COLORADO SCHOOL OF MINES

Team Graph

Alperen Ugus

Murat Tuter

Sukru Kiymaci

Client

Dr. Owen Hildreth

Advisor

Christopher Painter Wakefield

Introduction

During scientific projects and experiments, a lot of data is collected, and this data mainly needs to be graphed to be understood or compared with other data. During his master's years, Dr. Owen Hildreth, an Assistant Professor in the Department of Mechanical Engineering at the Colorado School of Mines, created an application called GraphBase. This application was able to organize, parse and graph a data set read from a file. The program was written with Objective C, which needed to be updated. Last year, with Dr. Hildreth's sponsorship, the program was re-written using the latest version of the Swift language with Apple's latest APIs by a student taking the CSCI-370 Field Session class. Dr. Owen Hildreth requested to add some additional features to the Graphs. First of all, when the program tries to graph a large file, it becomes unresponsive and blocks the user interface. This is because it runs on a single thread. Therefore, multithreading is needed. In addition, when a directory is imported to the program, files with an extension other than .csv, .txt, and .dat must be excluded. Last but not least, visual feedback on parsing and loading operations was required as well as when multiple files are selected, multi-graphing is needed to be enabled.

Requirements

1. Add multithreading to the parser and graph controllers to open up large files without blocking the UI and main thread.
2. Add an option to "skip data" for large files.
    - e.g., only graph every Nth data point.
    - This will require to develop some scheme to determine this and a UI for the user to select or deselect this option.
    - How will the user know this schema is being applied to a particular graph/data file?
3. Filter out the non-data files when the user dragging and dropping their directories to the program.
    - e.g., only include the targeted .txt type of files and don't include images in the database
4. Add visual feedback on parsing/loading operations

- Add a progress bar with a percentage to show user how much time left for parsing.

System Architecture

As a team, we worked on the Graphs program version 1.0. It is an application written in Swift programming language using the Model View Controller design pattern. In this pattern, the users interact with the Views, and their interactions are controlled by the Controllers. The Model represents the classes, i.e., data for the application.
For example, assume there is an oval button with a blue background. All these specifications are under view. When the button is clicked, the Controller handles the action. If the click triggers any changes on the Model, the Controller reflects the update to the view. The view is related to the visual display of the button. The Controller handles the functionality. Model is the data representation.

In the Graphs application, there is the main storyboard which shows all the views in the application on Xcode.

