Design Methodology

Problem Statement

Purpose: To develop a text-based user interface for a unit conversion program that enables users to convert between different units of measurement, including imperial and metric systems. The program will leverage a user-defined function created in a previous module to perform the actual unit conversions. This project aims to enhance user interaction and streamline the conversion process by providing a clear and intuitive interface.

Requirements to be fulfilled:

1. **Text-Based User Interface:**

A text-based interface that prompts users to input a number for conversion and select the conversion type from a menu, should be developed. The interface should be intuitive and easy to navigate, allowing users to interact with menus and prompts efficiently.

2. **User Input:**

Prompt the user to enter the number they want to convert, which can be a whole or decimal number. A menu for users to select the type of conversion they desire should be provided. Also, ensure error handling mechanisms are in place to handle cases where users make mistakes, such as selecting non-existent menu items or leaving inputs blank. Users should be prompted to re-enter correct inputs in such cases.

3. **Conversion Function Integration:**

Integrate the user-defined function, previously created in module 2, into the program. Upon user input, call the `myunitconv()` function to perform the appropriate unit conversion based on the selected conversion type.

4. **Output Formatting:**

Output the converted number along with its unit in a clear and coherent manner on the Command Window. Ensure the output is neatly formatted for easy interpretation by the user.

Input/Output Format:

Input: Number to be converted (whole or decimal). Selection of conversion type from a menu (numerical or character entry).

Output: Converted number along with its unit, presented in a clear and coherent manner on the Command Window.

Usability Considerations:

- The program's functionality and usability are equally important.
- It should streamline the conversion process, making it simple and intuitive for users to input values and obtain accurate conversions.
- Error handling mechanisms should ensure a smooth user experience by guiding users to correct any input mistakes promptly.
- By addressing these requirements, the program will provide users with a user-friendly interface to perform unit conversions effectively, catering to both functionality and usability aspects.

Algorithm Design (Flow Charts)

