Mass Spectrometer Interface

Desktop Application for Reading Instrument Data

Cousins Photosynthesis Lab in the School of Biological Sciences at WSU



**Team Linnaea Borealis**

****

Kyler Kupp, Erik Holtrop

CptS 423 Software Design Project II

Spring 2025

**TABLE OF CONTENTS**

[I. Introduction 2](#_Toc1748030534)

[II. Team Members & Bios 3](#_Toc1611070759)

[III. Project Requirements Specification 4](#_Toc151812943)

[III.1. Project Stakeholders 4](#_Toc1466111495)

[III.2. Use Cases 5](#_Toc873675628)

[III.3. Functional Requirements 9](#_Toc878464889)

[III.4. Non-Functional Requirements 11](#_Toc1562166835)

[IV. Software Design 12](#_Toc528469208)

[IV.1. Architecture Design 12](#_Toc1171064743)

[IV.2. Data Design 19](#_Toc1248947056)

[IV.3. User Interface Design 20](#_Toc857914674)

[V. Test Case Specifications and Results 22](#_Toc1895135968)

[V.1. Testing Overview 22](#_Toc1235395042)

[V.2. Environment Requirements 24](#_Toc866378835)

[V.3. Test Results 25](#_Toc1901370507)

[VI. Projects and Tools Used 26](#_Toc1155370079)

[VII. Description of Final Prototype 27](#_Toc1554996843)

[VIII. Product Delivery Status 30](#_Toc617386600)

[IX. Conclusions and Future Work 30](#_Toc1879701270)

[IX.1. Limitations and Recommendations 30](#_Toc517355230)

[IX.2. Future Work 30](#_Toc1144578347)

[X. Appendix A - User Stories 30](#_Toc1964287008)

[XI. Appendix B - Traceability Matrix 32](#_Toc689290077)

# Introduction

Plants play a crucial role in regulating the Earth's atmosphere by absorbing carbon dioxide (CO₂) and releasing oxygen (O₂). Understanding the factors influencing plant respiration rates is essential for addressing both past evolutionary patterns and future agricultural advancements. At Washington State University’s Cousins Photosynthesis Lab, researchers use gas chromatography mass spectrometry (GC-MS) to measure plant respiration rates through precise monitoring of gas concentrations. However, the proprietary software previously used generates excessive data and offers limited data processing or exporting, making analysis inefficient.

Our client is the Washington State University (WSU) Cousins Photosynthesis Lab. The lead researcher in this lab is Dr. Asaph Cousins, who has also acted as our primary point of contact.

The Mass Spectrometer Interface consists of four primary modules, each designed to address specific aspects of data analysis. Modules 3 and 4 are the focus of this report and our attention, since modules 1 and 2 were complete products when we began work. We are still responsible for support for modules 1 and 2. This support includes patching bugs and integrating other modules’ features where applicable.

Module 3 expands the functionality of the application by enabling isotope ratio analysis and plotting. It calculates the proportion of specific isotopes, such as ¹³C¹⁸O₂, and visualizes their trends over time. Additionally, it introduces a real-time derivative calculation of these ratios, providing researchers with insights into gas exchange dynamics during experiments. The graphical tools in this module allow users to dynamically manipulate scales and isolate key data ranges for precise analysis.

Module 4 addresses a critical need for compatibility with additional instruments. By converting raw data from a second mass spectrometer into a format that Modules 1–3 can process, this module ensures that data streams from multiple devices can seamlessly integrate into the lab's workflow. This enhancement not only improves interoperability but also future-proofs the software for the addition of new instruments.

This document provides a comprehensive overview of the **Mass Spectrometer Interface** project, summarizing its progress and technical details. It serves to detail our engineering efforts, including the design, implementation, and testing of key system components. By outlining our current progress, challenges, and solutions, the report captures the essence of our work on the alpha prototype, offering insights into the architecture, functionality, and stakeholder-focused goals of the project. This document also sets the foundation for future development by presenting our findings, demonstrating the prototype, and identifying areas for refinement based on feedback and testing outcomes.

# Team Members & Bios

Our team brings together a diverse set of skills and experiences, ranging from software engineering to data visualization and algorithm development. Each member contributes unique expertise and a shared commitment to delivering a robust and efficient data collection and visualization system.

Erik Holtrop is a dedicated computer science and mathematics student at Washington State University, where he is set to graduate in 2025. His academic journey has honed his skills in software engineering, algorithms, and mathematical computing. Erik’s technical expertise includes Python, C/C++, C#, SQL, and Haskell, along with experience in Python libraries such as MatPlotLib, PyEDA, and PyQT. His responsibilities include managing data structures, debugging, project management, and integrating mathematical algorithms into the system.

Kyler Kupp is a driven computer science student at Washington State University, aiming to complete his degree in 2025. He combines academic excellence with hands-on industry experience, having worked as a Software Engineer Intern at Monson Fruit Company. In this role, Kyler developed monitoring programs, automated reporting systems, and contributed to software managing critical production processes. His technical toolkit includes proficiency in Python, C#, SQL, and JavaScript, with familiarity in libraries like PyEDA and SFML. Kyler is responsible for UI/UX, stakeholder relations, and documentation oversight.

# Project Requirements Specification

When plants breathe, they take carbon dioxide (CO2) out of the air and replace it with oxygen (O2). Determining what affects plants’ respiration rate, or their breathing rate, is incredibly valuable data. These factors point backwards in time, reflecting causes for evolutionary trends, and forwards in time, providing opportunities to improve agriculture. We can use a mass spectrometer to measure this breathing rate. The mass spectrometer measures volumes of gasses, enabling us to see the flow of carbon dioxide and oxygen, and even different isotopes.

The Cousins Photosynthesis Lab in the School of Biological Sciences at Washington State University uses one of these mass spectrometers. These instruments are complicated devices, requiring complex calculations for calibration. The lab uses proprietary software from the mass spectrometer’s manufacturer, but that software outputs massive amounts of data over the course of a multi-hour lab, most of which isn’t needed. This problem has been partially solved with the creation of a Python desktop application, but this application is not perfect. Our task is to improve this application. This application currently faces small bugs, and only works for one instrument. The application is also in process of a UI upgrade. There’s also a few non-spectrometer instruments in the lab that are provide similar data, that would be easier to use if their data-streams were combined.

## Project Stakeholders

Our development process incorporates feedback from key stakeholders, including Dr. Asaph Cousins and the researchers in his lab. As academic researchers, they primarily value accuracy and usability. Our modules need to represent data accurately so that it can be used in academic studies, and it needs to be use-able by researchers from widely varying backgrounds, including a complete inexperience with coding. Focusing on this ease of use for non-technical users, the application uses Python, particularly the PyQt5 library for a robust graphical user interface. The ultimate goal is to create a tool that not only supports current research needs but also lays the foundation for future projects in plant biology and environmental science.