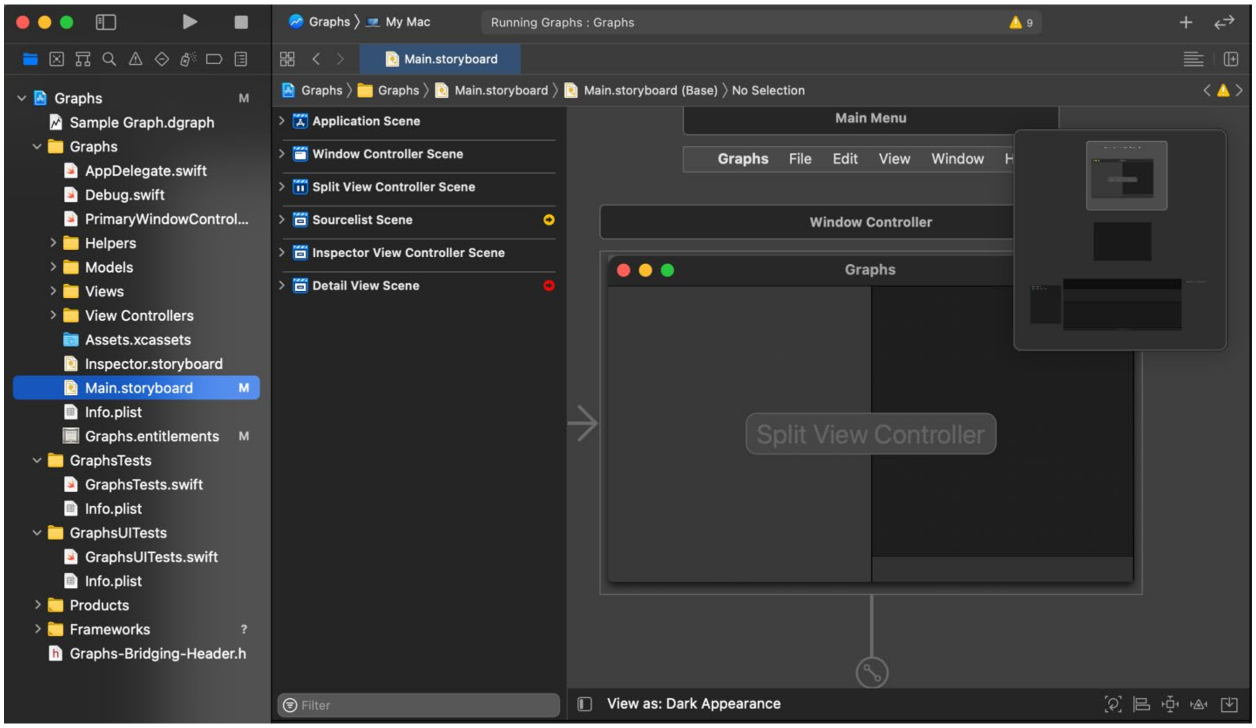


Figure 1: Main Storyboard

As can be seen from the screenshot above, Graphs has one single window. There are different subviews under the main window, and each view in the application has its own Controller. The fundamental three views of the Graphs application are Sourcelist Scene, Inspector Scene, and Detail View Scene.

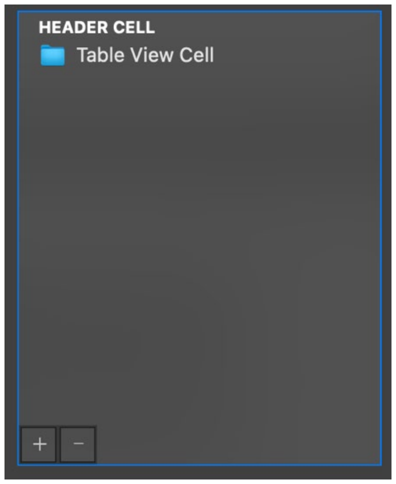


Figure 2: Source List Scene

The source list is responsible for showing all imported folders. More directories can be added, existing ones can be deleted, and a new folder can be created in this view. In addition, users can drag and drop directories to import them simply.

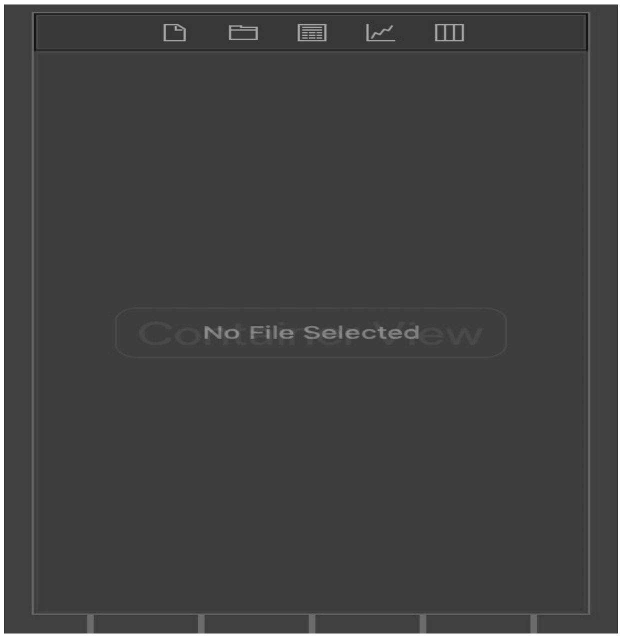


Figure 3: Inspector Scene

Inspector Scene is responsible for many sub-views: File inspector, directory inspector, parser inspector, graph inspector, and data inspector views. All have their own controllers and functionalities, as mentioned above.

