2020

MatlabTA: A Style Critiquer For Novice Engineering Students

Marissa L. Walther
Michigan Technological University, mlwalthe@mtu.edu

MATLABTA: A STYLE CRITIQUER FOR NOVICE ENGINEERING STUDENTS

By

Marissa L. Walther

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2020

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Science.

Department of Computer Science

Report Advisor:	*Dr. Charles Wallace*
Committee Member:	*Dr. Laura Brown*
Committee Member:	*Dr. Jon Sticklen*
Department Chair:	*Dr. Linda Ott*

Table of Contents

Acknowledgements

I would like to thank my advisor Dr. Wallace for providing me with the tools and guidance necessary to work on my project. I would also like to thank Mr. Ureel for providing me with the opportunity to work on this project, along with the Engineering Fundamentals department for supporting me and providing us with the idea of this project.

Abstract

Novice programmers, considered to be those who have yet to understand the fundamentals of programming, exist in both engineering and computing fields. Within computing, various resources exist to help novice programmers understand fundamentals and style guidelines such as WebTA, a code critique program that gives Java students feedback about their error and style issues. There is, however, a gap in automated code critique for MATLAB, a programming language that is popular in the engineering community. When it comes to MATLAB, there are not many programs that help novices understand their errors, and even fewer that help them understand style guidelines. To help assist these engineering novices, I created a program called MatlabTA. Based on feedback from Engineering Fundamentals instructors on the most common errors they encounter in student code, MatlabTA exists to give novices more intuitive feedback for a few of the most common MATLAB errors, along with providing them different style guidelines for different MATLAB antipatterns such as inconsistent tabbing and function output variable matching. This report will provide an overview of the process in developing MatlabTA, along with examples of the different outputs the application produces.

1 Background

As technology continues to play an increasingly critical role in many aspects of everyday life, more jobs are requiring that disciplines outside of Computer Science teach base programming skills. One job sector that is depending more on programming is engineering. As such, many universities are requiring their engineering students go through different classes that teach the programming skills required to solve engineering problems. At Michigan Tech, engineering students learn MATLAB in their Engineering Fundamentals classes. MATLAB was chosen since it is a programming language that was built for mathematical computation and visualization. As such, engineers and other scientists are expected to know MATLAB or similar languages in their work. MATLAB itself is an interpretive language that is based around functions and scripts to do things such as machine learning, signal processing, image processing, automate tests, and others. Those who are learning MATLAB, whether experienced in other programming languages or not, struggle with learning the fundamentals. This is due to MATLAB being different from object-oriented languages in ways such as all variables are handled by value instead of reference.

1.1 Engineering Fundamentals

The Engineering Fundamentals department at Michigan Tech has been trying to find different tools to use in their classes to help their students learn the foundations of programming. While learning the basics, a student is guided to follow traditional code styling guidelines to help with readability and to enforce good practices. Since the Engineering Fundamentals classes has a high student to teacher ratio due to every engineering student going through their program and learning MATLAB, the Engineering Fundamentals department wanted to find a tool that would help the students and alleviate the pressure on the professors to help all of them at once.

As the Engineering Fundamentals students are mostly novice programmers, they are also presented with the problem that most error messages are cryptic. These hard-to-understand messages prove to be an issue since most do not include information for how to debug a segment of code, leaving the novices confused and frustrated. A human TA, looking at student code, would be on the lookout for common novice mistakes and would provide feedback tailored to the student. This is what we want to capture in MatlabTA.

1.2 Review of MATLAB Grader

In the past, I worked with the Engineering Fundamentals department to evaluate MATLAB Grader. MATLAB Grader is an application that was developed by Mathworks, the creator of MATLAB, to allow for the automated grading of MATLAB assignments. It works by students writing individual scripts and functions to solve

different problems, and then MATLAB Grader runs them and provides a grade based on the output. To test this, we worked with Mathworks directly to create practice problems and other curriculum to give to students to see how well the application would work. During our evaluations, we found that MATLAB Grader had a good foundation for evaluating and printing out the results of a function or script. However, we found that there were a few missing functionalities that we wanted to add to the application.

