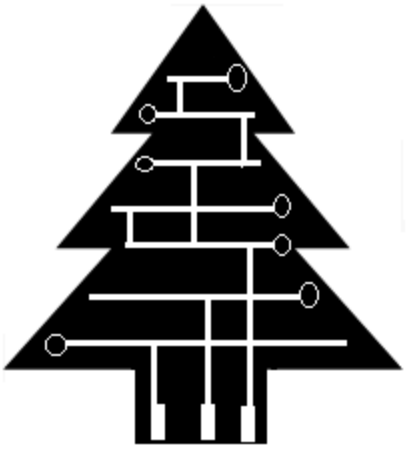

Final Report

April 10th, 2018

Sponsor

Dr. Andrew Richardson

Dr. Mariah Carbone

Mentor

Ana Paula Chaves Steinmacher

Team

Sam Beals, James Beasley,

Andrew Greene, Joseph Kelroy

Table of Contents

Table of Contents **2**

Introduction **3**

Process Overview **4**

- Team Member Roles 4
- Development Process 5
- Task Management 6
- Bug Tracking 6

Requirements **6**

Architecture **7**

Implementation **9**

- Home Screen 9
- MetaData Screen 9
- Graph Screen 10
- File Directory 11
- View Screen 12
- Data Files 12

Project Timeline **13**

- Major Milestones 13
- Timeline Breakdown 13

Testing **14**

Future Work **14**

Conclusion **14**

Glossary **15**

Appendix A **16**

- Hardware 16
  - Android Device 16
  - Computer 16
- Toolchain 16
- Set up 17
  - Setting up a Developer Environment 17
  - Installing CO2 Flux 17
- Production Cycle 18

Introduction

Global climate change is an issue that will affect every person on the planet. It is widely acknowledged that climate change is driven by rising levels of atmospheric carbon dioxide (CO2) resulting from fossil fuel burning. The importance of research into climate change cannot be understated, but when people think of data associated with climate change they typically think of the carbon footprint of our cities. They often forget that the environment and its carbon cycle hold data that is just as, if not more, valuable. Our goal is to ensure that ecologists can collect data as quickly, efficiently, and as easily as possible from the environment.

The Richardson-Carbone Lab is operated by Doctor Andrew Richardson and Doctor Mariah Carbone at Northern Arizona University. The Richardson-Carbone Lab studies carbon cycling in forest ecosystems. In a nutshell, the research is used to understand the balance between carbon uptake (photosynthesis by plants) and release (respiration both by living and growing plants, as well as by microorganisms decomposing dead organic matter in the soil). Through the use of modern technology, we made the collection of CO_2 data from trees a streamlined process with minimal cost and intuitive user interaction, providing users the ability to collect, visualize, and interpret field readings.

Currently, the Richardson-Carbone Lab has to use a cumbersome set of equipment to conduct their research. This method of data collection is slow, not very portable/durable, and it is not the easiest to view out in the field. The solution we developed made the entire process fast, portable, and easy to use while also providing important meta data to the researchers in real time. The application allows any Android device to communicate with the LI840A gas analyzer. The data is streamed into the app and displayed in real time. Any of the data that is logged in the app can be saved and transferred out of the app via email. Additionally, the data saved by the app can be reviewed and deleted at any time by the user. This document details the process taken to implement this solution, while also containing all information necessary for future programmers to make adjustments.

Process Overview

Team Member Roles

Our team had specific roles throughout the course of the project. James Beasley was Team Leader and Team Communicator. As the team leader, it was Jimmy's responsibility to make sure clear communication was maintained throughout the group and that reminders are occasionally sent out in regards to group meetings, deadlines, and sponsor requests. As the team communicator, it was also Jimmy's responsibility to communicate with the sponsor and mentor about any questions or concerns the team had, as well as relay their responses to the rest of the group.

Andrew Greene was the Project Architect. As the Architect, it was Andrew's responsibility to enforce technical standards. All decisions will still be left to the decision making process. The architect will be in charge of organizing coding standards, tools, and platforms. Once decisions were made, any changes or suggestions will be communicated to the client and mentor. The architect will ensure that design decisions will be followed during implementation.