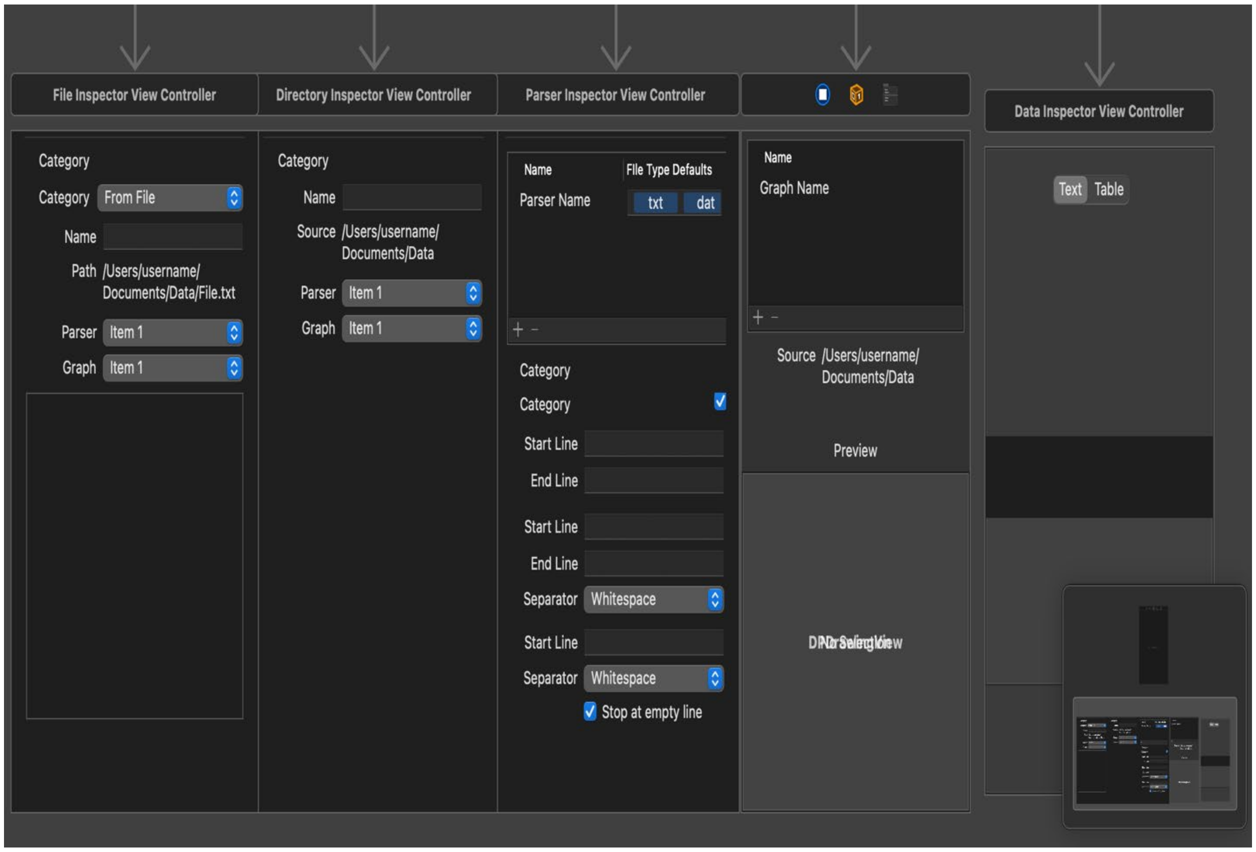

File Inspector View Controller
Directory Inspector View Controller
Parser Inspector View Controller
Data Inspector View Controller
Category
Category From File
Name
Path /Users/username/ Documents/Data/File.txt
Parser Item 1
Graph Item 1
Source /Users/username/ Documents/Data
Name
File Type Defaults
Parser Name
txt
dat
Start Line
End Line
Separator Whitespace
Stop at empty line
Graph Name
Preview
Text
Table

Figure 4: Inspector Scene Sub-views

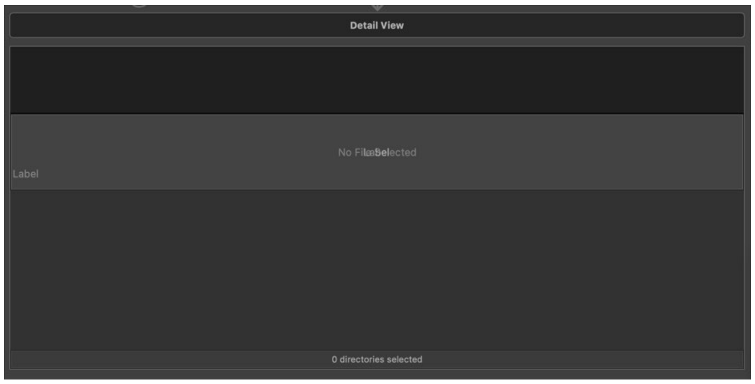
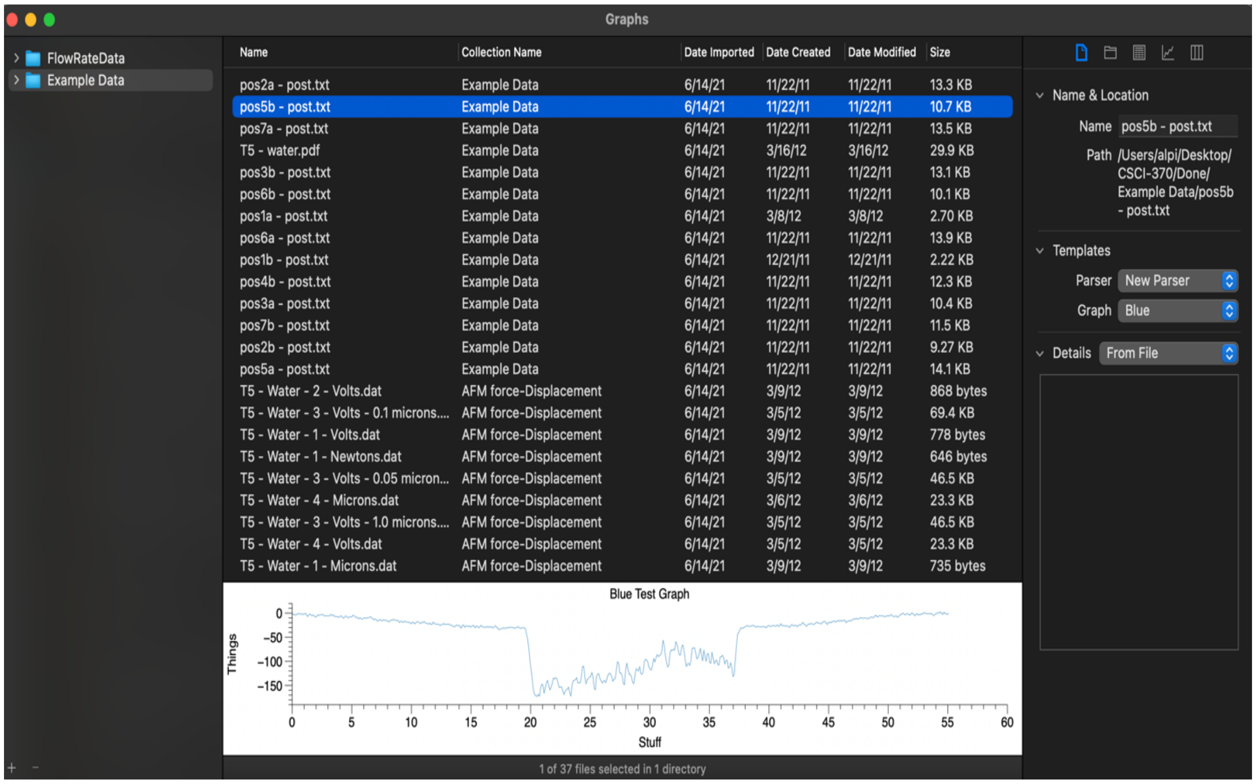