This project builds on prior capstone efforts, enhancing unfinished modules and introducing new functionalities such as multi-instrument data integration. By combining cutting-edge data analysis with practical usability, the Mass Spectrometer Interface will empower researchers to make meaningful contributions to the fields of photosynthesis and respiration research.

## Use Cases

The use cases below describe common scenarios of user interactions with the Mass Spectrometer Interface system, illustrating how various functional requirements are applied in specific situations. These use cases provide a clear understanding of how users will engage with the system's features to achieve their goals. The proposed use cases are visually represented in the use case diagram shown in Figure 1.

A screenshot of a diagram

AI-generated content may be incorrect.  
Figure 1: Use Case Diagram

**Use Case 1: Calculate Bicarbonate/CO2 Ratio**

|  |  |
| --- | --- |
| Use Case | Calculate Bicarbonate/CO2 Ratio |
| Actors: | Lab researcher |
| Pre-condition: | File path to data folder has been selected, and bicarbonate and CO2 have both been calibrated. |
| Post-condition | Bicarbonate/CO2 Ratio is displayed in decimal format. |
| Main Flow | * User selects data acquisition folder. * Starts plotting data. * Selects data segment. * Calibrates bicarbonate and CO2. * Selects calculate Bicarbonate/CO2 button. * Bicarbonate/CO2 ratio is displayed. |
| Alternative Flow | * If bicarbonate or CO2 have not been calibrated, display an error message and retry. |
| Related Requirements | FR-1: Calculate Atom Ratio  FR-2: Center Mean Bars |

**Use Case 2: Plot Derivatives**

|  |  |
| --- | --- |
| Use Case | Plot Derivatives |
| Actors: | Lab researcher |
| Pre-condition: | File path to data folder has been selected. |
| Post-condition | The Atom Percent and its derivative are fully plotted on their respective graphs. Alert user data has run out. |
| Main Flow | * User selects data acquisition folder. * Starts plotting data. * Data is plotted to each graph in real-time. |
| Alternative Flow | * Invalid data in acquisition folder. * User is prompted to select new file path. |
| Related Requirements | FR-3: Plot Atom Percentage  FR-4: Plot Derivative |

**Use Case 3: Convert Data**

|  |  |
| --- | --- |
| Use Case | Convert Data |
| Actors: | Lab researcher |
| Pre-condition: | File path to data folder has been selected. File path to output folder has also been selected. |
| Post-condition | The output folder has been populated with formatted data and the user has been notified by popup that conversion has finished. |
| Main Flow | * User selects data acquisition folder. * Selects output folder. * Starts conversion. * Output file is populated with formatted data. * User is notified that conversion has finished. |
| Alternative Flow | * Invalid data is encountered. * Error message is displayed. * User may select a new input file. |
| Related Requirements | FR-8: Format Data |

**Use Case 4: Select Input Files**

|  |  |
| --- | --- |
| Use Case | Select Input Files |
| Actors: | Lab researcher |
| Pre-condition: | User has opened program. |
| Post-condition | File path has been set. Confirmation message is displayed. |
| Main Flow | * User selects data acquisition folder. * Confirmation message is displayed. |
| Alternative Flow | * Invalid input file is encountered. * Error message is displayed. * User may select a new input file. |
| Related Requirements | FR-9: Select Data Streams |

**Use Case 5: View Graph**

|  |  |
| --- | --- |
| Use Case | View Graph |
| Actors: | Lab researcher |
| Pre-condition: | Input file has been selected. Plotting has been started. |
| Post-condition | Graphs are filled by plotted data. User is notified that end of data source has been reached. |
| Main Flow | * User selects data acquisition folder. * Starts plotting * Graphs are populated * User is notified that end of data source has been reached. |
| Alternative Flow | * Invalid input file is encountered. * Error message is displayed. * User may try plotting again. |
| Related Requirements | FR-10: Plot Data Streams |

**Use Case 6: Manipulate Graph Scale**

|  |  |
| --- | --- |
| Use Case | Manipulate Graph Scale |
| Actors: | Lab researcher |
| Pre-condition: | Program is open |
| Post-condition | Graphs are resized to desired scale. |
| Main Flow | * User hovers over graph * Scrolls up or down to zoom in or out * Graph scales appropriately |
| Alternative Flow | * Maximum or minimum scale is reached * Zooming is limited by maximum size. |
| Related Requirements | FR-11: Manipulate Scale |

**Use Case 7: Change Plotting Speed**

|  |  |
| --- | --- |
| Use Case | Change Plotting Speed |
| Actors: | Lab researcher |
| Pre-condition: | Input file has been selected and data stream has not run out |
| Post-condition | Speed at which data is plotted is changed. |
| Main Flow | * User selects input folder. * Begins plotting. * Sets plotting speed. * Speed at which data is plotted to graph changes appropriately. |
| Alternative Flow | * Data stream runs out. * User is notified and must start over plotting. |
| Related Requirements | FR-12: Change Plotting Speed |

**Use Case 8: Select Data Points**

|  |  |
| --- | --- |
| Use Case | Select Data Points |
| Actors: | Lab researcher |
| Pre-condition: | Input file has been selected and data has been plotted to graph. |
| Post-condition | Vertical bars display the bounds of selected data. |
| Main Flow | * User selects input folder. * Begins plotting. * Selects Mean Bar option * Adjusts Mean Bars’ location |
| Alternative Flow | * If data has not been plotted, the Mean Bars will still appear but contain no data yet. |
| Related Requirements | FR-13: Mean Bars |

**Use Case 9: Calculate Mean**

|  |  |
| --- | --- |
| Use Case | Calculate Mean |
| Actors: | Lab researcher |
| Pre-condition: | Input file has been selected and data has been plotted to graph. Mean bars have been selected and adjusted to the desired location. |
| Post-condition | Mean value within the selected data segment is displayed. |
| Main Flow | * User selects input folder. * Begins plotting. * Selects Mean Bar option. * Adjusts Mean Bars’ location. * Selects calculate mean. * Mean value is displayed. |
| Alternative Flow | * If empty data segment is selected, user is prompted to select a new segment. |
| Related Requirements | FR-14: Calculate Mean |

