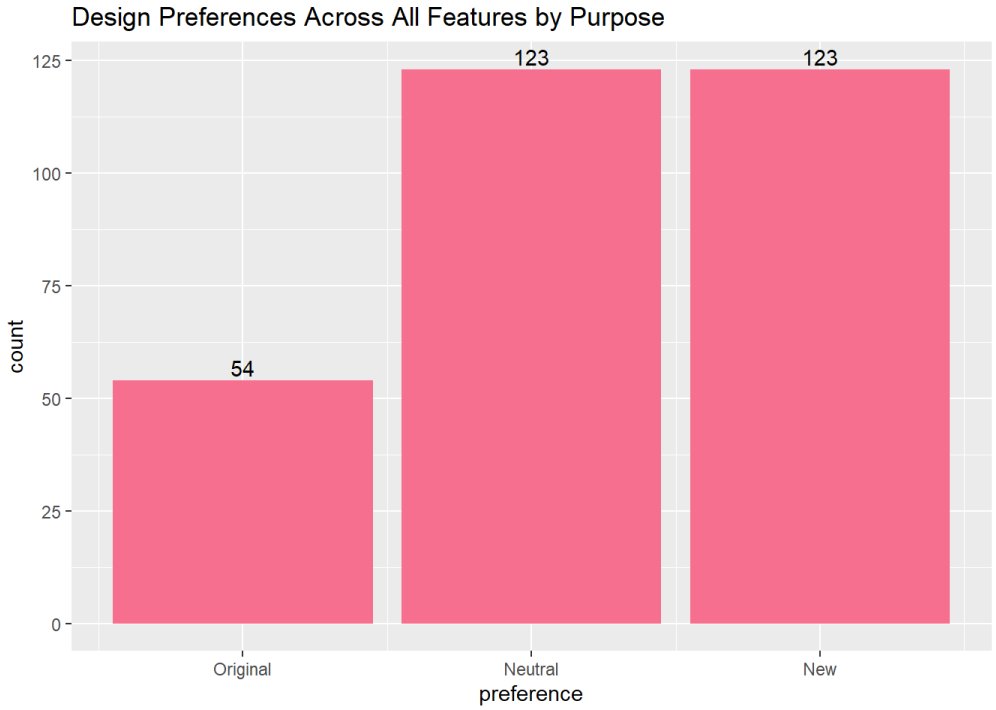

Figure 6.4: This graph shows the count of students' preferences across all five features for the purpose quality. For example, if a student chose Design B/New for all five features on their purpose questions, that adds a count of five to the far right bar, marked "New" in the X-axis.

6.3.1.2 User Experience Questionnaire

The researchers of the UEQ provide a resource, in the form of a Microsoft Excel sheet, to aid in the analysis the UEQ's answers (Schrepp et al., 2017). Figures 6.9 through 6.11 contain the outcomes of utilizing said resource.

Figure 6.9 displays the average score/value given for each of the 26 questions, or items. The majority of the items show that, on average, students rated WebTA positively

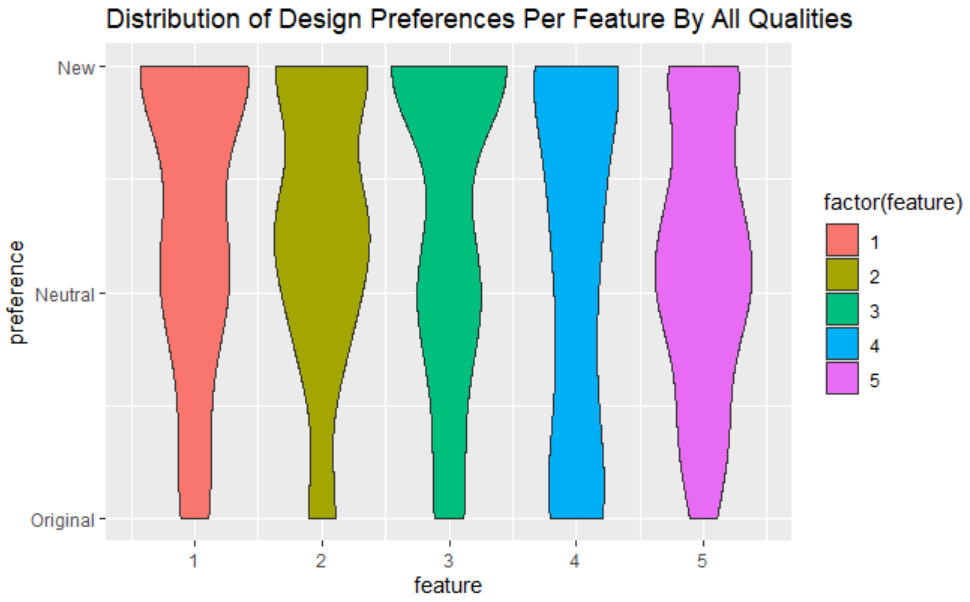

Figure 6.5: This violin plot shows the distribution of students' preferences between the old (A) UI design and the new (B) UI design. It is broken down by the five features, as mentioned in the above paragraph, focused on in the redesign and the summation of all three qualities (appeal, usefulness, and purpose).

but almost neutral. The worst rated items include: boring/exciting, usual/leading edge, demotivating/motivating, unattractive/attractive, and unpredictable/predictable. The most concerning from that list is students viewing WebTA as neutral when it comes to motivation. The best rated items include: not understandable/understandable, slow/fast, complicated/easy, not secure/secure, impractical/practical, and cluttered/organized. While not secure/secure may not wholly apply to WebTA, it is very encouraging for all of these items to be rated, on average, above a 1 (i.e. 4 out of 7) on the scale.