Joseph Kelroy was the Team Recorder. As the recorder, it was Joseph's responsibility to record all necessary information from each team meeting. This includes what each team member worked on the previous week, what each team member plans to work on the following week, who attended the meeting, and any decisions made. Lastly, information kept in this record was easily available to all other team members.

Sam Beals was the Release Manager. As the release manager, it was Sam's job to determine which features should be in each release version, ensure that each release compiles into a complete and functional package, and prepare the software for delivery to the sponsor with proper documentation. It was also Sam's responsibility to ensure each release has the desired features, and if it did not, request the features from the responsible team member.

Development Process

Along with the specified roles listed above, all team members were programmers. All required code was divided up amongst the team members as evenly as possible, with consideration for each member's abilities and preferences. In order to go about fully implementing our product, we decided to split tasks and responsibilities for each member based on the screens assigned to each of them. The application consists of four main screens and a home screen. Each screen was fully realized by an individual member. Andrew created the MetaData Screen. Jimmy created the Graph Screen. Sam created the File Directory Screen. Joey created the View Screen. Of course, there were some aspects of the functionality that are not entirely contained to a single screen within the app. To handle this, we set up a few measures for the group to prevent issues arising from this.

In order to avoid any discrepancies that may have arised from shared data and values between different screens, all of the work conducted by individual members was done in the same location. When any issues in the code or confusions in the design arose, it was discussed and settled in the same instance. This creates ease of communication and maximizes efficacy in regards to decision making and bug fixing. Prior to any initiative being taken on a new aspect of the program, the parameters of its design were decided and agreed upon. Any work that was done outside of the group meetings was approved over Discord. Discord is a Group Chat Application on which the entire group was connected. This allowed all members to be involved in decision making and prevented any members from being left in the dark in regards to new developments or changes in our design. It worked very well and served as our team's main point of communication throughout the development of the project.

Our development cycle was based off of the Waterfall Model. Our entire project had a schedule, which can be seen below. During the development phase, features were implemented based on priority. The most functional and crucial features were implemented first. Pushes to Github were made on a weekly basis. These features were based on weekly client meetings. Here we would discuss the progress we have made, and get client feedback. Every week the team would have their tasks finished 2 days ahead of the weekly meeting time. This way, it allowed for code review. The code would be checked to make sure that every task and feature was implemented

as intended. It also allotted time for bug testing. As new features were added, bugs appeared. It was at this time that any major bugs seen would be fixed. Once everything was thoroughly checked, a push would be made to Github.

Task Management

Each team member was assigned about 2-3 programming tasks per week. These were managed by weekly task reports. The app's navigation was on the forefront of the group's priorities, but tasks became more individualized as the project moved along in development. The File I/O system and the UI were among the first things to be completed. After they were developed, we moved into the phase where most of the features the user interacts with will be implemented. This was where everyone programmed their own screens for the duration of the semester. After all of the required features were implemented, the group worked on adding additional features as requested by the clients.

Bug Tracking

We then moved on to the testing and debugging phase, where we fixed any issues occurring within the app as well as improving the polish on the user interface and user experience. Throughout the course of the semester, we had a Bug Tracking Report. This was a Google Spreadsheet where our team would report any bugs found, whether or not it was found on their own screen or someone else's. During the development phase, bugs would be fixed as soon as possible. Each bug found had a name, location, description, severity rating, and status. Bugs were generally fixed as soon as they were found. When it came to the Testing and Debugging phase, rigorous tests were used to find more obscure bugs.

Requirements

When we first began this project, we had some very clear functional and non-functional requirements. The functional included :

- Real-time Data Streaming
- Live Data Visualization
- Data Transference

- Statistical Analysis
- Metadata Creation
- GPS integration
- Image saving
- File management

For non-functional requirements we had Mobility and Compatibility, meaning that our application needed to run on a mobile platform but still be able to communicate with LI-840A gas analyzer.

By the end of the project, we had maintained pretty much all of the requirements as specified in the beginning. The only change to our list of requirements was that we had picked up some additional functional requirements. We ended up adding Delete Functionality, External Image Saving, and Zooming and Scaling when looking at data. to the list of requirements. All of these were met with little issue and are present in the current product.