**Use Case 10: Convert Data**

|  |  |
| --- | --- |
| Use Case | **Convert Data** |
| Actors: | Lab researcher |
| Pre-condition: | EZView data spool file has been selected. |
| Post-condition | A folder is created containing the data from the spool file in the format of modules 1-3 |
| Main Flow | * User selects input folder. |
| Related Requirements | FR-15: Format Data |

## Functional Requirements

#### Calculating Concentrations (Module 1)

|  |  |
| --- | --- |
| **Functional**  **Requirement** | **[FR-1] Calculate Atom Ratio** |
| Description | The system must allow the user to calculate the ratio of bicarbonate to carbon dioxide. The result of the calculation must display the ratio in decimal format. |
| Source | Client |
| Priority | Level 0 (Essential) |
| **Functional**  **Requirement** | **[FR-2] Center Mean Bars** |
| Description | The system must visually present mean bars in the center of the graph window. These vertical bars must be movable by the user so that a segment of data can be selected to calculate the mean for. |
| Source | Client |
| Priority | Level 1 (Desirable) |

#### Analyzing Enzyme Activity (Module 3)

|  |  |
| --- | --- |
| **Functional**  **Requirement** | **[FR-3] Plot Atom Percentage** |
| Description | The system must be able to plot the percentage of carbon dioxide with a mass of 49 out of all present molecules. The graph must be normalized using natural log. |
| Source | Client |
| Priority | Level 0 (Essential) |
| **Functional**  **Requirement** | **[FR-4] Plot Derivative** |
| Description | The system must be able to plot the derivative of the natural log of the percentage of carbon dioxide with mass 49. |
| Source | Client |
| Priority | Level 0 (Essential) |
| **Functional**  **Requirement** | **[FR-5] Copy Table** |
| Description | The system must allow the user to copy the contents of the mean value table. |
| Source | Client |
| Priority | Level 1 (Desirable) |
| **Functional**  **Requirement** | **[FR-6] Plot Datasets Consecutively** |
| Description | The system must be able to plot data from one acquisition folder, stop plotting, and then allow the user to select a new acquisition folder to begin plotting from. |
| Source | Client |
| Priority | Level 0 (Essential) |
| **Functional**  **Requirement** | **[FR-7] Record Sample Name** |
| Description | The system must take user input and record the name of the sample to the mean table in the same row as its corresponding mean value. |
| Source | Client |
| Priority | Level 0 (Essential) |
| **Functional**  **Requirement** | **[FR-17] Mean Bars** |
| Description | Two vertical bars must be included in the derivative graph of the application. The user should be able to move the bars to encapsulate a desired segment of data from which the mean can be calculated. The bar rescale button should center the bars on the derivative graph |
| Source | Client |
| Priority | Level 1 (Desirable) |

#### Data Conversion (Module 4)

|  |  |
| --- | --- |
| **Functional**  **Requirement** | **[FR-8] Accept Data** |
| Description | The system must accept data from the EZ-Tap listener device attached to the secondary mass spectrometer through the USB port. |
| Source | Client |
| Priority | Level 0 (Essential) |
| **Functional**  **Requirement** | **[FR-8] Identify Port** |
| Description | The system must identify which port the EZ-Tap is plugged into and intelligently select it for listening. |
| Source | Client |
| Priority | Level 1 (Desirable) |
| **Functional**  **Requirement** | **[FR-15] Format Data** |
| Description | The system must convert data from the secondary mass spectrometer into a specific format so that it can be analyzed using existing applications. The data must be converted from hexadecimal to decimal format and be resorted into the appropriate column order. |
| Source | Client |
| Priority | Level 0 (Essential) |
| **Functional**  **Requirement** | **[FR-16] Split Data** |
| Description | The system must split the data into many different csv files to match the data file structure used by the primary mass spectrometer. |
| Source | Client |
| Priority | Level 0 (Essential) |

## Non-Functional Requirements

Our project exists in the context of academic plant biology research, and thus has several ideals and values it strives to represent and uphold. These ideals don’t necessarily describe what the program *does*, they rather describe how it *is*. This section clarifies the non-functional requirements which we’ll use to guide our design and development to uphold these ideals and values.

**Modern Visuals:**

The system shall look like a modern 2020’s program. PyQt5’s baseline UI elements meets this standard

**Python For Extendibility:**

The system shall use Python since it is one of the most common languages among Biology students, including those that use the Cousins Photosynthesis Lab. This will enable extension and maintenance.

**Documentation For Extendibility:**

The system shall have documentation for code that’s thorough enough to allow college 4th-year skill level software engineers to extend and maintain it.

**Documentation For Non-Technical Users:**

The system shall have documentation to enable non-technical users to make full use of it. For example, an explanation of how to create executables of each module.

**Accuracy:**

The system shall accurately calculate and portray data. The instruments have their own levels of accuracy, so maintenance of significant figures is our basic metric for accuracy.

**Compatibility:**

Each module shall use compatible formats, where applicable. Particularly, this means using the format of the primary Mass Spectrometer.

**Non-Destructive:**

The system shall not overwrite or otherwise destroy any pre-existing data.

# Software Design

This section aims to outline our project’s approach. This includes the different parts of the project modules (Architecture Design), data storage and manipulation methods (Data Design), and how the project modules will outwardly look (User Interface Design). These details are intended to elucidate objectives and methods for current and future developers, but to do so using language that can mostly still be understood by stakeholders for overview.

The project is designed to provide a comprehensive platform for managing and interpreting Mass Spectrometer data through various modules. It aims to present data insights through graphical and numerical perspectives while offering flexible data formatting and integration capabilities. The system consists of four primary modules: the first three focus on different ways to visualize and log Mass Spectrometer data, and the fourth module reformats data from a secondary Mass Spectrometer. While the fourth modules can be used as a standalone program for data conversion, it has also been integrated into modules one and two as an option for integrated data conversion. The general design for each of the modules is to accept input from a file or folder, which is selected by the user, process it in some way (reformat, normalize, derive calculations from, etc.), and then output that processed information, either to graphs on the screen, or to new files.

## Architecture Design

### Overview

Team Linnaea Borealis' architecture design is shared among modules 1, 2, and 3. Each module in the system operates as a standalone program, each processing and visualizing datasets with unique goals in mind. However, the overall architecture of the 3 systems remains the same.

The team has adopted a Pipe and Filter architecture for this design, which is well-suited for this system's data processing requirements. The system requires data to be read from a file, normalized, plotted, and then transformed. This process fits Pipe and Filter architecture well. This approach allows for clear separation of concerns, where data flows through a series of independent filters (or stages), each performing specific tasks such as reading, processing, and visualizing data. This modular design enhances flexibility, scalability, and maintainability, making it easier to integrate new features or modify existing ones without affecting the entire system. By leveraging this architecture, Team Linnaeus ensures a clean and efficient flow of data from input to output, supporting both current and future needs.