Figure 6.10 shows the distributions of the answers per item. This demonstrates more clearly which of poorly rated items were due to neutrality or a more negative

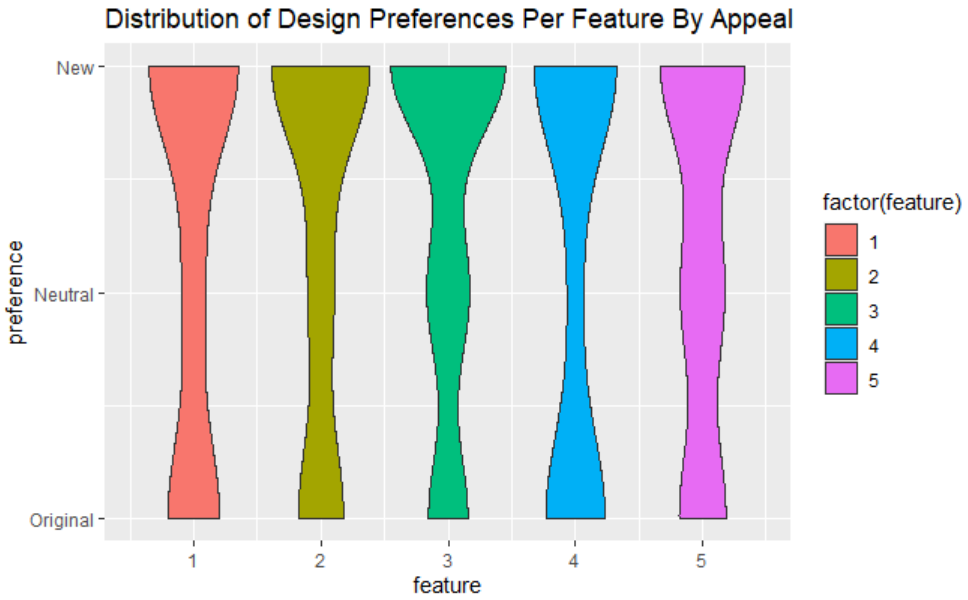

Figure 6.6: This violin plot shows the distribution of students' preferences of appeal quality between the old (A) UI design and the new (B) UI design. It is broken down by the five features, as mentioned in the above paragraph, focused on in the redesign.

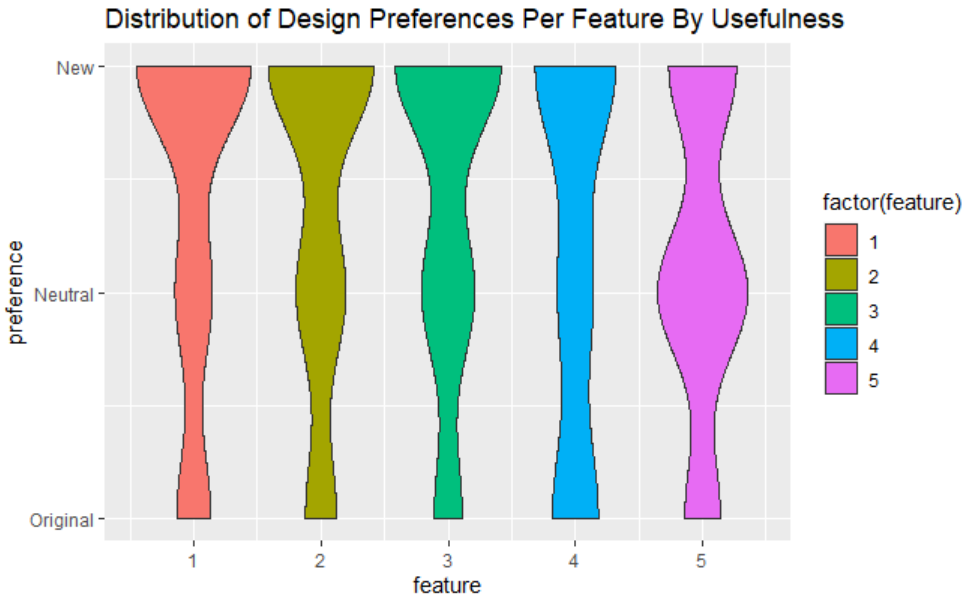

Figure 6.7: This violin plot shows the distribution of students' preferences of usefulness quality between the old (A) UI design and the new (B) UI design. It is broken down by the five features, as mentioned in the above paragraph, focused on in the redesign.

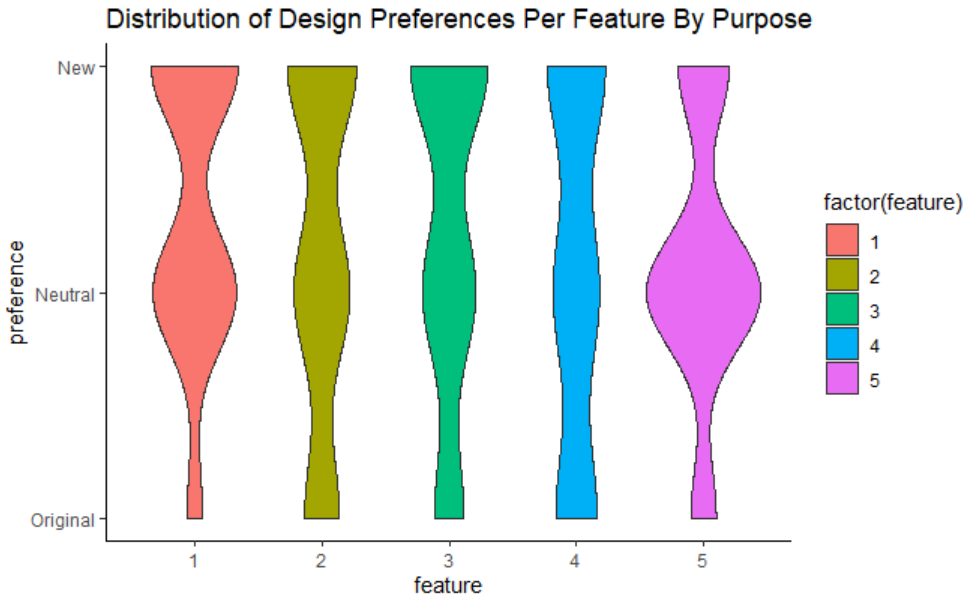

Figure 6.8: This violin plot shows the distribution of students' preferences of purpose quality between the old (A) UI design and the new (B) UI design. It is broken down by the five features, as mentioned in the above paragraph, focused on in the redesign.

perception. For example, usual/leading edge is more neutrally rated than demotivating/motivating, which has relatively even proportions between negative and positive rating.