Architecture

Our application was divided up into multiple screens, each of which were responsible for specific functionalities that served to meet the requirements as specified by our clients. Although each screen is specialized to perform specific tasks, they interact with and rely on one another in order to perform their tasks.

The first screen that the user sees is the most basic: the Home Screen. This screen consists of just two navigational buttons. One button will take the user to the beginning of the data collection process which on the Metadata Screen, while the other takes the user to the File Directory. This is the screen that the user will see whenever they boot up the application, and it also displays the app logo and version number.

The Metadata screen marks the beginning of the data collection process for the user in this application. The purpose of this screen is to gather user-provided data on the subset of data being collected itself. This is vital for researchers that are looking to re-examine data long after it has been saved, giving them reference for when they review. The data from the metadata screen is temporarily saved and taken to the next screen, the Graph Screen.

The Graph Screen is perhaps the most important one in this app, as it contains the means by which the user can record incoming data. It contains a graph object that updates in real-time as data is being streamed in from the gas analyzer. The user can re-record data until they are satisfied with it and ready to finish. When they choose the “finalize” option, the metadata, image (if one was provided), and the graphing data are all saved in the app’s permanent storage. Data saved this way can be viewed and managed in the next screen, the File Directory.

The Graph Screen relies on the Graph Manager Object, an object we designed specifically for this page. It is responsible for communicating with the data that is being streamed in from the gas analyzer. It parses it and passes it to the appropriate graph on objects on the graph screen, as well as saving it for later storage. This object is essential to the performance and functionality of the Graph Screen as a whole.

The File Directory screen is responsible for all of the data management that occurs within the application. It presents the user a list of all of the saved data sets in the app. One can select any number of files and delete them or email them. If the user chooses to view a file, the screen takes the data for the file and transfers it to the View Screen.

The View Screen displays all of the data that is within a data set. The screen shows the user the image related a data set, along with the metadata and all of the graphing data. This screen

allows the user to create a subset of the data set. If a data set is made and saved, the new subset is saved and displayed on the File Directory screen.

The View Screen uses the Line Graph Object, and object that is also used by the Graph Manager Object. The function of the Line Graph Object is to create savable objects that represent fully written and constructed graphs. Since the view screen isn't recording anything but rather just presenting old data, it uses these objects to reconstruct the graphs and display them on screen.

Implementation

Home Screen

The Home Screen just acts as a simple point of navigation to either the MetaData Screen or the File Directory Screen. Any time a user opens the application they will start here. Like all of the other screens, this screen contains a Java and XML file. If you would like to change the layout of this page you would need to edit the activity_home_screen.xml file. If you want to change the functionality of the page you would need to edit the homeScreen.java file.

MetaData Screen

If you would like to change the layout of this page you would need to edit the meta_data.xml file. If you want to change the functionality of the page you would need to edit the metaData.java file. In the onCreate, all of the text fields and listeners are created. The listeners are for field validation and GPS. Time and Date all created, and any shared preferences are loaded. All of the main functions and helper functions are well documented inside the code.

The MetaData Screen is where all of the user input occurs. Upon pressing the "Start New Data Set" Button on the home screen, the user will be directed to the MetaData Screen. This screen has eight Text Fields, three of which are required. These text fields were all specified by the client. All of the other fields are optional fields. All fields have character restrictions and a character limit on the length. The character restrictions, as well as other attributes for the text fields are all defined in the XML file meta_data.xml. The required fields are Operator Name,

Site Name, and Sample ID. The finish button is disabled by a validation function until the required fields are filled in. The optional text fields are, Temperature, Comments, Longitude, Latitude, and Elevation. If they are left empty, they will be listed as “NA” when saved. Operator Name and Site Name will both be filled in with the values of the last data set using shared preferences. If the user goes to the Graph Screen, and realizes that they made a mistake, then they can go back. The MetaData Screen will be reloaded with the strings using shared preferences and a flag stored as an extra.