The system begins with the User Interface (UI) component, which allows users to specify a file path and view plotted data and calculation results. The UI interacts with the File Reader, which reads raw data from the CSV file provided by the user and passes it to the Data Processor. The Data Processor serves multiple purposes depending on the module. In modules 2 and 3, the Data Processor is responsible for converting data from a secondary format to the standard format recognized by the system. In module 3, the Data Processor normalizes the raw data by applying a logarithmic transformation, preparing it for further steps. Once processed, the data is sent to the Graphing Engine, which updates the current graph by integrating the new data. The updated graph is then displayed in the UI for the user and sent to the Calculation Engine. The Calculation Engine performs further computations on the processed data, producing analytical results that are also displayed in the UI such as the mean of a selection of data. This flow of data, facilitated by the modular design, ensures efficiency, flexibility, and ease of maintenance.

A diagram of a system

Description automatically generatedFigure 2: System Block Diagram

### Subsystem Decomposition

This section outlines how the system has been decomposed into its major subsystems, each corresponding to the core components identified earlier: the User Interface (UI), File Reader, Data Processor, Graphing Engine, and Calculation Engine. The decomposition was designed to ensure that each subsystem represents a manageable unit of work for a single developer, with clearly defined responsibilities and minimal overlap. Each subsystem has been assigned a specific functionality: the UI handles user interactions, the File Reader manages data input, the Data Processor handles data transformations, the Graphing Engine updates the collection of data, and the Calculation Engine performs analytical operations. The rationale behind this decomposition emphasizes cohesion and coupling. High cohesion is maintained within each subsystem by ensuring that each is focused on a single task or closely related set of tasks. Meanwhile, coupling between subsystems is minimized by defining clear and straightforward interfaces for data exchange, allowing for seamless interaction without unnecessary dependencies. This modular design not only improves the maintainability and scalability of the system but also supports flexible integration of future enhancements.

#### User Interface

##### Description

The User Interface (UI) subsystem is responsible for managing the interaction between the user and the system. Its primary function is to display graphical data generated by the Graphing Engine and the results of calculations performed by the Calculation Engine. Additionally, the UI allows the user to input a file path containing CSV data, which is then passed to the File Reader subsystem for processing. Additionally, the user can select to convert the data from a non-standard format in modules 1 and 2. The UI acts as a central hub, presenting outputs from different subsystems in a coherent and user-friendly manner.

##### Concepts and Algorithms Generated

The User Interface subsystem was designed with simplicity and ease of use in mind, utilizing a basic file input mechanism along with graphical display capabilities. Concepts such as event-driven programming were considered, allowing the UI to react to user inputs (e.g., file path selection) and update dynamically when data from other subsystems (such as the graph or calculations) is received. The selected solution leverages standard UI frameworks to achieve these goals with minimal latency and high responsiveness. The decision to keep the UI simple was influenced by the need to maintain a clear separation of concerns, ensuring that complex data processing is handled by other subsystems. Trade-offs included balancing performance with usability, ensuring the UI remains responsive even when handling large datasets.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| GetFilePath | File Reader | The GetFilePath service will pass over the user-selected file paths for data acquisition. Three file paths should be returned, one for each instrument utilized by the system. |
| GetDataSelection | Calculation Engine | This service will return the bounds of the data selected by the user using two movable vertical bars. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| CalculateMean | Calculation Engine |
| GetLatestGraph | Graphing Engine |

#### File Reader

##### Description

The File Reader subsystem is responsible for reading data from a set of CSV files contained within a single folder. Each file contains a small segment of data. Files that are added to the folder during runtime will also be read allowing for near real-time updates. After reading the raw data from these files, the File Reader passes the data along to the Data Processor for further processing. This subsystem acts as the initial stage in the data pipeline, ensuring that the raw information is properly retrieved and made available for downstream components.

##### Concepts and Algorithms Generated

The File Reader subsystem was designed with the need to efficiently handle multiple data formats. The concept of parallel file reading was considered to improve performance when handling large datasets, but ultimately, a sequential approach was selected due to the relatively manageable file sizes. The File Reader ensures that each file is opened, its contents are read, and the data is collected into a format that can be easily passed to the Data Processor. Special considerations were made to handle potential file reading errors, such as missing or corrupted files. In such cases, error handling mechanisms trigger notifications to the user. The trade-off involved in this design was balancing robustness with simplicity, ensuring that file reading remains a lightweight process while effectively managing edge cases.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| ReadFileData | Data Processor | This service reads the contents of the three files, each corresponding to an instrument's data. The service accepts the file paths as input (provided by the User Interface) and returns the raw data from all three files as output. The data is passed on to the Data Processor for formatting and normalization. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetFilePath | User Interface |

#### Data Processor

##### Description

The Data Processor subsystem is responsible for transforming raw data received from the File Reader into a format that is more suitable for further analysis. Its primary task differs based on the module. In modules 1 and 2, the Data Processor is responsible for converting bytes of hex data into a readable format when the user selects to import data from the second mass spectrometer. In module 3, the primary task of the Data Processor is to normalize the data using the natural logarithmic transformation, making the data easier to process and visualize. Once the data is formatted and normalized, the Data Processor sends the processed data segment to the Graphing Engine, where the new data is added to the current graph. This subsystem ensures the integrity and usability of the data as it flows through the system.

##### Concepts and Algorithms Generated

The Data Processor utilizes a natural logarithmic transformation, which helps normalize the data. This technique was selected because it reduces variance in the data and handles large discrepancies in value magnitudes, making it more suitable for visual representation and computation. Alternative normalization methods, such as z-score normalization, were considered but ultimately not chosen, as the natural log method better aligned with the requirements of the data's structure and scale. Another algorithm was generated to parse and reconstruct bytes of hex data into the standard data format understood by the system. Special considerations included handling cases where the raw data contains zero or negative values, as these would cause issues during the logarithmic transformation. Additionally, cases in which an unexpected number of bytes were received were also taken into account by excluding additional bytes. Error handling mechanisms and data validation were incorporated to account for such anomalies, ensuring robust processing. The trade-off in this approach involved balancing data transformation speed and the complexity of handling edge cases.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| NormalizeData | Graphing Engine | This service receives the raw data from the File Reader, processes it by formatting and applying a natural logarithmic normalization, and then sends the processed data to the Graphing Engine for visualization. The input consists of the raw data a mass spectrometer, and the output is the normalized and formatted data ready for graphing. |
| ConvertData | Graphing Engine | This service takes data from the File Reader that comes from the lab’s second mass spectrometer, which outputs data in a different format. The service parses the stream of hex bytes, identifying specific data using it’s offset from a hexadecimal terminating sequence. The decimal data is then reconstructed in a tabular format. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| ReadFileData | File Reader |

#### Graphing Engine

##### Description

The Graphing Engine subsystem is responsible for visualizing the processed data received from the Data Processor. Its main role is to update the current graph by incorporating the newly processed data segment and rendering it in the user interface. This allows the user to observe real-time visual changes in the data as it is processed. The Graphing Engine ensures that the graphical representation is clear, accurate, and continuously updated as new data is received.