Figure 6.11 discloses the average answer based on the UEQ's six scales (i.e. Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation, and Novelty). This shows that of the 26 items, those that relate to efficiency and perspicuity (or clearness of WebTA) are what WebTA has the most of. In contrast, WebTA still has quite a bit of room for improvement in terms of stimulation and attractiveness.

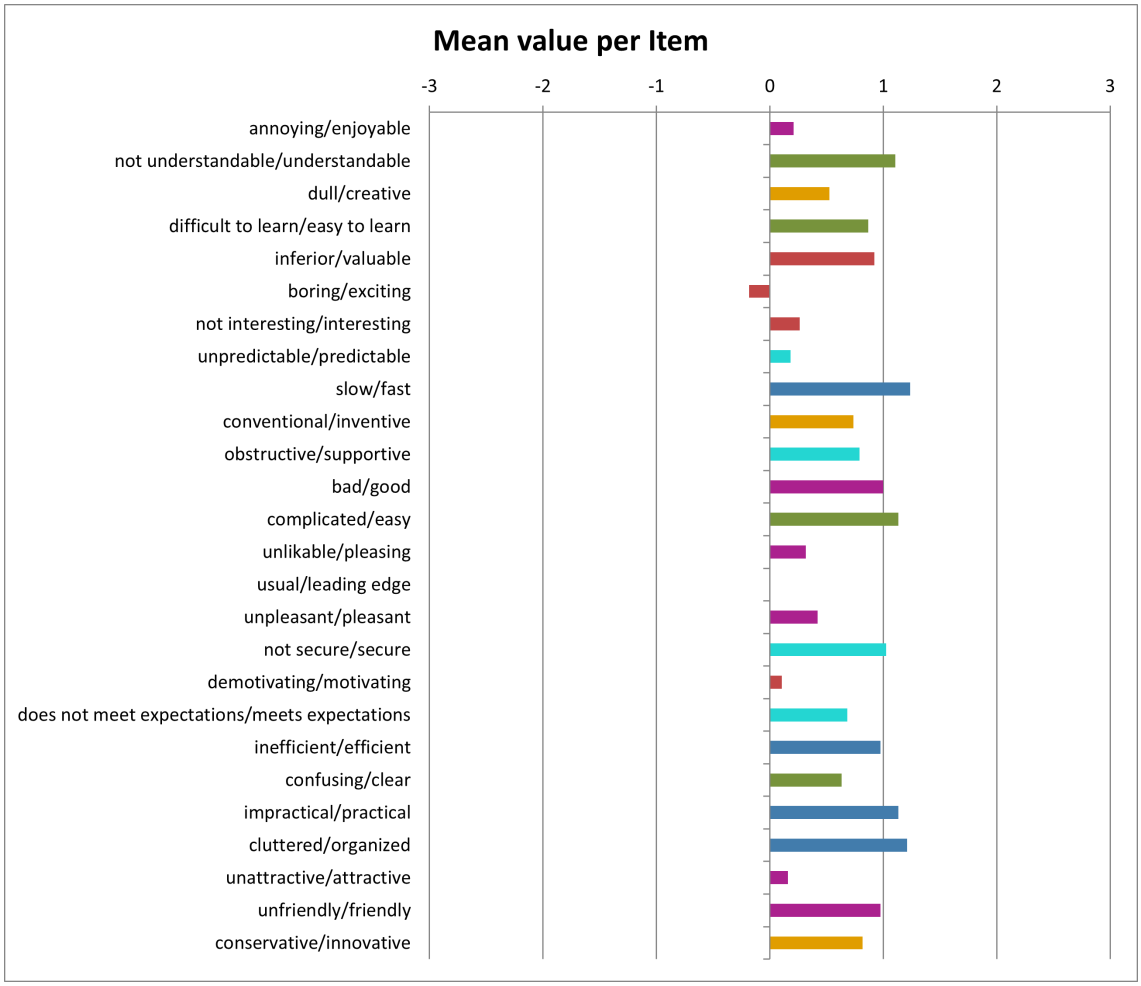

Figure 6.9: This bar graph shows the average answer of students for each of the 26 items that the UEQ looks at.

6.3.2 Student Behavior Data

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. Then, the three assignments for Spring 2024 will be G, H, I. Table 6.1 shows a general overview of the submissions made by students across the assignments/semesters.