Figure 5: Detail View Scene

The Detail View Scene is where the graph is displayed can be seen above.



FlowRateData
Example Data

Name	Collection Name	Date Imported	Date Created	Date Modified	Size
pos2a - post.txt	Example Data	6/14/21	11/22/11	11/22/11	13.3 KB
pos5b - post.txt	Example Data	6/14/21	11/22/11	11/22/11	10.7 KB
pos7a - post.txt	Example Data	6/14/21	11/22/11	11/22/11	13.5 KB
T5 - water.pdf	Example Data	6/14/21	3/16/12	3/16/12	29.9 KB
pos3b - post.txt	Example Data	6/14/21	11/22/11	11/22/11	13.1 KB
pos6b - post.txt	Example Data	6/14/21	11/22/11	11/22/11	10.1 KB
pos1a - post.txt	Example Data	6/14/21	3/8/12	3/8/12	2.70 KB
pos6a - post.txt	Example Data	6/14/21	11/22/11	11/22/11	13.9 KB
pos1b - post.txt	Example Data	6/14/21	12/21/11	12/21/11	2.22 KB
pos4b - post.txt	Example Data	6/14/21	11/22/11	11/22/11	12.3 KB
pos3a - post.txt	Example Data	6/14/21	11/22/11	11/22/11	10.4 KB
pos7b - post.txt	Example Data	6/14/21	11/22/11	11/22/11	11.5 KB
pos2b - post.txt	Example Data	6/14/21	11/22/11	11/22/11	9.27 KB
pos5a - post.txt	Example Data	6/14/21	11/22/11	11/22/11	14.1 KB
T5 - Water - 2 - Volts.dat	AFM force-Displacement	6/14/21	3/9/12	3/9/12	868 bytes
T5 - Water - 3 - Volts - 0.1 microns....	AFM force-Displacement	6/14/21	3/5/12	3/5/12	69.4 KB
T5 - Water - 1 - Volts.dat	AFM force-Displacement	6/14/21	3/9/12	3/9/12	778 bytes
T5 - Water - 1 - Newtons.dat	AFM force-Displacement	6/14/21	3/9/12	3/9/12	646 bytes
T5 - Water - 3 - Volts - 0.05 micron...	AFM force-Displacement	6/14/21	3/5/12	3/5/12	46.5 KB
T5 - Water - 4 - Microns.dat	AFM force-Displacement	6/14/21	3/6/12	3/6/12	23.3 KB
T5 - Water - 3 - Volts - 1.0 microns....	AFM force-Displacement	6/14/21	3/5/12	3/5/12	46.5 KB
T5 - Water - 4 - Volts.dat	AFM force-Displacement	6/14/21	3/5/12	3/5/12	23.3 KB
T5 - Water - 1 - Microns.dat	AFM force-Displacement	6/14/21	3/9/12	3/9/12	735 bytes

Name & Location
Name pos5b - post.txt
Path /Users/alpi/Desktop/CSCI-370/Done/Example Data/pos5b - post.txt
Templates
Parser New Parser
Graph Blue
Details From File

1 of 37 files selected in 1 directory

Figure 6: Graphs Application

Overall, the application can import folders including data files, parse and display the data of the selected file, and display the graph of the data.

