**Metric Converter Testing**

by

Nathan Collins

Pau Ferrer Collado

This project is submitted to Dr. Tajmilur Rahman in

partial fulfillment for the course SOFT310 – Software Testing

Gannon University

Erie, Pennsylvania 16541

Fall 2024

**Table of Contents**

[***1.***](#_heading=h.z337ya)***Introduction 1***

[***2.***](#_heading=h.32hioqz)***Requirements 1***

[**5.1**](#_heading=h.1hmsyys) **Requirements Collection 1**

[**3.1**](#_heading=h.41mghml) **User Story Development 1**

[**3.2**](#_heading=h.2grqrue) **User Classes and Characteristics 1**

[**3.3**](#_heading=h.vx1227) **Use-Case Models 2**

[***3.***](#_heading=h.3fwokq0)***Proposed Features 2***

[**3.1**](#_heading=h.1v1yuxt) **Key System Feature 1 3**

[**3.2**](#_heading=h.4f1mdlm) **Key System Feature 2 3**

[**3.3**](#_heading=h.2u6wntf) **Work Distribution 3**

[***4.***](#_heading=h.19c6y18)***Implementation Constraints 3***

[***5.***](#_heading=h.3tbugp1)***Design 4***

[**5.1**](#_heading=h.28h4qwu) **Architectural and Component-Level Design 4**

[**5.2**](#_heading=h.nmf14n) **User Interface Design 4**

[***2.***](#_heading=h.37m2jsg)***Testing 5***

[***3.***](#_heading=h.1mrcu09)***Conclusion 5***

[***4.***](#_heading=h.46r0co2)***Bibliography 6***

**Acknowledgements**

This project was done thanks to the support and instruction of Professor Tajmilur Rahman.

1. **Introduction**
   1. **Overview of the Project**

For this project we took a metric converter class from [BelloJacob](https://github.com/Bellojacob/Metric-Converter) to convert lengths and used their base logic to make it more usable, as well as adding new features. The new features added include conversions from volume, weight, and time, as well as more units to convert from. Throughout development we performed several types of Whitebox and Blackbox testing, and additionally created a UI to test the implementation. This converter class created and tested could be useful for many people to implement in desktop applications, websites, mobile apps, and more. The main purpose being to include useful features, such as lots of units, while still being straightforward to implement.

* 1. **Overview of Contribution**

The main improvements that we focused this project on are quite simple including:

* Making the software easy to implement
* Increasing usefulness by adding lots of units to convert to/from
* Incorporating test driven development to ensure a quality product

1. **Requirements**

The requirements for this project include adding new features to preexisting source code, performing multiple types of Whitebox and Blackbox testing throughout development, testing the implementation of your software with a UI, and lastly setting up an environment to perform Regression tests as your software undergoes changes.

## Planning and Requirements Gathering

5.1.1 Add additional units to convert to/from

This new feature is an addition to the preexisting method to convert length, to which we added four new length units to cover the most common conversions. Adding new units to this software makes it more useful for implementation, however it did raise issues. As you add new units, the number of possible conversions is n^2, where n is the number of units. The conversion method was then improved to solve this issue, by instead of performing direct conversions, it converts to a base unit, then from that base unit to the target unit, resulting in n \* 2 conversion needed. Below is a figure showing how we included new units to our software:

A black screen with text

Description automatically generated

**Figure 1. Java Enumeration for Length Units**

5.1.2 Add functionality to convert weight

The weight conversion is a new feature that includes the most important weight units to convert to/from, and is designed to follow the logic of the improved length conversion method. This new feature will have a separate method and an Enumeration for its units. The figure below shows how you can utilize this new feature in implementation.



**Figure 2. Convert Weight Call**

* + 1. Add functionality to convert volume

The volume conversion is a new feature that includes the most important volume units to convert to/from, and is designed to follow the logic of the improved length conversion method. This new feature will have a separate method and an Enumeration for its units. It can be called following the same structure as shown above.

* + 1. Add functionality to convert time

The time conversion is a new feature that includes the most important time units to convert to/from, and is designed to follow the logic of the improved length conversion method. This new feature will have a separate method and an Enumeration for its units. It can be called following the same structure as shown above.