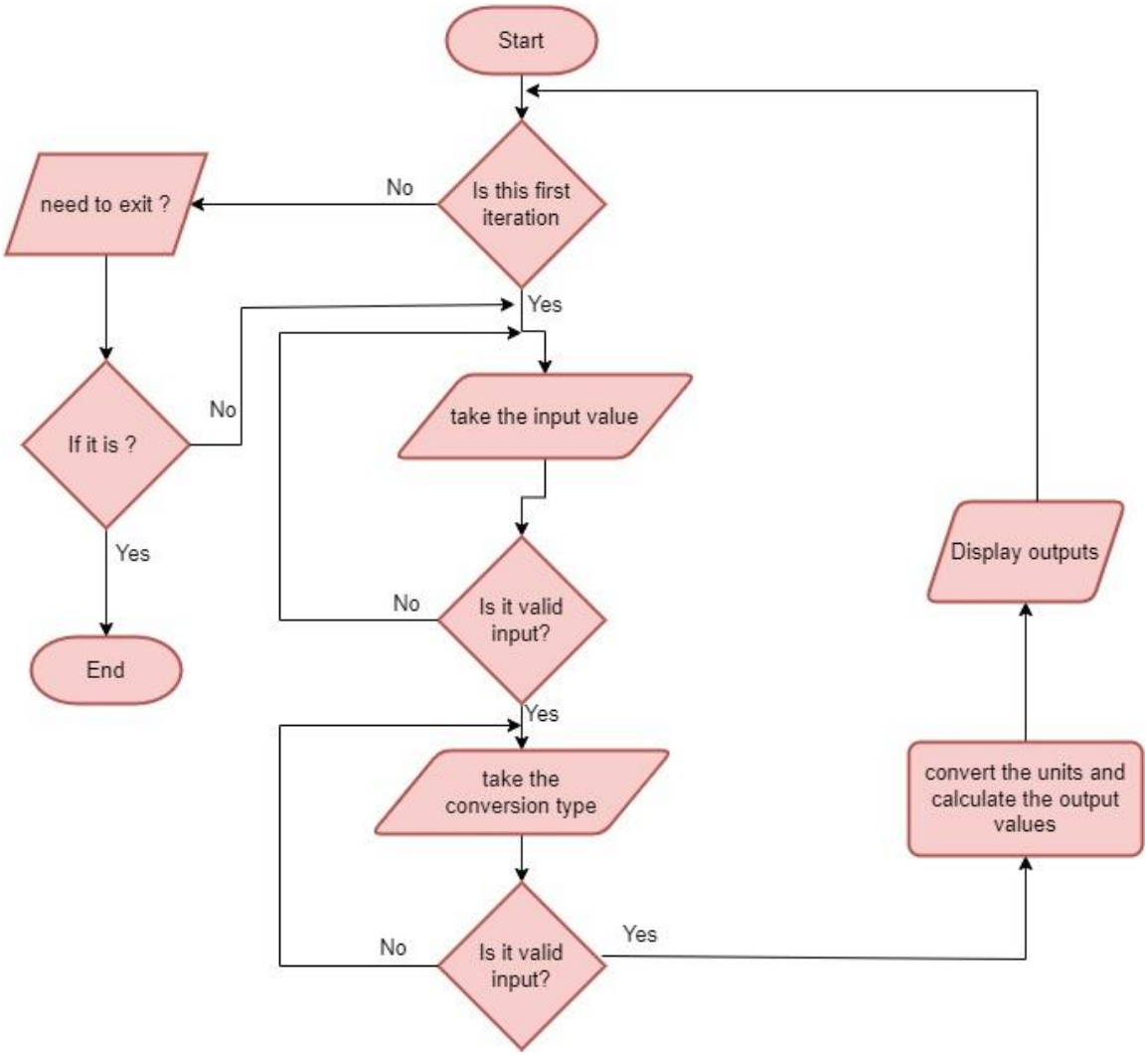

Figure 1: Flow-chart diagram

Testing and Evaluation

Behaviour of the Program:

Table I: Testing for Unit Converter

Type	Input Value	Output Value	Expected Answer	Match?
Temperature				
$^oF \rightarrow {}^oC$	100	37.7778 C	37.778	Yes
$^oC \rightarrow {}^oF$	15	59 F	59	Yes
Length				
cm → inches	20	7.874 in	7.874	Yes
inches → cm	5	12.7 cm	12.7	Yes
metres → feet	24	78.7392 ft	78.74	Yes
feet → metres	3	0.91441 m	0.9144	Yes
km → miles	50	31.0694 mi	31.069	Yes
miles → km	30	48.279 km	48.28	Yes
Mass				
grams → ounces	7	0.24692 oz	0.247	Yes
ounces → grams	2	56.699 gr	56.699	Yes
kg → pounds	38	83.7748 lb	83.776	Yes
pounds → kg	8	3.6288 kg	3.629	Yes
tonnes → tons	4	3.9366 LT	3.937	Yes
tons → tonnes	15	15.2415 t	15.242	Yes

Error Handling:

Test for erroneous inputs and how the program handles these is shown in the Figure 3 & 4 below.

```
>> main
Welcome to the unit conversion system!
Please enter the value: lk
Invalid input. Please enter a numeric value.
Please enter the value:
```

Figure 2: Error Handling: Main

```
-----Unit Conversion Types-----
1  --> from Celsius to Fahrenheit
2  --> from Fahrenheit to Celsius
3  --> from Centimeter to Inch
4  --> from Inch to Centimeter
5  --> from Meter to Foot
6  --> from Foot to Meter
7  --> from Kilometer to Mile
8  --> from Mile to Kilometer
9  --> from Gram to Ounce
10 --> from Ounce to Gram
11 --> from Kilogram to Pound
12 --> from Pound to Kilogram
13 --> from Metric Tonne to Imperial Ton
14 --> from Imperial Ton to Metric Tonne
Please enter the corresponding conversion type number: 16
Invalid input. Please enter a number between 1 and 14 according to the above table.
Please enter the corresponding conversion type number:
```

Figure 3: Error Handling

User Interface and it's Behavior:

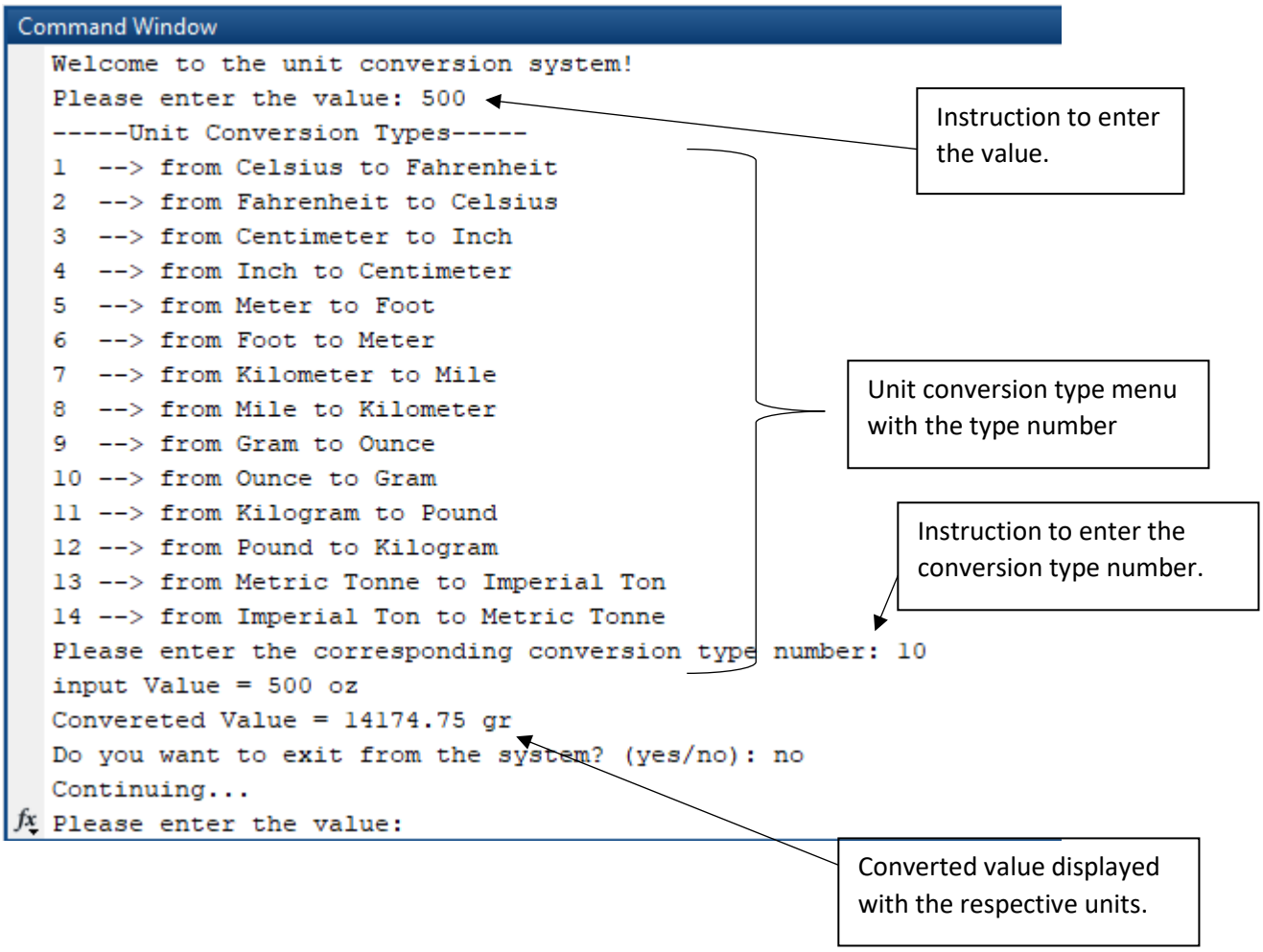

Figure 4: Text based User-Interface

Discussion:

Considering the given strengths & weaknesses below we can identify the success and the things we can further improve.

Briefing the main key **strengths** which make this stand out **User-Friendly Interface:** which this program provides a clear and concise interface for users to interact with and it guides the user through each step of the unit conversion process, displaying conversion options and prompting for input in a structured manner. Also, **Robust Input Validation:** this program incorporates robust input validation mechanisms to ensure that the user provides valid inputs for values and conversion types. This helps prevent errors and ensures smooth execution of the conversion process. Another is the **Modular Structure:** which allows the program to organize into separate functions for different tasks, such as handling termination, getting input values, validating conversion types, and performing unit conversion. This enhances the readability, maintainability, and reusability of the code. Also, the program includes **Error Handling** mechanisms to address invalid user inputs and unexpected scenarios, such as entering non-numeric values or selecting an invalid conversion type. This improves the overall reliability of the program.

Briefing on the **weaknesses** of this program, it has **Limited Conversion Options:** Currently this only supports a fixed set of unit conversion types (from Celsius to Fahrenheit, etc.), limiting its versatility. But additional conversion options could enhance its utility for a broader range of applications. Also, this program has **Lack of Unit Tests:** this lacks comprehensive unit tests to validate its functionality under different scenarios and edge cases. But implementing unit tests could help identify and address potential bugs or inconsistencies in the code.

Also further discussing the **improvement opportunities**, we can consider the given options below.

1. **Expand Conversion Options:** Th program can be enhanced by adding support for additional unit conversion types based on user feedback or specific requirements. This could involve incorporating conversions for more measurement units or specialized domains.

2. **Implement Automated Testing:** Comprehensive unit tests can be developed using testing frameworks like MATLAB's Unit Test Framework or external libraries like *MATLAB xUnit* Test Framework. These tests should cover various input scenarios and edge cases to ensure the correctness and robustness of the program.

All in all, as the conclusion, while this current program exhibits several strengths in terms of user interface, input validation, and modularity, there are also areas for improvement such as expanding conversion options, enhancing testing coverage, refactoring for code reusability, and exploring parallelization for improved performance. With further development and refinement, this program can become a more versatile and robust tool for unit conversion tasks.