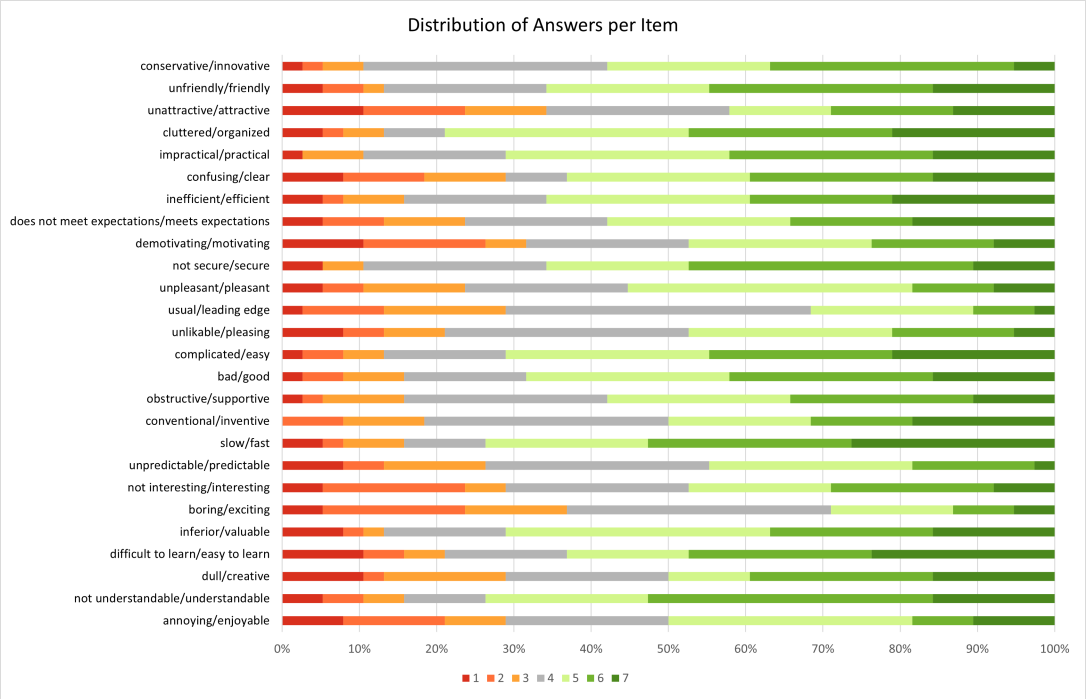

Figure 6.10: This bar graph shows the distribution of students' answers for each of the 26 items that the UEQ looks at.

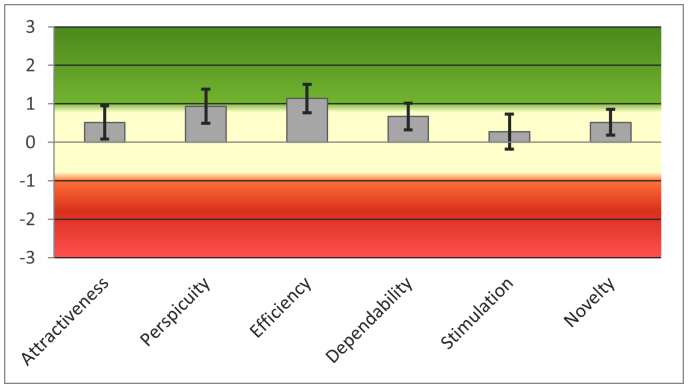

Figure 6.11: This bar graph shows the average students' answers for each of the six scales that the UEQ looks at.

Submissions By Assignment					
Assignment	Total	Min	Max	Median	Mean
Spring 2023					
A	21	0	7	0	0.32
B	281	0	12	4	4.32
C	172	0	10	2	2.65
Fall 2023					
D	262	0	10	1	1.76
E	107	0	6	0	0.72
F	438	0	17	2	2.94
Spring 2023					
G	133	0	8	1	2.33
H	93	0	9	1	1.63
I	87	0	9	1	1.53

Table 6.1
A breakdown of submissions for each assignment.

6.3.2.1 Spring 2023

As table 6.1 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.

Figure 6.12 graphs the number of patterns found in students' submissions over time, color-coded by assignment. The thick lines show the mean number of patterns found

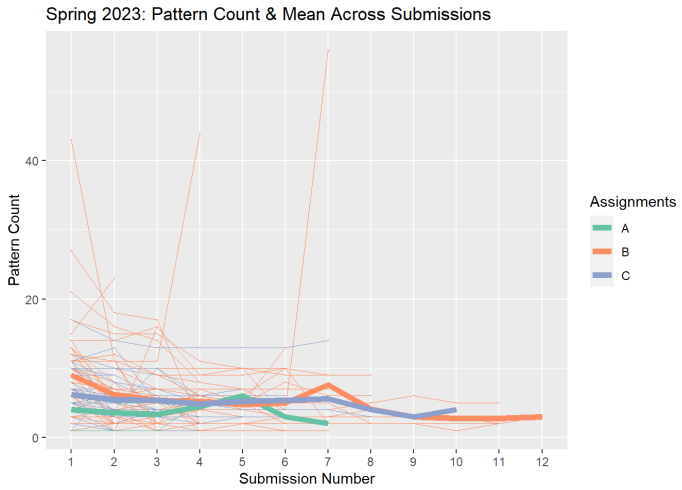

Figure 6.12: This data is from the Spring 2023 semester. Each thin line on this graph represents the number of patterns found across a student's submission(s) for a given assignment. The X-axis of submission number acts as time. The colors differentiate which assignment and the thick lines display the average across students at that submission/attempt number.

for a given submission number per assignment. Table 6.2 shows the results from a simple linear regression model on the thicker, mean lines from Figure 6.12. 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.

Linear Regression of Mean Per Assignment		
Assignment	Estimated Coefficient	p-value
A	-0.5746	$p < 0.001$
B	-0.2879	$p < 0.001$
C	-0.2806	$p = 0.006$

Table 6.2
The estimated coefficients and p-values of linear regression on the mean lines from Figure 6.12.