Technical Design

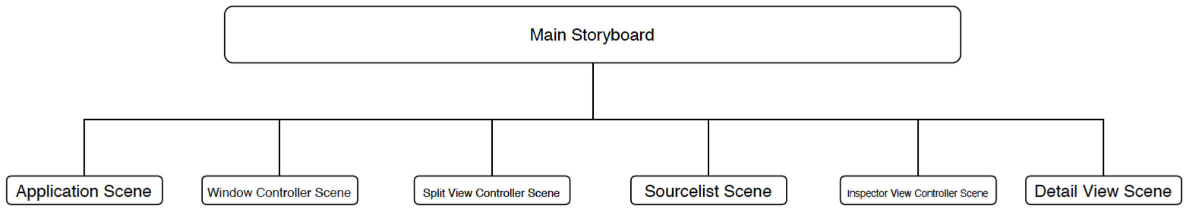


Figure 7: Main Storyboard

The program consists of some different scenes, as can be seen in the figure above. While the program runs on the main storyboard, scenes are divided into different screens in one window.

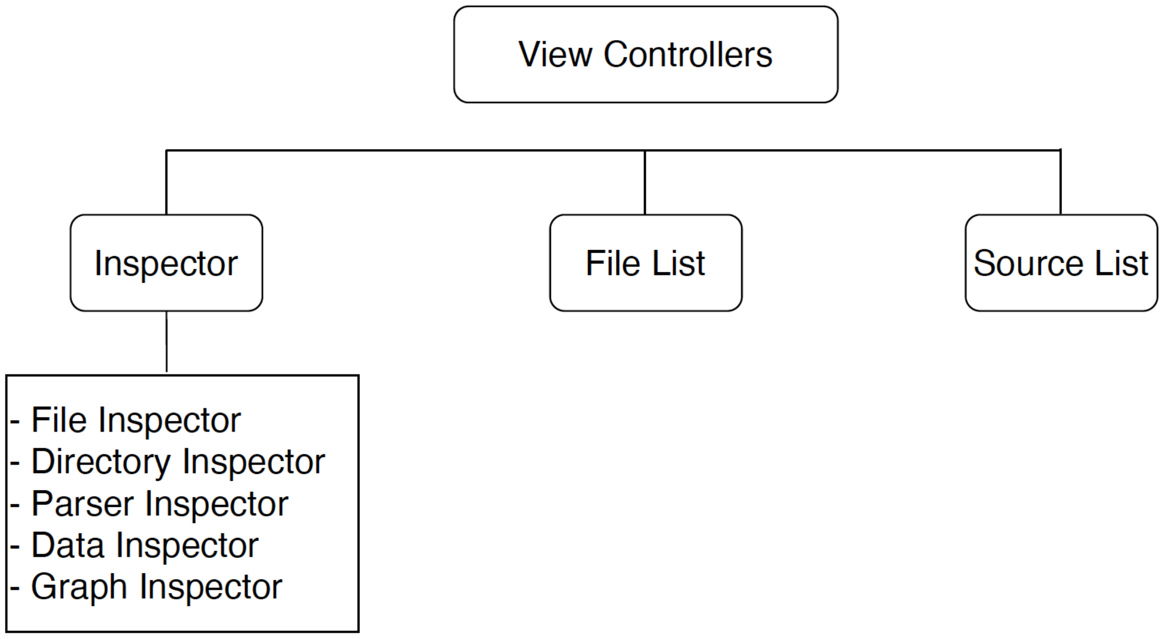


Figure 8: Main Storyboard

For all of the different screens, we have the view controllers to make changes on the file or use the different features of the Graph program.