##### Concepts and Algorithms Generated

The Graphing Engine employs algorithms to dynamically update the graph with new data segments. Various graphing techniques were considered, including line charts and bar charts, but a continuous line chart was selected as the best fit for real-time data visualization. The primary concept is to ensure that the graph scales efficiently as more data is added, while also maintaining clarity and responsiveness.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| UpdateGraph | User Interface | This service receives the processed data from the Data Processor and updates the current graph displayed in the User Interface. The input is the normalized data, and the output is an updated visual graph reflecting the latest data. The service ensures that the graph is refreshed in real-time as new data is processed. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetProcessedData | Data Processor |

#### Calculation Engine

##### Description

The Calculation Engine subsystem is responsible for performing various calculations on the data, including allowing the user to select a segment of data and calculating the mean of that segment. This subsystem plays a crucial role in providing numerical insights into the processed data, complementing the visual representation provided by the Graphing Engine.

##### Concepts and Algorithms Generated

The primary algorithm used by the Calculation Engine is the calculation of statistical measures such as the mean. The mean is calculated by summing the selected data points and dividing by the number of points. Several approaches were considered, including pre-calculating statistics for all data and storing them for quick access. However, the selected solution was to compute the required statistics on-demand based on the user’s selection. This minimizes resource consumption and ensures that calculations are based on the latest data segment selected by the user.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| CalculateMean | User Interface | This service calculates the mean of the selected data segment. The input is the processed data for the user-selected segment, and the output is the mean value. The mean is displayed in the User Interface as part of the calculation results. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetDataSelection | User Interface |
| GetLatestGraph | Graphing Engine |

## Data Design

This section covers the different ways our project stores and manages data, including temporary data structures as well as file storage. This overview will be a fairly technical overview, meant mostly for developers. Each subsection covers a different data structure, file type, or pattern used in the project. The data used in this project is not stored in a database, so the data design revolves around temporarily stored data in memory as well as data exported to files.

### Shared Singleton

Shared singleton is a design pattern that forces only one instance of an object to exist throughout the whole program. Although this design pattern does not usually describe a data structure, in modules 1-3, most important data structures are members of a shared singleton, named sharedData. These members include fileList (subsection IV.2) and dataPoints (subsection IV.3)

### Python List

In modules 1-3, Python lists are utilized to store the names of files that are read from. This list is named fileList and is stored in the sharedData shared singleton (subsection IV.1).

### Dictionary

In modules 1-3, we use a Python dictionary to store the mass spectrometer data, named dataPoints. The keys are time points, and the values are tuples of isotope masses. This dictionary is stored in the sharedData shared singleton (subsection IV.1).

### Log Table

In modules 1-3, there are certain data points that can be pulled out from the graph, or otherwise calculated from the data. These can be saved into a table built into the UI. These logs are then stored directly in the PyQT UI component QTableWidget.

### CSV Export

Modules 1-3 can export logged data from the log table (subsection IV.4) into a CSV file, which matches the format of the table. This data can also be copied to the clipboard for external use.

### Pandas DataFrame

Module 4 uses a DataFrame from the Pandas Python library to store data from Mass Spectrometer 2. This is very similar to a dictionary approach (subsection IV.3), but effectively replaces the need for unique keys with ordering and more importantly allows for the use of Pandas methods such as to\_csv at a small performance cost.

### CSV Series

The first mass spectrometer utilized in Cousin’s lab outputs a series of CSV files in which each row contains a time signature and isotope mass readings. These CSV’s are named with numbers in order of their time signatures. This is the main file format recognized by modules 1-3 and it is also the output format generated by conversion in Module 4. Module 4 exports the DataFrames (subsection IV.6) as CSV’s with ~8 rows each, for use in modules 1-3.

## User Interface Design

The user interface allows the user to select data acquisition folders sourced from a mass spectromer instrument by navigating to the File options. This portion of the interface corresponds with use case UC-4: Select Input Files. As seen in Figure 3, the user can also select an EZView Data File in the case that they would like to utilize module 4 to convert data from the second mass spectrometer type.

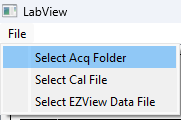


Figure 3: File Selection

The largest section of the user interface displays a black graph such as the one shown in Figure 4. To the left of the graph, color coded labels for each data stream are present to provide an intuitive viewing experience. These labels can also be used to toggle the visibility of different data streams. The graph is populated by line-graphs, color coded for each data stream. This corresponds to the use case UC-5: View Graph. The graph can be scaled easily using the scroll wheel to zoom in or out. Below the graph and to the right of Figure 4, a slider is shown. This slider allows the user to adjust the speed at which data is plotted from 0.5x to 32x. This corresponds to use case UC-7: Change Plotting Speed. This slider will automatically adjust itself if the plotting speed exceeds the rate at which data is coming in. On the left of the slider is also a pause/resume button as well as a start button that allows the user to toggle the plotting of data. In addition to the primary graph, modules 1-3 have additional graphs for calculations derived from the primary data.

A screen shot of a graph

Description automatically generatedFigure 4: Graphing and Calculations

Using the leftmost button in Figure 4 that features two vertical lines, the user can toggle the visibility of “mean bars”. These vertical bars pictured in Figure 5 can be shifted left or right using the cursor to select a specific portion of data. This corresponds to the use case UC-8: Select Data Points.

A blue grid with white lines

Description automatically generatedFigure 5: Data Segment Selection

Available calculations are featured below the graph and its controls. Here, the user will be able to use the “Get Mean” button to calculate the mean inside of the selected region. This corresponds to use case UC-9: Calculate Mean.



Figure 6: Mean Calculation

In addition to allowing users to scale the graph using their scroll wheel, users are able to achieve more precise scaling by left clicking the graph to open further adjustment options. As illustrated in Figure 7, users will be able to input the exact bounds of each axis, and the graph will adjust according to their selection.

