## 2.5 Testing

### 2.5.1 Iterative Test Plan

To ensure that my program is working properly I will use a range of data to verify that the program interprets the data correctly and processes it without any errors.

To ensure that the tests are carried out properly I will need to use data that mimics every possible input the user can access, such as a range of numbers and characters.

This is required because I need to test that the program correctly processes the data that is supplied to it. All the possible inputs are needed so that a bug doesn’t go unnoticed through the iterative development process. This could cause much larger problems further along the development cycle.

I will be testing my application through the use of unit tests. Each function that I implement will have a range of unit tests ensuring that it works correctly. The unit tests will take a variety of inputs that cover any possible scenario that the user can do. Therefore, the unit tests will ensure that the application responds properly to every possible input for every function.

An example unit test for the addition / subtraction function:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | Reason | Expected | Actual | Pass/Fail and Actions |
| 1 + 2 | Adds two positive numbers | 3 |  |  |
| 1 + + 2 | Two addition operators should result in an error. | Error Message |  |  |
| 1 + - 2 | Adds/subtracts an negative and positive number. | -1 |  |  |
| -1 - 2 | Subtracting 2 negative numbers | -3 |  |  |
| -1 + - 2 | Subtracting 2 negative numbers with an addition operator | -3 |  |  |
| 1 - - 2 | Two subtraction operators should result in addition. | 3 |  |  |

### 2.5.2 Testing of Final Implementation

The final implementation will be tested with a white box method. This will be done by myself as I will be most knowledgeable about the program and so I will be most qualified to test every aspect of the program.

This is an important testing method to use because it will ensure that every function of the programs works as intended. This may not happen with black box testing because the people testing the program are not familiar with it and therefore they could miss errors.

Therefore, white box testing is very important as the developer is most likely to recognise every error that occurs so the final implementation will be as bug free as possible.

I will be testing the final implementation’s user interface using black box testing. I will create a survey that has brief instructions of how to use the program and ask people who have little knowledge of the system to follow the survey’s instructions to use the program.

I can use the survey to track how easily people can interact with my program with minimal guidance. This should test how accessible and user friendly my final implementation is. The use of a survey will also make it easy to get feedback on the user interface to make any potential improvements.

The use of black box testing will provide an unbiased perspective on the app. This will generate valuable feedback that I would not otherwise be able to get using other testing methods.

The final method that will be used is destructive testing. In this testing method, the developer tries to break the system when in full use.

It’s important to use this method of testing with the final implementation because the final product should not have any fatal bugs that lead to a full crash. Destructive testing will reveal any of these potential fatal errors so safe guards can be put in place.

This type of testing will therefore ensure that user is provided with the cleanest possible experience and never experiences any fatal errors that crashes the application and interrupts the user.

In conclusion, I will use a combination of white, black and destructive testing because this will provide the largest range of different tests which should cover any potential errors that will affect the user’s experience.