Time and Date are always filled in automatically. If there is a GPS signal, Longitude, Latitude, and Elevation will be filled in automatically for the user as a background process. Lastly, the user has the option to take a picture. This opens a camera intent, where the user can take a picture. The image is returned as a bitmap, and is also shown in the preview window. In addition, all images taken by the camera are also saved in the device’s gallery. The dataset itself will only save the most recent image taken, which can be seen in the image preview.

Once the user has entered in the required fields, the Finish button is activated. When the “Finish button is pressed, the strings are taken from all the text fields, and saved as extras. The Image is saved as a bitmap extra. All of this is passed to the Graph Screen. The “Cancel” button will take the user back to the Home Screen. All metadata that has been entered on the Metadata Screen will not be saved.

Graph Screen

The graph screen is where any kind of communication with the LI-840A gas analyzer occurs. After the analyzer is connected to the Android device, data will automatically start streaming into the application and appearing on the live graph readout. If the analyzer is ever disconnected while on this screen, a notification will briefly display alerting the user.

On this screen, the user can record a new data set and manipulate the graph through zooming and scaling, all while maintaining the ability to switch between all four data values offered by the analyzer. After the user has finished recording a data set, they can press the “Save and Exit” button to proceed to the file directory, or they can begin recording a new data set which will overwrite the previous one.

After the “Save and Exit” button has been pressed, all the graph data will be saved to the device, along with its corresponding metadata file and image file. This new data set will now be visible in the file directory.

File Directory

Any sort of data management will be performed on the FIle Directory Screen. The application allows the user multiple means of managing the data that has been saved via CO2 Flux. The screen will display all of the files currently saved in the app, and sorts with the newest sets being the highest on the list. When performing operation on files, one must first select them. This is easily achieved by simply pressing on the item on screen, highlighting it green. When an item is highlighted, any selected operations will be performed on it.

The foremost operation that one can perform is file deletion. If the user feels that certain files are unnecessary or causing clutter, they can select them by pressing on them and choose the “Delete” option located in the bottom left of the screen. The user will be prompted with a warning message asking them to confirm their choice. This is done to ensure that any permanent file deletion is completely intentionally and any accidents can be easily avoided.

The other primary operation that can be performed on the File Directory screen is the “View” option. It should be noted that it can only be selected when only a single file has been highlighted. For the user’s convenience, they easily narrow down the list of files and find ones they are seeking by using the Filter functionality on the screen. By filling in characters in the text field located in the top right of the screen and pressing the “Filter” button, they will see only files with names that contain the text the user filled in. Once “View” is selected, they will be taken to the View Screen to review their selected data.

In order to fulfill the requirement of being able to transfer data out of the application, we implemented Email client functionality. When any non-zero number of files are selected on the File Directory screen, the user may choose the “Email” option located in the bottom right. At that point, they will be prompted to select from a menu of email client applications that are currently installed on the device being used. If the user’s device does not have an email application

currently installed, they will need to download one before this part of our application can be used. Provided the user has one installed and chooses it, they will be taken out of CO2 Flux and to the app they chose. All of the files are attached automatically, all the user needs to do is fill out the To Address and the subject and their job is done.

View Screen

If you would like to change the layout of this page you would need to edit the view screen xml file. If you want to change the functionality of the page you would need to edit the view screen java file. The on create is where all the objects are created to build this screen. It creates all the graphs and reads a file to put all the information to the screen. If you want to change any of the functionality beyond what the page is originally pulling up there are many helper functions with descriptions on what they do.

Data Files

Writing to a data file is done in two places, originally they are created and written in GraphScreen.java and then later they can be edited in the viewScreen.java. The graph screen initially writes all the files and names them and then the view screen reads these files and if the user chooses they are rewritten to. If you want to change what is being written to the file after the creation, change the code after the file is opened for reading on the graph screen. The view screen can write to files two different ways. It can write to the original file by changing comments and it can create a copy of these files with a subgraph. If you want to change what is being written to the original file it is in the function update original comments. If you want to update anything with subgraph writing it will be in any function with the name subgraph in it.