A screenshot of a computer program

Description automatically generated

Figure 7: Detailed Graph Scaling

Overall, the user interface is designed for an intuitive data acquisition and visualization experience. Users can select acquisition folders as well as EZVeiw formatted folders as their input data. Figure 3 (UC-4: Select Input Files). The main area features a black graph displaying color-coded line graphs, accompanied by labeled indicators for clarity (UC-5: View Graph). Users can zoom using the scroll wheel and adjust the plotting speed with a slider (UC-7: Change Plotting Speed). Controls for toggling mean bars and selecting data points further enhance functionality (UC-8: Select Data Points), while a “Get Mean” button allows for quick calculations of averages within selected regions (UC-9: Calculate Mean). Additionally, precise graph scaling options enable users to customize their view, ensuring a comprehensive and user-friendly interface.

# Test Case Specifications and Results

This section provides an overview of the steps we take to test different elements of our project. This overview includes the overall flow, the unit tests for individual parts, and integration/system testing for combinations of different parts. The processes outlined are very particular to the context of our project; we’re very aware of our stakeholders and the integration plan that they prefer. Ultimately this section is intended to outline what we consider the ideal methods of testing, in order to keep development in line with those practices.

## Testing Overview

The following is our loose approach to testing a particular module or feature:

1. Identify the requirement(s) involved in this module/feature. This should either come from the Requirements and Specifications Document or be added to the Requirements and Specifications Document before continuing.
2. Establish the test(s) that will be used. In other words, identify the process of using the module or feature. Document these tests in the Testing Plan Document.
3. Identify any necessary dependencies. This includes other components and input data. Include assumptions about these dependencies in the Testing Plan Document.
4. Build a representation of what acceptable results look like. This must consider our assumptions made in the previous step. For example, an Excel graph of a data acquisition: the particular data acquisition should be clarified in the previous step, with the Excel graph built off it in this step. This mockup(s) should either be included in the Testing Plan Document, or in the relevant module’s “Testing” folder with reference to it in the document.
5. Perform the test(s).
6. If the test(s) is unsuccessful, fix it if possible. If the test(s) is not successful by next standup meeting, prepare a short explanation or document explaining the issue.
7. If the test is successful, move the relevant GitHub issue to Review/QA, or from Review/QA to Done.

Ultimately, our strong connection to our primary stakeholder, Dr. Cousins, allowed us to adapt our development process to a more flexible approach that handled opportunities and issues as they arose. Sometimes when the requirements were vague, the following approach was more effective than generating more specific requirements and a mockup:

1. Implement the most obvious executions of a requirement.
2. Present those executions to the client/stakeholder(s). Receive feedback.
3. If one of the implementations is acceptable, move the relevant GitHub issue to Review/QA, or from Review/QA to Done.
4. If none of the implementations are acceptable, either return with novel implementations or revert to the primary approach, depending on team consensus.

Our delivery process is basically Continuous Delivery. Our client prefered executable files over python scripts, so a new deployment must be manually created by a team member each time. The modular nature of the project lends itself to creating a new iteration of each improved module every sprint. In this regard, our development is continuously integrated with monthly releases.

#### Unit Testing:

The primary objective of unit testing for this system is to validate the functionality and reliability of individual components, or “units,” by isolating them from the rest of the code and checking for bugs or unexpected behavior. Specifically, the unit tests cover core functionalities, including data parsing from CSV files, data transformations, calculations, and graphical display setup. Unit tests are designed for each function and method within the modules, verifying both expected outputs and error-handling mechanisms when presented with invalid data inputs. This approach ensures that each unit functions independently and accurately, laying a stable foundation for subsequent integration and system testing phases.

#### Integration Testing:

The purpose of integration testing in this system is to identify faults that may arise when individual components interact, focusing on groups of components rather than isolated units. This phase ensures that data flows smoothly between components. For instance, components responsible for parsing CSV files and transforming data are integrated and tested as a cohesive unit to confirm that each stage performs as expected in the broader workflow. To manage dependencies, a test data set simulating real-world CSV inputs is be used to validate functionality and data consistency across components.

#### System Testing:

System testing will be conducted to ensure that the mass spectrometer interface system operates as a cohesive unit, meeting all specified requirements. This phase involves executing a series of planned tests to validate both functional and non-functional aspects of the comprehensive system.

#### Functional Testing:

In functional testing, we tested cases based on the functional requirements outlined in the project documentation. Each functional requirement corresponded to at least one test case. Each standalone system was tested in realistic scenarios to ensure that they meet user expectations. Test cases were prioritized to focus on critical user paths and high-risk areas, ensuring that the most important features were validated first. Any failures or discrepancies found during testing were fixed in the next system iteration.

#### Performance Testing:

To assess the system's performance, we focused on response times, resource utilization, and overall system stability. This includes stress testing the systems by simulating high-load conditions using large datasets to determine how the system performs under pressure. We measured the speed at which the system can plot all data points from a large sample at once. Key metrics such as processing speed, memory usage, and data handling capacity were monitored to ensure that lab computers could handle the system. No major performance issues were identified.

#### User Acceptance Testing:

User Acceptance Testing utilized lab researchers in evaluating the system based on their operational needs. Lab researchers performed key tasks, such as loading CSV files, converting data, plotting data, and processing calculations in the same way they would during their academic research. Feedback was be gathered during these sessions to identify any areas requiring adjustment or enhancement. This testing phase was crucial for ensuring that the system is user-friendly and meets the designated requirements.

## Environment Requirements

To ensure comprehensive testing of the mass spectrometer interface system, this section outlines the necessary and desired properties of the testing environment. The setup will enable thorough verification of each module’s functionality, accuracy, and performance in processing and visualizing mass spectrometry data.

The testing environment should be equipped with a Windows 10 or Windows 11 operating system. The environment will rely on Python, version 3.8 or higher, to match the development specifications of the system. Key Python libraries, such as PyQt5 for graphical user interface elements and pandas for handling data and CSV file operations, should be pre-installed. Each module has a requirements.txt file that can be used with pip to install the necessary libraries. Additionally, a sample CSV file will be prepared, including real mass spectrometer data, to evaluate the system’s handling of data directly from a file.

Module 4 has extra special requirements for its extra specific use case. The module takes in data from a Mass Spectrometer through an EZ-Tap serial-to-usb listener device. Both the Mass Spectrometer and the EZ-Tap are assumed for the testing and usage of module 4. In order to ensure these conditions, we use remote-access to test on the computer that has the EZ-Tap plugged into it. Additionally, the program requires EZView to spool the data from the EZ-Tap before it can be reformatted. EZView is proprietary software from Stratus Engineering, the same company that manufactures EZ-Tap.