Figure 6.13 shows the distribution of the number of submissions made by students. The closest assignment to a normal distribution is assignment B. However, all three

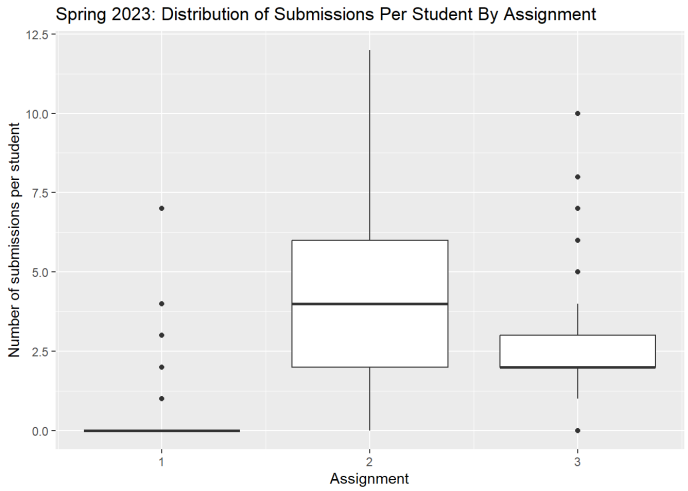

Figure 6.13: This data is from the Spring 2023 semester. This boxplot shows the distribution of the number of attempts/submissions that students made per assignment.

assignments are skewed towards zero submissions.

Figure 6.14 observes ratio of a pattern's presence over students' submissions (time). 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.

6.3.2.2 Fall 2023

This semester is peculiar compared to the other two 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 WebTA submissions) that the Spring semesters.

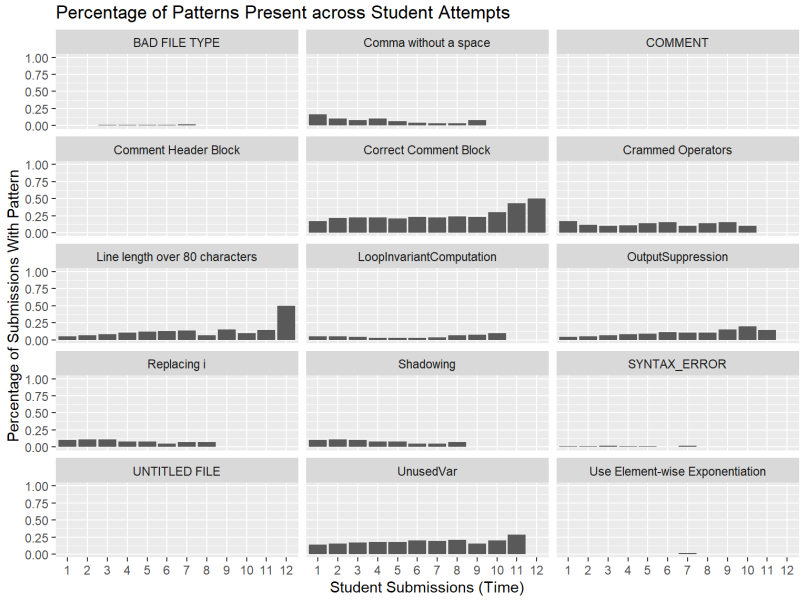

Figure 6.14: This series of bar charts shows the ratio of students whose code contained the given pattern at the given submission number. For example, in the Spring 2023 semester, no students that made 10 or more submissions had the "Comma without a Space" antipattern by their 10th submission.

Figure 6.15 graphs the number of patterns found in students' submissions over time, color-coded by assignment. 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 6.3 shows the results from a simple linear regression model on the thicker, mean lines from Figure 6.15. This table reveals with good confidence ($p < 0.001$) that the slope of assignment E is negative.

Linear Regression of Mean Per Assignment		
Assignment	Estimated Coefficient	p-value
D	0.6825	$p = 0.018$
E	-0.6464	$p < 0.001$
F	0.4260	$p = 0.008$

Table 6.3
The estimated coefficients and p-values of linear regression on the mean lines from Figure 6.15.

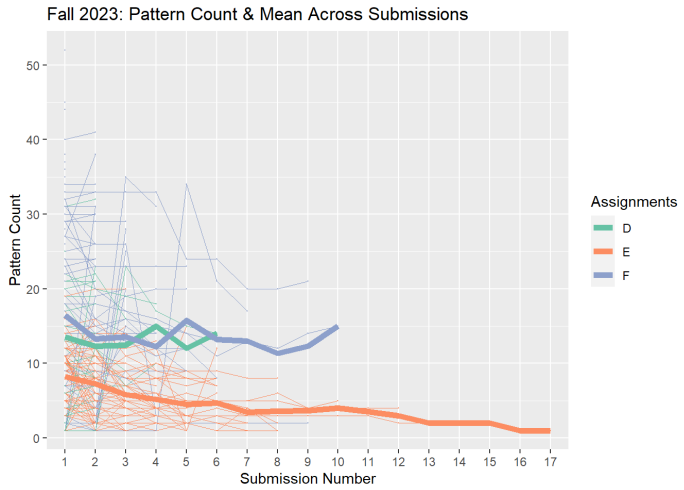

Figure 6.15: This data is from the Fall 2023 semester. Each thin line on this graph represents the number of patterns found across a student's submission(s) for a given assignment. The X-axis of submission number acts as time. The colors differentiate which assignment and the thick lines display the average across students at that submission/attempt number.

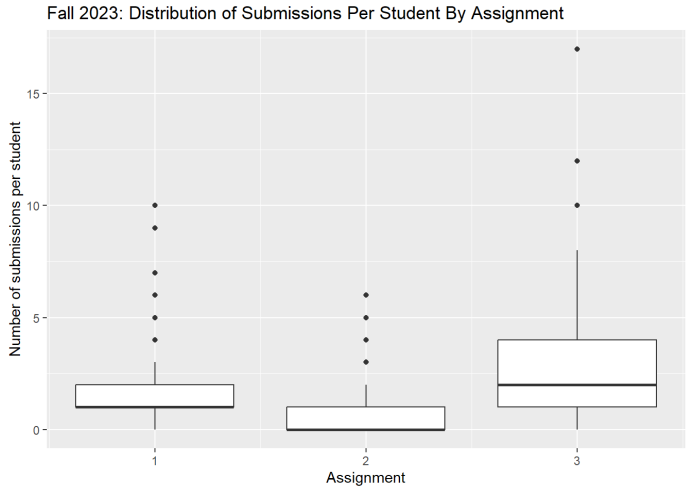

Figure 6.16: This data is from the Fall 2023 semester. This boxplot shows the distribution of the number of attempts/submissions that students made per assignment.