Overall the only difference in what we were going to originally build from what we build is how we use the data. We thought we would need a data object to pass the data to each page and easily be able to pull it and save it. It ended up being easier than this though where android studio allowed us to pass the information from page to page and we wrote the information to files instead of objects. This really took out the middle step of the object whee we could use the data files that we were passing out of the application. Besides this change, we implemented a lot of functionality that was not originally thought out by our clients. They suggested some

additional features that they found to be useful after we started showcasing the application and were able to get them more than they originally wanted.

Project Timeline

Major Milestones

Semester 1

October - November: Requirements Acquisition

November: Technical Feasibility

December: Early Prototype

Semester 2

January - April: Development

April: Testing and Debugging

May: Product Delivery

Timeline Breakdown

The first semester was all about project set up. In october we started our meetings with our client to gather all the requirements we would need for this project. Once all requirements for the project were established the team divided up the requirements to plan on how to go about implementing these requirements. With this came the feasibility proof of our application. We all decided how we planned to implement it and if it was feasible. Once we decided we had everything planned out we moved into creating the prototype as a team. When the second semester started we actually had to have a more concrete way of creating this application. We set up a chart on exactly what we wanted done and when we wanted it done by. The easiest way we found to work on the application was by splitting it up into screens. Andrew worked on the metadata screen, Jimmy worked on the graph screen, Sam worked on the file directory screen, and Joey worked on the view screen. This way we all could develop at the same time. We alloted time for formal testing in April. Finally we packaged everything we worked on up and delivered it in may.

Testing

The bulk of the testing for this application came as we built. Whenever we put something new into the application we would stress test the durability of it by trying to break it. The way this was done was inside the source code and inside the actual application we would input dummy data. Once we thought that whatever we worked on was fully functional we moved on to a new feature. We tested this way because we made time in April to formally test. We went about integration testing and unit testing the same way as before but way more in depth. Usability testing was done differently though. We had members from the Richardson- Carbone team and also random users go through and test our user interface and user experience. We did this to get a well rounded background of testers to ensure the application was as simple and intuitive as possible. We used time tests and a pass or fail system for the users and once they were doen testing we got feedback to make changes if need be.

Future Work

This application will be used daily by many people so new ideas to improve the application will be inevitable. Two thing that our clients mentioned that might be useful in the future are compatibility with other analyzers and custom metadata. Li-core, the company that makes the LI-840A, makes many other analyzers and the clients would like to have this application be compatible with more of their products that they use. Also they would like to have custom metadata functionality to be parsed into the CSV file. Another thing talked about for the future of the application was more statistical computation to be able to do even more in ap versus out of the application.

Conclusion

We developed CO2 Flux, a mobile application that aids researchers in ecological data collection. This was developed with the intention of converting of all of the existing functionality in our client's current software and transferring it to a mobile platform, along with additional

features. The client's current setup involves using a laptop that runs their software, which is of course bulky and cumbersome and does not lend to quick work routine.

By developing a mobile alternative for our client, we significantly reduced the setup and teardown time in their work routine, eliminating a considerable amount of wasted time. Some of the additional features of our software that were not present in the previous software includes:

- Data Transference
- In-app Data Review
- Subset Creation
- Statistical Analysis
- Real-time Data Streaming

Our application will afford our clients a much simpler and easy way to go about their ecology research. Fortunately, the reach of our product's impact is not limited to just our clients. Our application is open-source and will be publicly accessible to any body who may be interested in using it for their research. Furthermore, our clients will be publishing a paper detailing the uses and functionality of our product, disseminating the knowledge our work to other peers. The app will be able to modified by others to their liking after our work is done, making the future of our product limitless.

More information on this project and our team can be found by visiting the project website. https://www.cefns.nau.edu/capstone/projects/CS/2018/CO2Software/

Glossary

Android - Mobile Operating System developed by Google
Android Studio - Software used for Developing Android Applications
Java - Popular computer-programming language
XML - Markup-Language used for refining and formatting documents
Extra - Data passed between different screens in an app
WaterFall Method - A technique for distributing and performing work on software in the context of a group
Discord - Application used for sending messages and voice chatting among groups

GitHub - Online repository for updating and storing publicly accessible projects.