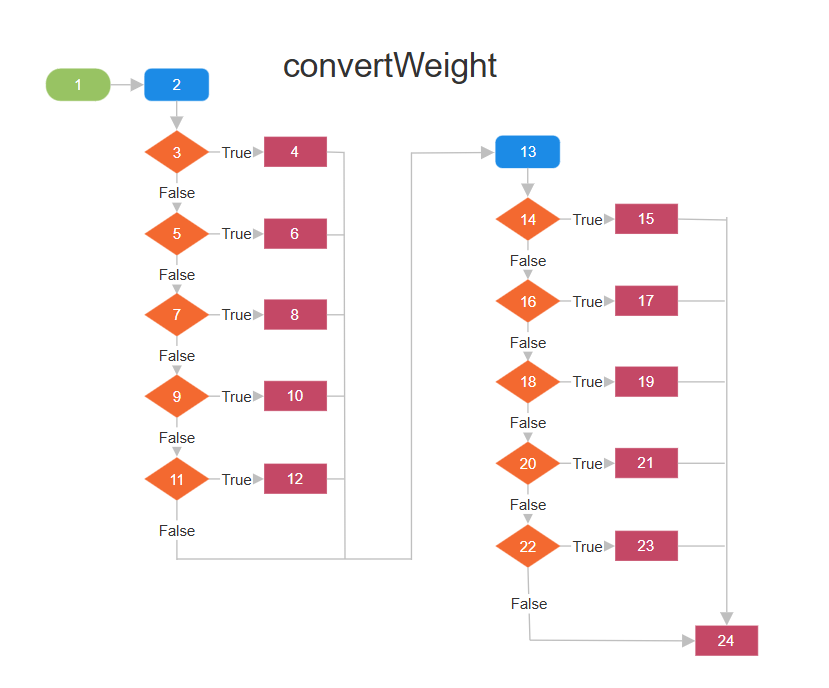
## 

## 3 Test Plans

For this development there are several tests planned to ensure a quality product. The tests planned include unit tests to check basic functionality of our methods, control-flow tests to cover all nodes and branches, boundary value analysis, integration tests, usability tests, and lastly regression tests.

## 3.1 Control-Flow: Node Coverage

We planned to achieve a 100% node coverage for the new features/methods. For the new methods, you simply input a number as a double, the unit you are converting from, and the unit you are converting to. Below I will display the control flow diagram for the convert weight method.



After analyzing the source code for the method, we found 25 possible paths:

*A number in a row

Description automatically generated with medium confidenceA number grid with black numbers

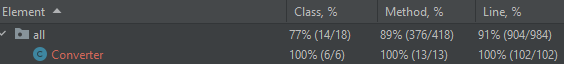
Description automatically generated with medium confidence*

As there are a lot of test cases needed, I will showcase a few below, but all test cases can be found [here](https://github.com/NathanCollins4551/MetricConverterTestingProject/tree/main/src/test/java).

*A computer screen shot of a program

Description automatically generated*

When running the Test Convert Path Coverage file it convers 100% of the Converter class:



## 3.2 Control-Flow: Branch Coverage

In this convert weight method, there are two switch case statements that contain a collective total of 10 conditional branches. To achieve full branch coverage each condition must be evaluated to true and false. Due to the nature of our method, it was necessary for each condition to evaluate to true and false to reach every node and achieve full node coverage. Therefore, we have already achieved full branch coverage through the tests previously used.

A diagram of a graph

Description automatically generated with medium confidence

## A diagram of a diagram Description automatically generated with medium confidence3.3 Blackbox: BVT Coverage

We finished our testing by performing boundary value analysis to see how our software held up against extreme and unexpected values. To achieve this, we used multiple types of BVA including normal BVA, robust BVA, and robust worst case testing. We chose BVA to see if our program is still accurate under extreme conditions and if our program threw any error under extreme values. Input values included are many but contain values like, 0, -1, Interger.MAX\_VALUE, Integer.MAX\_VALUE \* 12, etc. and using Junit assert tests we could see if our program produced the correct result to a certain degree of accuracy.

After running the tests there were numerous failed tests, showing values that were simply inaccurate due to the large values added. Simply if you try to convert 1,475,894,374 miles to mm your answer will not be accurate to 0.01. Given more time we would ideally examine each test and identify how accurate we require the results to be, then use the BVA tests to see which conversions meet our accuracy standards and which need improved on, then make the necessary improvements.