Figure 6.16 shows the distribution of the number of submissions made by students. The closest assignment to a normal distribution is assignment B. However, all three assignments are skewed towards zero submissions.

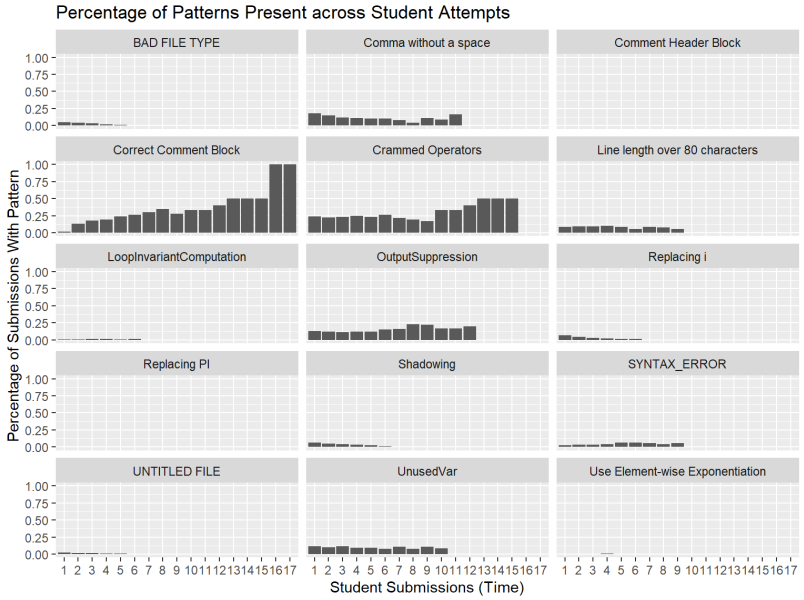

Figure 6.17: This series of bar charts shows the ratio of students whose code contained the given pattern at the given submission number. For example, in the Fall 2023 semester, no students that made 11 or more submissions had the "Comma without a Space" antipattern by their 11th submission.

Figure 6.17 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.

6.3.2.3 Spring 2024

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

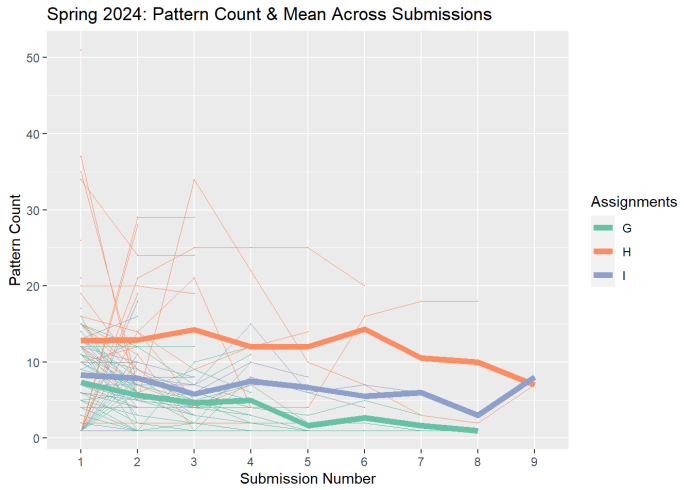

Figure 6.18: This data is from the Spring 2024 semester. Each thin line on this graph represents the number of patterns found across a student's submission(s) for a given assignment. The X-axis of submission number acts as time. The colors differentiate which assignment and the thick lines display the average across students at that submission/attempt number.

Figure 6.18 graphs the number of patterns found in students' submissions over time, color-coded by assignment. The thick lines show the mean number of patterns found for a given submission number per assignment. Table 6.4 shows the results from a simple linear regression model on the thicker, mean lines from Figure 6.18. This table reveals with good confidence ($p < 0.001$) that the slope of assignments G and I are negative.

Linear Regression of Mean Per Assignment		
Assignment	Estimated Coefficient	p-value
G	-1.3365	$p < 0.001$
H	0.1071	$p = 0.587$
I	-0.6509	$p = 0.003$

Table 6.4
The estimated coefficients and p-values of linear regression on the mean lines from Figure 6.18.

Figure 6.19 shows the distribution of the number of submissions made by students.

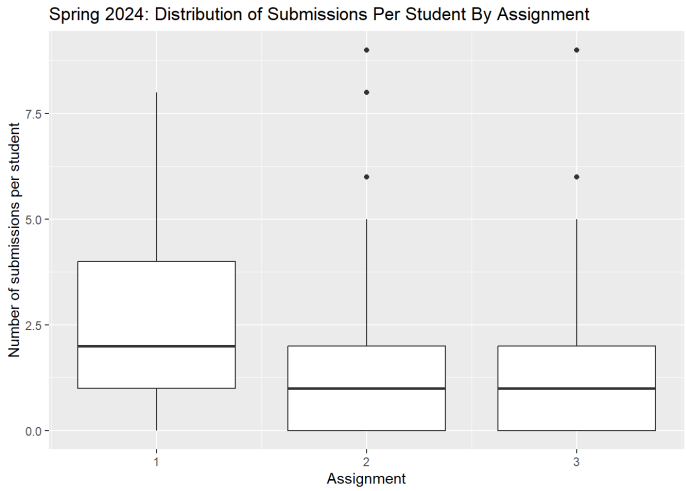

Figure 6.19: This data is from the Spring 2024 semester. This boxplot shows the distribution of the number of attempts/submissions that students made per assignment.