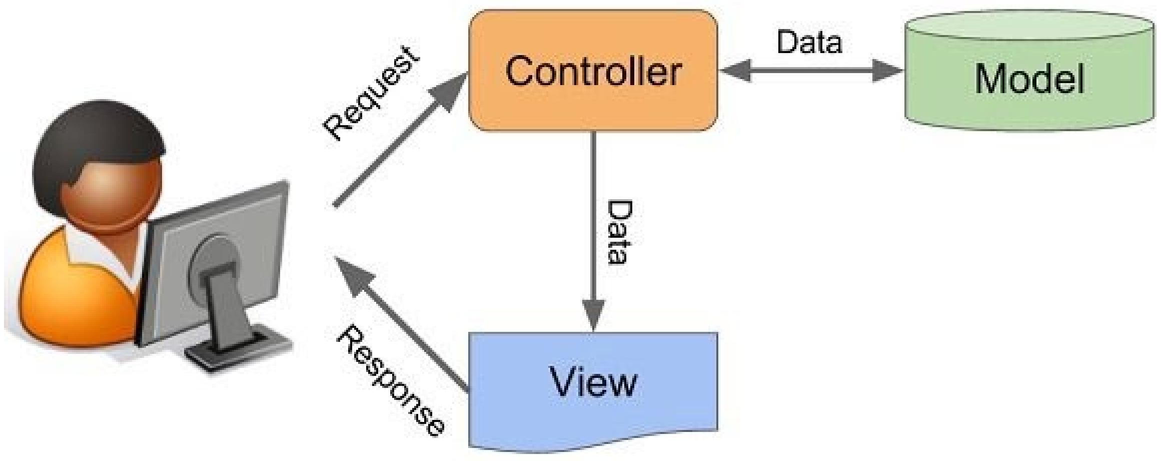


Figure 9: MVC[1]

As described in the System Architecture section of the report, the program uses a model-view-controller. As a result, the way the program interacts with the user could be seen better in Figure 9.

To implement the requested new features, what we needed to do is to handle the models, the views, and the controllers at the same time. For example, when a user clicks on a file for multithreading, it should trigger a function that will start a new thread and parse the file without blocking the main UI thread. Also, we needed some indicators showing the progress of parsing and graphing a file. We needed to add new views such as indicator green and red buttons, but these buttons change color with respect to the progress on parsing the files. So, the architecture is all related to each other and should be handled accordingly.

Quality Analysis

For the software quality of the Graphs program, we implemented some measures. Also, we did extensive testing after finishing the program. During the Sprints of the project, we are completed:

Code Review:

Continuous code review enabled us to detect bugs in each step and implement new features in a structured way to help future developers understand the code fast and integrate new code easily. Also, our feedback mechanism helped us to see different ways of implementing and writing code. Therefore, in Software Development LifeCycle, the most important phase that assures quality reviews enables the code writers to see their approach from different perspectives.

Pair programming:

With the help of pair programming, the way the algorithms are implemented became more efficient, pairs saw the parts the other missed, and it developed a great environment for colleagues to work together.

Functionality Testing:

We tested compliance with the program and components to make sure all the functions are working efficiently.

User Acceptance Testing:

At the end of each sprints, we have sent the updated version of the program to the product owner. Based on the feedbacks we got, we have changed the program in the next sprints. Our most important update was the for the progress bars. Client suggested to use dots instead of the new UI we have implemented to make it look better. So that our updated progress bar has only dots with three different colors: green, yellow, and red. Green refers to ready, and red refers to error, yellow indicates loading.

Results

- **List of features we did not have time to implement:**
  Persisting parsed and cached graphs on disk instead of temporary memory.
  The program does not allow users to only graph nth data points.
- **Performance testing results:**
  With the implementation of multithreading, the performance of the program improved significantly. In the previous version of the program, it was not responding when parsing a large data file. Now, the UI responds to every action of the user while parsing a large file. Therefore, it is not blocking the main thread. On the other hand, since the parser does not run on the main thread, parsing a large file takes a relatively long time. However, since the files are not large in general, the pros of that implementation way more than the cons.
- **Summary of testing (e.g., browsers):**
  We have tested the program using the black-box methodology. It has all of the core functionalities and works as expected. In addition, there is a noticeable performance improvement that is enabled by the multithreading implementation.
- **Results of usability tests (e.g., testing educational software with real students):**
  We have requested different people to test previous and updated versions of the program, and they preferred the updated version since it is more user-friendly. Dr. Hildreth did testing on June 10th, 2021. Based on his feedback, the UI is changed for the progress icons. Also, based on another feedback, the program changed not to import any file other than .dat, .csv, .text. Our first implementation was showing “SKIPPED” icon for non-data files in the UI. It was not needed after last implementation so, we removed it.
- **Lessons learned:**
  Continuous development requires continuous testing and delivery. Therefore, stakeholder feedback is important while developing an application or adding new features to an

existing application. On the other hand, as a team, we realized that pair programming improves the code quality and helps us solve complex problems easily.

- **Features implemented during the field session:**
    - Multi-threading
    - UI for progress icons (Updated twice based on the client's feedback)
    - Multi-graph (functionality and UI)
    - File Filtering,

Future Work

Our team worked on the Swift language for the first time so, we have worked on Swift syntax in the first sprint, and then we have started to develop the program in the second sprint. As of today, we completed all requirements other than the following ones.