These environment specifications aim to provide a controlled, reliable setting that ensures system stability, accuracy, and performance across diverse data scenarios.

## Test Results

.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aspect Being Tested | Expected Result | Observed Result | Test Result | Test Case Requirements |
| Calculate Atom Ratio | Pressing the “BiCarb/CO2” button will yield 0.84. | 0.84 | Pass | Acquisition 4336 is loaded into Module 1. Data between 40-45 seconds is selected. |
| Mean Bar Placement | Mean bars appear in the center of the graph view upon pressing mean bars button. | Mean bars appeared centered | Pass | Dataset has been chosen in Module 3 |
| Atom Percentage plotting accuracy | Curve displayed matches that of test oracle software. | Identical curve | Pass | Load Acquisition 4336 into Module 3 as well as oracle (deprecated system) |
| Derivative plotting accuracy | Curve displayed matches that of test oracle software. | Identical curve | Pass | Load Acquisition 4336 into Module 3 as well as oracle (deprecated system) |
| Table Copy Functionality | Table data is copied to clipboard and pasting into excel matches the program’s table. | Program table and pasted excel table match. | Pass | Load Acquisition 4336 into Module 3. Use mean bars to select data segment. Click “Add to Table” and copy table contents before pasting into excel. |
| Consecutive Dataset Plotting | After stopping plotting of dataset, a new acquisition folder may be selected and will plot the new dataset. | New dataset was plotted. | Pass | Load any acquisition folder into module 3. Start and stop plotting. Load Acquisition 4336. |
| Record Sample Name | Typed sample name appears next to sample data in table. | Sample name was present in correct row. | Pass | Load Acquisition 4336 into Module 3. Use mean bars to select data segment. Enter Sample name and click “Add to Table”. |
| Mean Bar Data Selection | Cursor dragging adjusts mean bars horizontally. | Cursor dragging adjusts mean bars horizontally. | Pass | Load Acquisition 4336 into Module 3. |
| Access EZTap data | Module 4 prototype continuously outputs stream of data. | Module 4 prototype continuously outputs stream of data. | Pass | EZTap is connected to pc and mass spectrometer. |
| Identify correct port | Module 4 prototype continuously outputs stream of data. | Module 4 prototype continuously outputs stream of data. | Pass | EZTap is connected to pc and mass spectrometer. |
| Formatting of Data | Data output of Module 4 matches Acquisition output files. | Data output of Module 4 matches Acquisition output files. | Pass | EZView is running and data spooling is turned on. |
| Split Data Into CSV Chunks | Multiple CSV files are output each with roughly 8 rows of data. | Multiple CSV files are output each with roughly 8 rows of data. | Pass | EZView is running and data spooling is turned on. |

# Projects and Tools Used

The table below lists the libraries, frameworks, and tools used to implement our project.

|  |  |
| --- | --- |
| Tool/Library/Framework | What It Was Used For |
| PyQt | GUI library/framework used in Modules 1-3 |
| Pandas | Maintaining data in Modules 1-3 |
| PyQtGraph | Adding graphs to PyQt GUI |
| NumPy | Establishing compatibility between our data and PyQtGraph |
| PyInstaller | Creating .exe executables |
| Excel | Generating example output for tests |
| LabView | Orienting visual design |

The table below lists the languages used in our project.

|  |  |  |
| --- | --- | --- |
| Languages Used In Project | | |
| Python | Markdown | C |

# Description of Final Prototype

With this prototype, Module 3 delivers 100% of its proposed functionality. Users are now able to plot log-normalized atom percentages. This graph shows the percentage of carbon dioxide molecules with a mass of 49. Normalization simplifies data interpretation by eliminating large spikes that would otherwise exceed the viewport. The UI component for this feature is shown in Figure 8 where real-world data has been plotted.

Figure 8

The system also allows users to compute and plot the derivative of the Atom percentage for trend analysis. This graph is accompanied by vertical bars used for data selection. These bars have been modified to always appear centered in the viewport and scale appropriately with the graph. Both of these implementations are featured in Figure 9.

Figure 9

In this prototype, data from the mean value table can be easily copied for external use. Additionally, users can input sample names, linking them directly to their corresponding mean values for better organization and reference. The mean value table is displayed in Figure 10.



Figure 10

In module 3, users can now plot multiple datasets without restarting the program. This was not possible in the previous solution because users were not allowed to select a second dataset. Now, the system can plot one dataset, stop, and then plot a new dataset as selected by the user.

The system is now able to accurately plot the normalized percentage of carbon dioxide molecules with a mass of 49. Additionally, the system supports plotting the derivative of this percentage. Mean bars for data selection have been updated to scale appropriately with the graph, as shown in the figure below. The ability to copy data from the mean value table has also been implemented, enabling seamless transfer of information for external use. Furthermore, the prototype allows for consecutive dataset plotting, where users can plot data from one acquisition folder, stop, and then select a new folder to begin a fresh plot without restarting the program. Lastly, the system supports recording sample names, enabling users to input and store identifiers alongside their corresponding mean values, as illustrated in Figure 11. If no further bugs are identified, no work remains to be done on module 3.



Figure 11

Since new versions of the prototype were delivered to the client with each update, several preliminary tests were conducted on module 3. These tests were primarily conducted during weekly meetings with the team and client. Real-world mass spectrometer datasets were utilized to observe the plotted results and compare them to past software solutions as well as ideal outcomes visualized in Excel. The first set of prototypes yielded derivative graphs that did not match expectations. This testing process helped form the ideal calculations for future versions. Lab researchers also helped verify the accuracy of plotting by comparing the plotted values with a previously used graphing software. With later prototypes, all functions mentioned in the previous section were successfully tested with the exception of modifications to the mean table and consecutive dataset plotting which are currently undergoing testing by the client.

Module 4 has been fully implemented and integrated into modules 1 and 2. Figure 11 display how Modules 1 and 2 now allow the user to simply select an EZView Data File to utilize Module 4 to convert the data into a compatible format. As the data conversion feature provided by Module 4 is not user-facing, there are no other figures displaying its user interface.

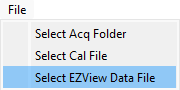


Figure 11

Module 4 functions by first accessing data from a text file that is created by EZView, a proprietary software that accompanies the EZ-Tap USB device. To access the data, the user first starts EZView and selects the spooling button featured in Figure 12. The data is then saved to a text file in hexadecimal chunks separated by the terminating sequence “ffffffff”. These chunks of data are parsed, converted to decimal, and formatted in a tabular structure before being exported to CSV files. Each output file contains about 8 rows of data. This data can then be utilized by modules 1-3 for data analysis.