- The first functionality is the presentation of error messages. MATLAB Grader, being an application created by Mathworks, presented error messages in the same way that MATLAB does. This was a problem since there were many error messages that were hard to understand by the novice.
- Another feature was that all code had to be done inline. A student could not work on their program through the MATLAB IDE and submit a file.
- Lastly, the student would not get any feedback on how they were programming. A student could hard code values, use inconsistent tabs and spaces, and mismatch variables while MATLAB Grader gives full points to the student.

After evaluating MATLAB Grader, I worked with the Engineering Fundamentals and Computer Science departments at Michigan Tech to create an application that would solve these issues.

2 MatlabTA

To help present error messages in a way that helps students debug and critique the style of their code, I worked with the Engineering Fundamentals and Computer Science departments at Michigan Tech to create MatlabTA. This new program is a Java application that runs the student's code along with providing error and style output messages. MatlabTA's main goal is to provide the students with the resources to fix their bugs while practicing good programming habits. In order to do this, MatlabTA uses different regular expressions to match error messages and code lines. When a match is made with an error message, MatlabTA takes out the information that is that is important to fixing the error and presents it in a different way to allow the student to more easily recognize what went wrong. Afterwards, if a match is made with a line of code, then MatlabTA has found a style issue that corresponds with different antipatterns. After recognizing the antipattern, MatlabTA will print out the style error and provide a link to a site that provides more examples and information about the style issue.

2.1 Common Antipatterns Recognized By MatlabTA

Antipatterns, defined as code patterns that do not follow good programming style guidelines, are a common occurrence in novice programmers' code. An example of a common antipattern that is prevalent in many different programming languages is not having spaces on both sides of an equals sign or similar operators.

```
1.

a = 1 + 2



2.

a=1 +2
```

The above code shows an example of good versus bad style in programming. The first example is the good code as there are spaces around each operator and assignment symbol. The second example shows bad programming style as there are not spaces on both sides of the equals and plus signs. In programming, it is important to have these spaces for readability, consistency, and professionality. When students use good programming style practices, it becomes easier for the student and for anyone else who is reading the code to help reveal bugs and other problems that exist in the code. It also allows the students to start practicing a habit that they will have to use in a professional setting.

MatlabTA will also recognize the antipattern of a line of code not being tabbed enough or if it is tabbed too far. These antipatterns are common with other programming languages as well such as Java and C. However, MATLAB is different from Java and other object-oriented languages since it handles all variables by value instead of by reference. Due to this, when I met with a few professors in the Engineering Fundamentals department to choose one common antipattern to evaluate that was specific to MATLAB, I chose the matching the number of variables a function outputs to the number of variables that are actually being stored. An example of this is as follows:

```
width = foo()

function [width, distance] = foo()
        width = 1
        distance = 2
end
```

Function foo is storing values in two different variables, width and distance. The call to foo, however, is only storing one of those values: width, resulting in the information stored in distance to be lost. MATLAB sees this as valid code and will run the above without throwing an error message. In particular, this becomes a warning sign within novice code that the novice might not understand MATLAB or the assignment that they are working on. As such, we recognize this error to provide the student with an error message that tells them how many variables they are storing, along with how many the function is outputting. In providing the students with this message, the students are able to reevaluate the amount of variables that are in the function and that they are storing.