The closest assignment to a normal distribution is assignment B. However, all three assignments are skewed towards zero submissions.

Figure 6.20 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.

6.4 Discussion

While it is difficult to make broad statements on education-related research, especially pilot data, this study still provides a sturdy base for a first iteration of design thinking.

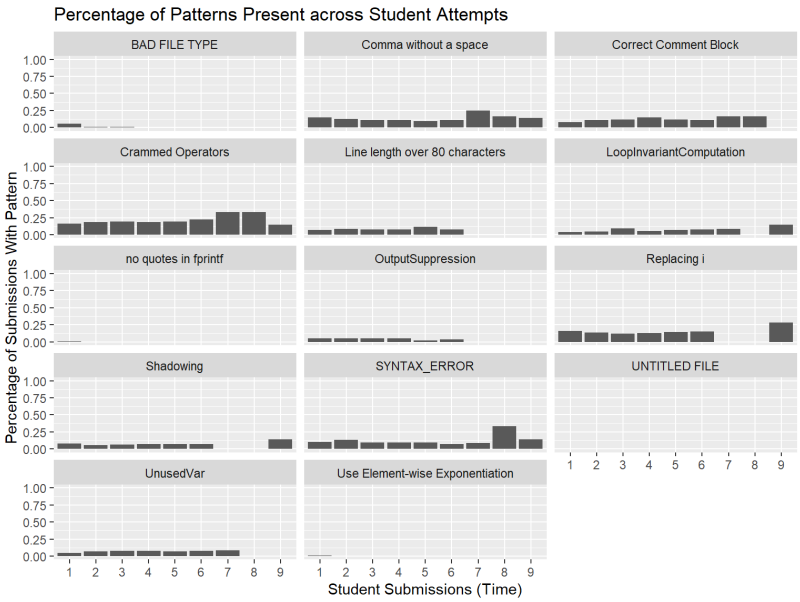

Figure 6.20: This series of bar charts shows the ratio of students whose code contained the given pattern at the given submission number. For example, in the Spring 2024 semester, no students that made 7 or more submissions had the "Line Length Over 80 Characters" antipattern by their 7th submission.

The key takeaways from this study include:

1. The average student prefers the aesthetics of the new design over the old design (Figs. 6.2, 6.6).

2. The average student perceives the new design to be more useful than the old design (Figs. 6.3, 6.7).

3. The average student perceives the new design to fulfill the given purpose(s) more than the old design (Figs. 6.4, 6.8).

4. The overall usability and user experience of WebTA is seen in a neutral to slightly positive light by students, though it does vary between students. (Figs.

6.9 - 6.11).

5. There is evidence of WebTA's learnability with this pilot data as the average student is able to decrease the number of antipatterns found in their code. This is a trend found in multiple assignments across multiple semesters and two different UI designs (Figs. 6.12, 6.15, 6.18; Tabs. 6.2,6.3,6.4).

With ample the room for WebTA to grow, future modifications are necessary. There are numerous minor changes that can make the process more efficient, such as a "re-submit" button on individual critique pages— as opposed to how they currently have to go back to the assignment page to submit more code. Following any modifications will include additional testing. Each test phase will determine if that modification and/or feature remains as part of WebTA. In regards to the five changed features redesigned and tested in this thesis, the decision remains to accept them with the inclusion of minor changes and brand new features to be tested in the coming semesters.

Chapter 7

Discussion & Future Work

Chapter 4, or Study 1, was first published in the 2023 conference proceedings of Frontiers in Education, or FIE (Albrant et al., 2023b).

7.1 Implications & Lessons Learned

This research demonstrates a concrete and earnest example of design thinking being implemented in a higher-education setting. While it is only one full iteration of this nonlinear process, this research lays out a template to a solid beginning to enhancing student' user experience with a code critiquer, regardless of the field of study it is implemented in.

It also shows that applying HF principles and methods to the development and appraisal of an autocritiquer should be a more utilized approach. Human factors is the prevailing field when it comes to usability and user experience. Without comprehensively analysis of students' user experience with an auto critiquer, one cannot truly understand the impact the autocritiquer has in a classroom.

A prime, anecdotal example of how human factors, specifically design thinking, aided me in my research happened behind the scenes. Most of WebTA's researchers come from a computer science background, including myself, while MATLAB WebTA is implemented into an engineering course. The empathy and define stages of DT had an unexpected side effect of highlighting notions from computer science that do not necessarily transfer into the engineering space (e.g. the weight of learning code style). In other words, my background in computer science actually worked against me at first in the beginning of my research. My HF approach is ultimately what mended the disconnect of understanding what computer science says and what the engineering students required from this design.

7.2 Limitations

By implementing Design Thinking for this research, I entered a technically never-ending, recursive process. Though, that easily describes all of research in the pursuit

of knowledge. The two studies discussed in this thesis demonstrated a single cycle of Design Thinking. Study 1 (§4) practices empathizing with students and defining the their problem space through observational data. The prototyping (§5) occurred concurrently with the first study and before the second study. Study 2 (§6 tested through students' self-reported preferences and logged behavior. The A/B Testing and UEQ surveys imply that students greatly prefer the new design over the original. However, there is significant room to improve as the majority of students where neutral about their experience with WebTA, if not bored by it. While it is my experience that there is always a chance of this opinion with any education tool, it is vital to listen to students and find solutions when possible. The analysis of student behavior across three semester is not conclusive, but rather extremely suggestive of WebTA's usefulness to enhancing students' learning while programming.

Nevertheless, there is another way in which my work may fall short. My work focused almost purely on the students and improving their user experience. I did not put much attention towards course instructors. As WebTA currently stands, our researchers are the ones to prepare WebTA's back-end for each assignment students must use it for.

7.3 Future Work

My future work plans to amend this gap while simultaneously continuing the Design Thinking process for the students' UX. This includes:

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,
6. and the creation of a dashboard for professors to employ as an aid to their pedagogy.