User Interface - Display in the application with which the user interacts

User Experience - The experience the user has while interacting with an application

Appendix A

Hardware

Android Device

CO2 Flux was tested using the Lenovo Tab 4, but should operate on any Android device capable of running Android 7.0 or higher. The interface for CO2 Flux was designed for use in portrait screen orientation.

Computer

Any computer capable of running Android Studio can produce updates for CO2 Flux. Android studio is supported on Windows, Mac, and Linux platforms, and minimum specifications can be found by visiting the official Android developer website.

Toolchain

The following technologies were used to develop CO2 Flux, and will need to be installed and properly configured with Android Studio in order to make adjustments:

1. Android Studio

   Android Studio is the default method of developing Android applications, and is produced and maintained by Google. More information on Android Studio can be found using the following link: https://developer.android.com/studio/

2. Android GraphView

   Android GraphView is an open source library for implementing visual graphs in your Android application. It is specifically used in CO2 Flux by the line graph object. More information on Android GraphView can be found using the following link: http://www.android-graphview.org/

3. UsbSerial

   UsbSerial is a serial port driver library for Android applications. It allows easy communication between an Android application and hardware which reads in and exports data through a serial port. More information on UsbSerial can be found using the following link: https://github.com/felHR85/UsbSerial

Set up

Setting up a Developer Environment

Please read the Hardware section to make before trying to set up an environment. Over time, changes will be made to this application. In order to set up and edit the code, you must have Android Studio. Android Studio is a free development environment that this project was made in. Make sure that Android Studio is up to date. CO2 Flux is an open source project available on Github. There are 2 options for setting this up. Option 1: Manually Import the Project. CO2 Flux is an open source project available on Github. Download the project off the Github Repo: https://github.com/CO2Software/LI840AInterface.git. Next, import the project into Android Studio by finding the project in your Downloads Directory. You may want to then move it to a safer directory, such as AndroidStudioProjects. Now changes can be made. Option 2: Import The Github Repo using Android Studio. When Android Studio Opens, click “Check out project from Version Control”. A window will be opened, under Git Repository URL, copy and paste this line: https://github.com/CO2Software/LI840AInterface.git. This is a link to the current Github Repo. For the Parent Directory, it should be under AndroidStudioProjects, for example, mine is C:\Users\Admin\AndroidStudioProjects. For Directory name, LI840AInterface. Now changes can be made. Please note that this is just a Clone of the original code, which you may still upload to Github. To test the changes, under the “Run” tab, click “Run ‘app’”. To push the changes to Github, To push just the APK, under the “Build” Tab. click “Build APK(s)”. Once it has finished building, a popup will appear. Click “locate” and this will show you the new APK in a directory. You can upload this APK, or any other changed files, to the Master Folder on Github, using the same link above. You may also create a new branch if desired.

Installing CO2 Flux

Please read the Hardware section to make before trying to set up an environment. If you are just setting up the app on an Android device, follow these steps:

1. The installation requires downloading the APK titled “CO2Flux.apk” from Github. The APK can be found here:

   https://github.com/CO2Software/LI840AInterface/blob/master/CO2Flux.apk

2. Download the APK onto the mobile device. If the download is not available on a mobile device, download the APK on a computer. Then, email the APK to the mobile device, where it can then be downloaded.
3. Android may give warnings about the installation. Make sure that the permissions are allowed. It will ask if you would like to Install CO2Flux, press “Install”. Then press “Done”. Navigate to “All Apps”, this is where CO2 Flux will be found.
4. After Installation is successful on the device, open the application. You will be prompted to allow for permissions. The required permissions are Camera, Location, and Storage. The app will not allow you to proceed without these permissions. Once the permissions are set, installation is complete.

Production Cycle

Once your environment is set up and you have all files needed in android studio you can edit the application. You will mostly be working with the .java and .xml files of all the pages. If you would like to make an update to the application, edit the code you would like to change. Then click run on the top navigation bar(or click the play button). A window will pop up with the device you want to run it on. If you want to run it on an emulated device on the computer, either set up a device or click whatever one you have already set up. If you would like to run it on a physical device, make sure it is connected to the computer then select the device. If there are no errors in the compilation of the edit you made to the application then the application will pop up on the device and you can see the changes.