2.2 Similarity Check

After MatlabTA compares the number of output variables to the number of variables stored, it then runs a similarity check for each variable. The similarity check compares the distance between each data-storing variable to each output variable. In particular, the similarity check is run to determine the antipattern of whether or not the data-storing variables are in the right order when compared to the list of output variables. It also looks to see if the data-storing variables follow a similar naming scheme to that of the output variable it matches with. If the similarity check returns that there is no match between the two variable names, MatlabTA then prints out an error message saying that the two do not match. After printing the message, MatlabTA will then compare the naming schemes of the remaining variable pairs. If a variable passes the similarity check with the remaining variables, MatlabTA outputs a message prompting the user to check that their variables are listed in the correct order. An example is shown below:

```
[distanceA, distanceB] = foo()


function [distance, width] = foo()
end
```

In this case, the function is outputting two variables: distance and width. However, the names of the variables storing the results are named distanceA and distanceB. This implies that either the function is not returning the correct information, or that the information being stored is misinterpreted by the programmer. As such, MatlabTA will display an error message saying that distanceB does not closely match the variable name width, prompting the user to reevaluate the relationship between the two variables.

It is also important to note that we present this information to try and help the student visualize what is happening in their program. Not every variable will be mismatched if a note is thrown, however it is important to help novices recognize the different things that are happening in their program.

2.3 Website

If a check for an antipattern is detected, MatlabTA will print out an error message that points to the exact style guideline that is being breached, along with providing a link to a website. I created this website to provide more information and examples for the different style guidelines that MatlabTA recognizes. In providing this information, the programmer can see these examples and use them to change their program into "good code".

2.4 Error Messages

Another feature of MatlabTA is that it reads in error messages that are returned by the MATLAB compiler, and provides a more novice friendly alternative to help the novice debug. Novice friendly alternatives to error messages are important, because when a novice is learning to program, they often get overwhelmed and confused by how cryptic various compiler messages seem to be. MatlabTA aims to reduce the ambiguity by extracting the important information from the original message and placing it inside a more novice friendly message. By providing the information in this way, the programmer given more ideas on how to fix their bugs and get their program working. A common error message that MATLAB prints out is "Array indices must be positive integers or

logical values.” This message is printed out when a user is trying to access an array at index 0. When MatlabTA recognizes with this error message, it prints out a new message that is aimed to help the novice fix their mistake. The new message is “In MATLAB, arrays start at 1. Make sure that you aren’t trying to get the value at index 0”. By providing this feedback, it is our hope that students will be able to make connections between MATLAB’s error messages and what they have to do to debug, allowing them to grow as programmers.

3 Regex

In order to recognize these various antipatterns and error messages, MatlabTA utilizes different regular expression statements (regex). These regex statements were crafted in a way that allows for the extraction of useful information such as variable names and assignments. Initially, the regex that MatlabTA used were very long and required almost perfect matching for a line to match. As I learned more about regex, they became more general to reduce complexity. An example of a regex before and after rework:

Before:

```
\bUndefined function or variable\b\s'([a-zA-Z]*)'.
```

After:

```
Undefined function or variable '(.*)'.
```

3.1 MParser

One of the applications that I looked at integrating into MatlabTA was a project called MParser. MParser reads in MATLAB code and return a full abstract syntax tree. Also included was the ability to recognize certain MATLAB errors such as a string variable missing an ending quotation mark. Additionally, I modified MParser to include the line numbers along with outputting that information to a file that can be uploaded to MatlabTA. This will help to provide more information about errors that occurred in their code. Currently, MParser can only return the correct line numbers of certain errors if there are no empty lines in the file. However, MatlabTA can run without this abstract syntax tree file if this poses too big of a problem.

4 Implementation

MatlabTA is a Java application with a JavaFX UI. It works by asking the user to choose a MATLAB file and if possible a syntax tree file that was output by the modified MParser. The user then inputs the name of the class and name of the function that should be ran by the MATLAB API. Once the user hits run, MatlabTA calls the MATLAB API that is included with every standard download of MATLAB, and runs the code from the specified class name and method. While the API is running, all the standard MATLAB output is shown in the console. If the API encounters an error, the original compiler message will be output at that point, after which a novice friendly message will be printed out with the necessary information from the original message. If a syntax tree file was also included, MatlabTA will also output the errors found from the abstract syntax tree at this point.