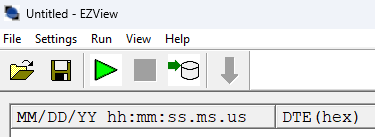


Figure 12

# Product Delivery Status

All source code is available on the publicly viewable GitHub repository KylerKupp/python-desktop-app. Dr. Asaph Cousins has also been given control privileges in this repository.

Executables have also been created for each module. These executables are available to Dr. Asaph Cousins through a shared OneDrive folder. These executables are not necessary to run their respective modules, they simply enable seamless portability and executability. The GitHub repository includes all necessary files for making these executables, including source code and .SPEC executable-creation-instruction files.

Although the product does not directly include any equipment, it is built to work with the mass spectrometers in the Cousins Lab. These mass spectrometers are the property of WSU, have stayed in the Cousins Lab throughout our work, and will remain in the Cousins Lab until WSU sees fit to move or remove them.

# Conclusions and Future Work

## Limitations and Recommendations

The primary limitations of this project are in the domain of performance. The most significant place where this stands out is when Modules 1 and 2 are used to take in data from Mass Spectrometer 2. When this is done, it makes use of an integrated version of Module 4. The combined performance delays of Modules 1/2 and Module 4 make for significant delays in information display at times.

Another limitation is the data intake speed when using Mass Spectrometer 2. In this regard, we are limited by the USB connection’s relatively low bitrate. We recommend further development look into the possibility of a direct serial connection.

## Future Work

The clear next step in this project is to create a module for combining the data streams of 3 other instruments in the lab. Our client has expressed interest in this additional module for a while now. This module would involve collating these data streams and displaying them, much like modules 1-3. It would need to synchronize the data as well.

# Appendix A - User Stories

The following user stories outline key tasks that users can perform within the system, highlighting their goals and the reasons behind them. Each story provides a clear and concise description of what the user aims to achieve, along with the expected system behavior, ensuring that user needs are directly addressed and aligned with the system's functionality.

**User Story US1: Calculate Bicarbonate/CO2 Ratio**

As a Lab Researcher, I need to calculate the ratio of bicarbonate to carbon dioxide so that I can log the given output.

Feature: Calculate Bicarbonate/CO2 Ratio

Scenario: Lab researcher calculates bicarbonate/CO2 ratio

Given the mass spectrometer data file has data for HCO-3 and CO2  
AND CO2 and HCO-3 have been calibrated using the plotted data  
When I click the button "BiCarb/CO2"  
Then the correct ratio of BiCarb/CO2 will be computed and displayed.

**User Story US2: Plot Derivatives**

As a Lab Researcher, I need the first and second derivatives of percent CO2 49 So that I can analyze and log it.

Feature: Plot Derivatives

Scenario: Lab researcher plots Atom Percent and its derivative for CO2 data

Given the mass spectrometer data file has data on the CO2 49, 47 & 45 masses.  
When I run module 3 (using the "start" or "unpause" button)  
Then two graphs will show the first and second derivatives of percent CO2 49 respectively.

**User Story US3: Convert Data**

As a Lab Researcher, I need modules 1-3 to work for the second mass spectrometer so that I can analyze its data the same way.

Feature: Convert Data

Scenario: Lab researcher converts data to be compatible with modules 1-3

Given Instrument B (the second mass spectrometer) is outputting data, or has outputted data  
WhenI select the Instrument B option on module 1/2/3  
And I select the Instrument B datastream (likely a directory)  
Then module 1/2/3 functions as normal, including all use cases for module 1/2/3.

**User Story US4: Datastream Combining**

As a Data Researcher, I need data from the LI-COR Leaf-gas Exchange System, Tunable Diode Laser, and Picarro consolidated into one or more spreadsheets so that I can analyze the data more efficiently.

Feature: Datastream Combining

Scenario: Lab researcher combines three streams of data

Given I have data from all three instruments  
When I run module 5  
Then one or more spreadsheets collectively containing all the data, collated by time, is created.

**User Story US5: Data Isolation**

As a Data Researcher, I need to isolate portions of data so that I can view and analyze the most important parts of a multi-hour lab experimentation session.

Feature: Data Isolation

Scenario: Lab researcher isolates data

Given the System has collected a non-trivial amount of data from the mass spectrometer  
When I drag the left and right edges of my selection to a portion of the data graph  
Then only data from that portion of the graph will be analyzed in the calculation dashboard.

**User Story US6: Data Isolation**

As a lab researcher, I need modules 1-3 to work for instrument B (the second mass spectrometer) so that I can analyze its data the same way.

Feature: Data Conversion

Scenario: Lab researcher Converts Data

Given Instrument B (the second mass spectrometer) is outputting data, or has outputted data through EZView

When I select the data spool file

Then module 4 creates a folder containing that data in the format used by modules 1-3.

# Appendix B - Traceability Matrix

The table below links functional requirements to their corresponding use cases and user stories. This mapping ensures that every requirement is fully addressed and connected to relevant user interactions, providing a clear line of traceability throughout the system.

|  |  |  |  |
| --- | --- | --- | --- |
| Functional Requirement | Use Case | User Story | Priority |
| [FR-1] Calculate Atom Ratio | UC-1: Calculate Bicarbonate/CO2 Ratio | US5: Data Isolation | Level 0 |
| [FR-2] Center Mean Bars | UC-1: Calculate Bicarbonate/CO2 Ratio | US1: Calculate Bicarbonate/CO2 Ratio | Level 1 |
| [FR-3] Plot Atom Percentage | UC-2: Plot Derivatives | US2: Plot Derivatives | Level 0 |
| [FR-4] Plot Derivative | UC-2: Plot Derivatives | US2: Plot Derivatives | Level 0 |
| [FR-8] Format Data | UC-3: Convert Data | US3: Convert Data | Level 0 |
| [FR-9] Select Data Streams | UC-4: Select Input Files | US4: Datastream Combining | Level 0 |
| [FR-10] Plot Data Streams | UC-5: View Graph | US4: Datastream Combining | Level 0 |
| [FR-11] Manipulate Scale | UC-6: Manipulate Graph Scale | US4: Datastream Combining | Level 1 |
| [FR-12] Change Plotting Speed | UC-7: Change Plotting Speed | US4: Datastream Combining | Level 0 |
| [FR-13] Mean Bars | UC-8: Select Data Points | US5: Data Isolation | Level 0 |
| [FR-14] Calculate Mean | UC-9: Calculate Mean | US4: Datastream Combining | Level 0 |