## 3.4 Regression Testing

For regression testing we set up a pipeline using Jenkins to build and test our project. We have it setup to run all our test classes every time we build the project, this ensure that any new developments to the code will have to pass the tests for the build to succeed.

We also configure the Jenkins file to publish test reports as shown in the example below:

A screenshot of a computer

Description automatically generated

## Testing Through Continuous Build Pipeline

We configured the build pipeline first fetch the latest source code, then attempt to build our project, lastly, it performs all our test classes and publishes the test report.

**4 Work-Load Distribution**

The workload was evenly distributed between us for the project. Early in the project timeline, we worked together to design the methods we wanted to include and plan out how we wanted to incorporate our new features. Small meetings occurred between us in person between classes where we planned out our testing and decided who should work on what tasks. The workload can be represented by the table below:

|  |  |  |
| --- | --- | --- |
| Contributor(s): | Task: | Notes: |
| Nathan C / Pau F | Create Converter Class | Worked together to design and implement the features of our converter class throughout test driven development |
| Pau F | Unit Testing | Wrote unit tests to ensure our methods were functioning properly |
| Nathan C | Control Flow Testing | Created control flow diagrams, performed control flow node testing and branch testing to ensure %100 coverage |
| Pau F | Boundary Value Analysis | Created normal BVA, robust BVA, and worst case BVA tests to check our program for error or inaccuracies with different inputs. |
| Nathan C | Integration Testing | Created UI to test integrating our software into application |
| Nathan C | Regression Testing | Configured Jenkins to preform tests on every build and publish result |
| Pau F | Presentation | Created professional presentation to showcase our project |

## 4.1 Component 1 – Adding Feature

Nathan led the development of adding new features by coming up with a design for our methods, and how we should implement our new features. I created some of the methods and Pau create some of the methods. We shared lots of brainstorming and thinking ahead about what tests to perform as we develop. The main issue we found was how to add lots of units without having to code 20+ conversions, instead we came up with a solution involving converting to a base unit, then convert to the target unit resulting in much less direct conversions needed, which translated into being able to add more units.

## 4.2 Component 2 – Testing

Pau took an interest in the testing and led the testing by performing unit tests early on to validate that our methods are functioning correctly. Nathan performed control flow testing involving drawing control flow diagrams and finding paths to all nodes and tests for all branches. Pau performed boundary value analysis to identify any possible defects and errors in the software by testing inputs in both extremes.

## 4.3 Component 3 - Implementation

Nathan led development for the implementation by developing a UI using java swing features. We tested the implementation of our software by making a desktop application that incorporates our conversion features.

**Implementation Constraints**

The main limitation we faced was being restricted to the java programming language. We would have loved to write this code and test it using a language more comfortable with such as c# or creating our converter in JavaScript with the intent of being implemented easily in a website. Aside from that, busy schedules made it difficult to explore different more creative features and testing methods that would have otherwise been of interest. One area we needed more time to explore was the BVA testing. Tests were performed and showed potential for improvement; however, we did not have the time to make all the improvements to our code. There were several conversions that when put under extremely large inputs, the result was not super accurate. With more time we could find a way to make our program handle these inputs better and yield more accurate results. All in all, there were not many constraints for the project.

# Conclusion

To conclude, our metric converter project allowed us to explore and practice test driven development. We used multiple types of testing to create a quality product that, without TDD we would struggle to produce. The main types of testing used included: unit testing, control flow testing, BVA testing, implementation testing, and regression testing. Our product was then implemented into a test desktop application to see how we can implement it.

# References

|  |  |
| --- | --- |
| [1]  [2]  [3]  [4]  [5] | [Metric Converter Project Github](https://github.com/NathanCollins4551/MetricConverterTestingProject/tree/main/src/test/java)  [Metric Converter Implementation Github](https://github.com/NathanCollins4551/MetricConverterImplementation)  <https://jenkins.ihk.network/job/SOFT310%20-TEAM_C_METRIC_CONVERTER/>  <https://www.geeksforgeeks.org/control-flow-software-testing/>  <https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html> |