After the MATLAB API is done running, MatlabTA will run its style check on the original MATLAB code. Before checking each line for its style, MatlabTA will first go through the entire MATLAB file and create a dictionary entry for every function, where the key is the function name and the value is a list of the output variables for that function. After the dictionary is created, MatlabTA will run line-by-line through the code and check the lines against various regex statements. If a match is made between a regex statement and the line of code, then a style error has occurred. MatlabTA will then output the erroneous line with a customized error message that contains why the line violated the style guidelines. In addition, MatlabTA will then also print out a URL for a custom-built style website created from html and css files, allowing the user to have more information about that style guideline if they want it. After MatlabTA has printed out every line in the file, the UI changes to show that the job is complete allowing the user to run MatlabTA again. The similarity check functionality is done using the Apache Commons Text library. Currently, MatlabTA and the style website are both ran locally..

5 Future Work

In the future, the goal is to add more style and error message regex along with fully integrating the application with WebTA. The integration would allow professors to use MatlabTA in their classes and connect the results to Canvas, along with hosting the application on the web instead of being locally ran. In doing this, MatlabTA would also support the grading of assignments based on criteria given by the professors. Additionally, MatlabTA should use the results from MParser to recognize more syntax errors from the syntax tree, along with recognizing when there is good code that is misplaced in the file. For example, if code is correct but is not within a function and instead is just sitting in the MATLAB file, the goal would be to detect this and inform the student. Another way to use the syntax tree would to be to compute the code complexity using methods such as cyclomatic complexity. Using these methods, we can point out if students are using logic statements that make the program too complicated. If a statement is recognized to be too complicated, then we can point out this issue since the more complicated a program is the harder it is to read and test. With the style website, the goal is to add more examples and pages to allow the students to explore the different style issues and learn about different antipatterns. The website would also have a functionality where a student could interact with the different examples in a game like environment. The student could be given a prompt to write out a statement, and then go ahead and write out MATLAB code for that statement. The game could then check the style of the student's statement and point out if any style errors were found. This could allow the students to become more engaged with the information MatlabTA is trying to teach them.

6 Reference List

Fangohr H. (2004) A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering. In: Bubak M., van Albada G.D., Sloot P.M.A., Dongarra J. (eds) Computational Science - ICCS 2004. ICCS 2004. Lecture Notes in Computer Science, vol 3039. Springer, Berlin, Heidelberg

Jason Nicholson (2020). Matlab Style Guidelines Cheat Sheet (https://www.mathworks.com/matlabcentral/fileexchange/45047-matlab-style-guidelines-cheat-sheet), MATLAB Central File Exchange.

Kang, Hyeonsu, and Philip J. Guo (2017). "Omnicode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations" Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology UIST 17.

Lee, M. J., & Ko, A. J. (2011). "Personifying programming tool feedback improves novice programmers' learning" Proceedings of the seventh international workshop on Computing education research (pp. 109-116).

Ureel II, L. C., & Wallace, C. (2019, February). Automated Critique of Early Programming Antipatterns. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (pp. 738-744).

Ureel II, L. C., & Wallace, C. R. (2018). Webta: Online code critique and assignment feedback. In Proceedings of the 49th acm technical symposium on computer science education (pp. 1111–1111).

Watson, Christopher, Frederick W. B. Li, and Jamie L. Godwin (2012). "BlueFix: Using Crowd-Sourced Feedback to Support Programming Students in Error Diagnosis and Repair." Advances in Web-Based Learning - ICWL 2012 Lecture Notes in Computer Science (pp. 228–39).

A Copyright Documentation

MParser is from Github. It is licensed for modification and public use under the MIT license. Please see below for full citation and attribution information.

“MParser” by Alexander Luzgarev, github.com – MIT. Licensed under MIT via github - https://github.com/mahalex/MParser