- Add an option to "skip data" for large files e.g.
    - Only graph every Nth data point.
- Adding a progress bar with a percentage to show user how much time left for parsing.

Abovementioned work was optional and was known when we started the project.
Below is the features that we discovered while working on the project and thought that could be nice to add in the future.

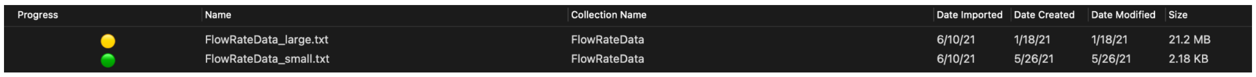
- When the program is closed, the parsed and cached graphs are not saved to the hard drive. If this is implemented, there will be a huge performance upgrade.
- For multi-graph UI, data names can be added to graphs so that users can understand which one is which.
- After Project Finished, we planned to use SonarQube Software Composition Analysis Tool to evaluate the program's effectiveness, but due to the time limitation, we could

not do it. After all the functional requirements are finished, we wanted to check if our code matches the OWASP standards with static analysis and do dynamic software analysis with a Sandbox approach to see the quality of our code and performance metrics. We wanted to create a sandbox environment in SonarQube and check our code on the run. This could be done in the future.

Appendices

The application uses three different indicators to show the parsing stage of the file. The first indicator is Green. It shows that the program finished parsing the file. It can be seen in Figure 10 below. The second indicator is Yellow. This shows that the program is still parsing the file. It can be seen below in Figure 10 as well. Finally, it shows a Red icon to show the program in case of a problem parsing the file.

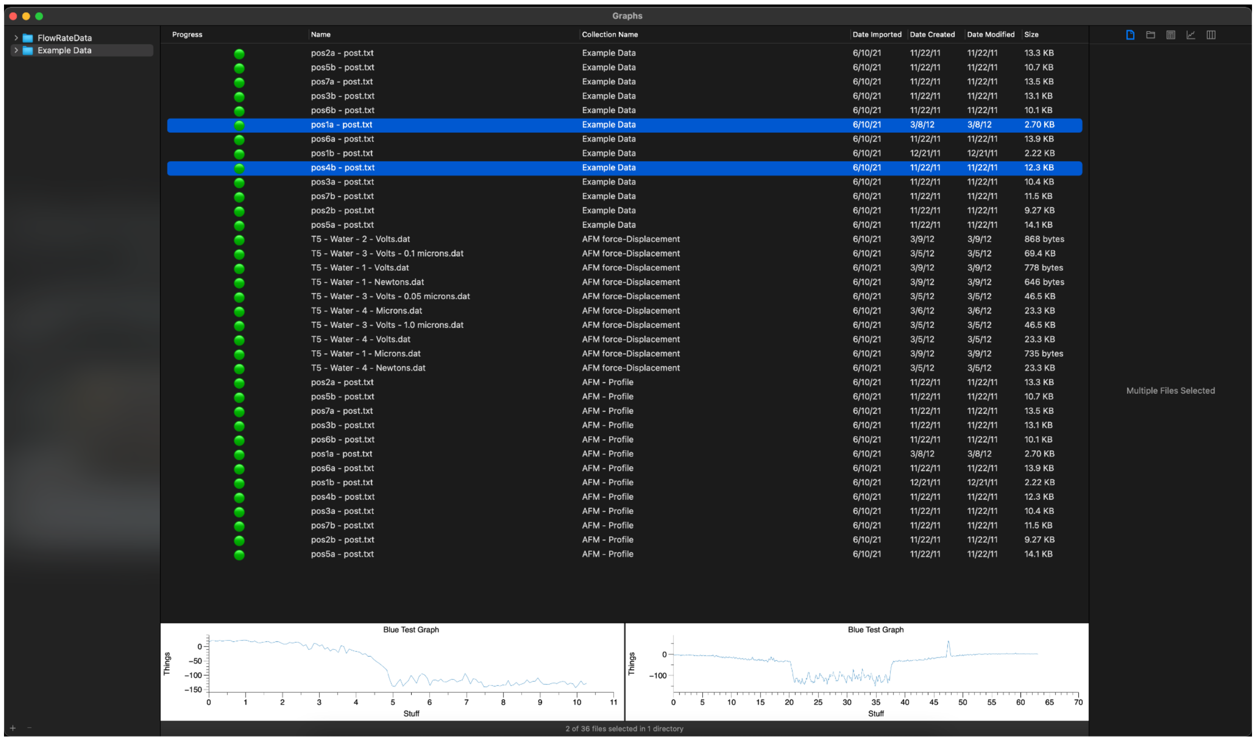
In the previous version of the program, there was no indication of that. So, it will be helpful for users to check the indications.