At the time of writing this, the focus is on the points 1 - 3 to put into action first.

References

Adam, A. and Laurent, J.-P. (1980). Laura, a system to debug student programs. *artificial intelligence*, 15(1-2):75–122.

Albrant, L., Pendse, P., Brown, L. E., II, L. C. U., Sticklen, J., and P.E., M. E. J.-E. (2023a). Wip: Matlab webta, enhancing the bigger picture through human factors. In *14th Annual First-Year Engineering Experience (FYEE) Conference*, University of Tennessee in Knoxville, Tennessee. ASEE Conferences. https://peer.asee.org/44859.

Albrant, L., Pendse, P., Brown, L. E., Sticklen, J., Jarvie-Eggart, M., and Ureel, L. C. (2023b). Work-in-progress: Preliminary work introducing automated code critiques in first-year engineering matlab programming. In *2023 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, Los Alamitos, CA, USA. IEEE Computer Society.

Albrant, L., Pendse, P., Masker, D., Brown, L. E., Sticklen, J., Jarvie-Eggart, M., and

Ureel, L. C. (2023c). Work-in-progress: Python code critiquer, a machine learning approach. In *2023 IEEE Frontiers in Education Conference (FIE)*, pages 1–6.

Baddeley, A., Shiffrin, R. M., Nosofsky, R. M., and Miller, G. A. (1994). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 101(2,343-352).

Benjamin, M., Albrant, L., Pendse, P., Jarvie-Eggart, M., Sticklen, J., Brown, L. E., and Ureel II, L. C. (in press 2024). Exploring the perception and understanding of antipatterns and a code critiquer in matlab programming among instructors. In *15th Annual First-Year Engineering Experience (FYEE) Conference*.

Casey, A., Li, J., Doherty, J., Chevalier-Boisvert, M., Aslam, T., Dubrau, A., Lameed, N., Aslam, A., Garg, R., Radpour, S., et al. (2010). Mclab: an extensible compiler toolkit for matlab and related languages. In *Proceedings of the Third C* Conference on Computer Science and Software Engineering*, pages 114–117.

Deeva, G., Bogdanova, D., Serral, E., Snoeck, M., and De Weerdt, J. (2020). A review of automated feedback systems for learners: Classification framework, challenges and opportunities. *Computers & Education*, 162:104094.

Fathauer, L. and Rao, D. M. (2019). Accessibility in an educational software system: Experiences and design tips. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. IEEE.

Greifenstein, L., Obermüller, F., Wasmeier, E., Heuer, U., and Fraser, G. (2021). Effects of hints on debugging scratch programs: an empirical study with primary school teachers in training. In *Proceedings of the 16th Workshop in Primary and Secondary Computing Education*, pages 1–10.

Johnson, R. (2024). MATLAB Style Guidelines 2.0.

Johnson, S. C. (1977). *Lint, a C program checker*. Bell Telephone Laboratories Murray Hill.

Koh, J., Chai, C., Wong, B., and Hong, H.-Y. (2015). *Design Thinking for Education: Conceptions and Applications in Teaching and Learning*. The Design and Technology Association.

Oshana, R. and Kraeling, M. (2013). *Software engineering for embedded systems: Methods, practical techniques, and applications*. Newnes.

Paas, F., Van Gog, T., and Sweller, J. (2010). Cognitive load theory: New conceptualizations, specifications, and integrated research perspectives. *Educational psychology review*, 22:115–121.

Razzouk, R. and Shute, V. (2012). What is design thinking and why is it important? *Review of Educational Research*, 82(3):330–348.

Robert, J.-M. and Brangier, E. (2009). What is prospective ergonomics? a reflection and a position on the future of ergonomics. In *Ergonomics and Health Aspects*

of Work with Computers: International Conference, EHAWC 2009, Held as Part of HCI International 2009, San Diego, CA, USA, July 19-24, 2009. Proceedings, pages 162–169. Springer.

Schrepp, M., Hinderks, A., and Thomaschewski, J. (2017). Design and evaluation of a short version of the user experience questionnaire (ueq-s). *International Journal of Interactive Multimedia and Artificial Intelligence, 4 (6), 103-108.*

Tang, M. (2011). *Caesar: a social code review tool for programming education.* PhD thesis, Massachusetts Institute of Technology.

Truong, D., Roe, P., and Bancroft, P. (2005). Automated feedback for'fill in the gap'programming exercises. In *Australasian Computing Education: Proceedings of the Seventh Australasian Computing Education Conference (ACE2005)*, pages 117–126. Australian Computer Society.

Tómasdóttir, K. F., Aniche, M., and van Deursen, A. (2017). Why and how javascript developers use linters. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 578–589.

Ureel II, L. C. (2020). *Critiquing Antipatterns In Novice Code.* PhD thesis, Michigan Technological University.

Ureel II, L. C. (2021). Integrating a colony of code critiquers into webta. In *Seventh SPLICE Workshop at SIGCSE.*

Ureel II, L. C., Brown, L. E., Sticklen, J., Jarvie-Eggart, M., and Benjamin, M. (2022). Work in progress: The rica project: Rich, immediate critique of antipatterns in student code. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop*. Zenodo.

Von Wangenheim, C. G., Hauck, J. C., Demetrio, M. F., Pelle, R., da Cruz Alves, N., Barbosa, H., and Azevedo, L. F. (2018). Codemaster–automatic assessment and grading of app inventor and snap! programs. *Informatics in Education*, 17(1):117–150.

Wang, W., Zhang, C., Stahlbauer, A., Fraser, G., and Price, T. (2021). Snapcheck: Automated testing for snap! programs. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 227–233.

Woolf, B. P. (2010). *Student Modeling*, pages 267–279. Springer Berlin Heidelberg, Berlin, Heidelberg.