Progress	Name	Collection Name	Date Imported	Date Created	Date Modified	Size
🟡	FlowRateData_large.txt	FlowRateData	6/10/21	1/18/21	1/18/21	21.2 MB
🟢	FlowRateData_small.txt	FlowRateData	6/10/21	5/26/21	5/26/21	2.18 KB

Figure 10: Progress Icons

Another feature that we implemented is multi-graph. The user can do it by selecting two files in the programs while pressing the "Shift" button. This way, the user can create the multi-graph, as can be seen in Figure 11.



Progress	Name	Collection Name	Date Imported	Date Created	Date Modified	Size
●	pos2a - post.txt	Example Data	6/10/21	11/22/11	11/22/11	13.3 KB
●	pos5b - post.txt	Example Data	6/10/21	11/22/11	11/22/11	10.7 KB
●	pos7a - post.txt	Example Data	6/10/21	11/22/11	11/22/11	13.5 KB
●	pos3b - post.txt	Example Data	6/10/21	11/22/11	11/22/11	13.1 KB
●	pos6b - post.txt	Example Data	6/10/21	11/22/11	11/22/11	10.1 KB
●	pos1a - post.txt	Example Data	6/10/21	3/8/12	3/8/12	2.70 KB
●	pos6a - post.txt	Example Data	6/10/21	11/22/11	11/22/11	13.9 KB
●	pos1b - post.txt	Example Data	6/10/21	12/21/11	12/21/11	2.22 KB
●	pos4b - post.txt	Example Data	6/10/21	11/22/11	11/22/11	12.3 KB
●	pos3a - post.txt	Example Data	6/10/21	11/22/11	11/22/11	10.4 KB
●	pos7b - post.txt	Example Data	6/10/21	11/22/11	11/22/11	11.5 KB
●	pos2b - post.txt	Example Data	6/10/21	11/22/11	11/22/11	9.27 KB
●	pos5a - post.txt	Example Data	6/10/21	11/22/11	11/22/11	14.1 KB
●	T5 - Water - 2 - Volts.dat	AFM force-Displacement	6/10/21	3/9/12	3/9/12	868 bytes
●	T5 - Water - 3 - Volts - 0.1 microns.dat	AFM force-Displacement	6/10/21	3/5/12	3/5/12	69.4 KB
●	T5 - Water - 1 - Volts.dat	AFM force-Displacement	6/10/21	3/9/12	3/9/12	778 bytes
●	T5 - Water - 1 - Newtons.dat	AFM force-Displacement	6/10/21	3/9/12	3/9/12	646 bytes
●	T5 - Water - 3 - Volts - 0.05 microns.dat	AFM force-Displacement	6/10/21	3/5/12	3/5/12	46.5 KB
●	T5 - Water - 4 - Microns.dat	AFM force-Displacement	6/10/21	3/6/12	3/6/12	23.3 KB
●	T5 - Water - 3 - Volts - 1.0 microns.dat	AFM force-Displacement	6/10/21	3/5/12	3/5/12	46.5 KB
●	T5 - Water - 4 - Volts.dat	AFM force-Displacement	6/10/21	3/5/12	3/5/12	23.3 KB
●	T5 - Water - 1 - Microns.dat	AFM force-Displacement	6/10/21	3/9/12	3/9/12	735 bytes
●	T5 - Water - 4 - Newtons.dat	AFM force-Displacement	6/10/21	3/5/12	3/5/12	23.3 KB
●	pos2a - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	13.3 KB
●	pos5b - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	10.7 KB
●	pos7a - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	13.5 KB
●	pos3b - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	13.1 KB
●	pos6b - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	10.1 KB
●	pos1a - post.txt	AFM - Profile	6/10/21	3/8/12	3/8/12	2.70 KB
●	pos6a - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	13.9 KB
●	pos1b - post.txt	AFM - Profile	6/10/21	12/21/11	12/21/11	2.22 KB
●	pos4b - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	12.3 KB
●	pos3a - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	10.4 KB
●	pos7b - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	11.5 KB
●	pos2b - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	9.27 KB
●	pos5a - post.txt	AFM - Profile	6/10/21	11/22/11	11/22/11	14.1 KB

Multiple Files Selected

2 of 36 files selected in 1 directory

Figure 11: Multi-Graph

This program is versioned as 1.1 by incrementing 0.1 from the previous version. It can be changed from the general settings of the Xcode environment. Versions are pushed with another branch, and after clients like the final version, it will be pushed to the master branch in the Github environment.

References:

[1] Shi, L. (2018, December 2). *What is MVC (Model View Controller)?* Medium. https://medium.com/@louis.shi/what-is-mvc-model-view-controller-834271cdf65f.