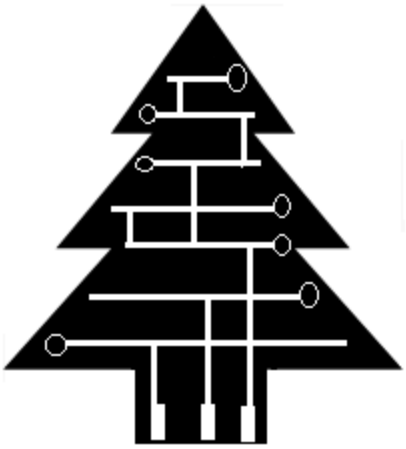

CO₂ SOFTWARE

Technical Feasibility Document

October 26th, 2017

Sponsor

Dr. Andrew Richardson

Dr. Mariah Carbone

Mentor

Ana Paula Chaves Steinmacher

Team

Sam Beals, James Beasley,

Andrew Greene, Joseph Kelroy

Table of Contents

1. 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 (CO_2) resulting from fossil fuel burning. The importance of research into climate change can not 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. Ensuring ecologists can collect data as quickly, efficiently, and as easily as possible from the environment is the goal that we hope to achieve.

We will be developing this software at Northern Arizona University, with the guidance of our sponsors Doctor Andrew Richardson and Doctor Mariah Carbone. 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 aim to make 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.

There are many problems with the current method of data collection. The research requires mobility and durability for the outdoors as the data collection will occur at many different sites. Currently, the Richardson-Carbone Lab has to use a cumbersome set of equipment to conduct his research. The stock device's software can only be run on a Windows PC as there is no mobile application available that satisfies the requirements of his work. Bringing a laptop to the research site is slow. The laptop must be powered up, connected, and then the data collection software must be manually run. The software on the gas analyzer has been unchanged for 15 years and only returns raw data for the user. This is difficult to read and must be analyzed before it can be of any use to the researcher. Once the data is collected, the laptop must taken to the next site, where the process is repeated. All of these problems together slow down the research collection process.

The proposed solution will make the entire process fast, portable, and easy to use while also providing important meta data to the researchers in real time. This document will outline our solution to the problem of slow field data collection, as well as break down which existing technologies we will be implementing and why we chose them. We will start by addressing the needs and challenges of the problem. We will explore the different technologies we have chosen, and what specific parts of the problem they will be solving. Finally, we will explain our integration process and how all the individual parts will interact to form one complete piece of software.

2. Technological Needs and Challenges

Creating a system which allows for the simple collection and interpretation of field data poses several problems. In this section we will outline each of these issues and state the types of restrictions they may pose on our solution.

First, the solution needs to be simple to operate. We expect the standard users of our system to be ecologists, and other environmental scientists. This means that they need to be able to interact with the system in a way that is comfortable and familiar to average computer users. Interacting with the system through the use of a terminal, or similar method, is not an acceptable option.

Second, the solution needs to be portable. This software is expected to be used for data collection in the field, meaning that it will need to be moved many times between various locations throughout the day. Requiring the user to carry around a large, bulky device such as a powerful laptop computer is unacceptable due to its large size.

Third, the solution needs to be compatible with the LI-840A Gas Analyzer. This device is a popular tool scientists use to collect various readings from the environment. The device will be used as the standard for our development process. Ensuring the accurate communication of data between the device and our chosen development platform is an absolute necessity.

Fourth, and arguably most import, the data readings need to be displayed in a way that allows for fast interpretation and statistical analysis. This will be done by displaying the data using line graphs, and utilizing the power of the device for quick statistical computations. This will be of large value to the end user, and will greatly expedite their workflow.

3. Technology Analysis

Our solution will require the use of several existing technologies. In this section, we will break down each technology, stating the issue it will be solving, the possible choices we have to pick from, which specific technology we have decided on, and the proof that is it technically feasible.

3.1 Operating System

For this project, the end product will be an accessible mobile app that our client can use in conjunction with their gas analyzing hardware. The app software will support a direct connection to the hardware, eliminating any unnecessary additional data transfer points. Therefore, the app that we develop will have to be run in an operating system that allows for a clean user interface

while also being completely functional with direct-serial connection. The app's interface will involve a layout of the data visualization based on the output of the gas analyzer. The nature of this project calls for an operating system that affords us generous customizability.

3.1.1 Alternatives

As far as the approaches to selecting the operating system for the app goes, there are three primary options from which to choose. There is iOS, which is developed by Apple and run on all of their products. There is the Android operating system, which is open-source and run on more devices than any other OS in the world. Lastly, there is the JBoss, which is the platform on which Java-based applications are developed, deployed, and hosted.

iOS provides us with a few distinct advantages, but with many disadvantages as well. iOS utilizes interface features that are immediately accessible to most people and would lend itself to giving our product a sharp look. Additionally, Apple only has a few select products out at one time, making the different devices that need to be accounted for quite small in number.

The downsides of using iOS are also quite considerable. Apple provides many premium products, and as such, they are very expensive. This would make simple field testing for the app costly as even older Apple product are not cheap. Apple also has a notoriously stringent review policy for apps that are put onto the App Store. There is a lengthy review process that could potentially cause scheduling issues for the project. This issue also applies to whenever an update would need to be pushed out for the app at any point past its deployment. This loss in time could offset any time saved during the user interface development.

The next operating system to be considered is JBoss. Not unlike Java itself, JBoss affords its users with a very charming straightforwardness that allows for ease of access and usability. There is also a good deal of flexibility in the types of apps that one can produce on the Java platform. It can be run on any device that can run Java, which means it provides the user with a large array of devices from which to choose. Based on the needs of the client, we can develop the app as either an applet, web application, or a desktop application. A web app could be accessed via a mobile device and perform the functions requested by the client.

Despite its flexibility, there are some interface and performance issues that greatly hinder the JBoss prospect. The look and design of Java apps are not as modern or slick as one may expect in this day and age. The interface design tools are a bit dated and prioritize function over fashion, which is a red flag for an app that has a large focus on cleanly and smoothly displaying data to its user in real time. Java-based apps also often have issues with performance speed when compared to other operating systems. Since the app we are developing needs to be gathering, logging, and visualizing data in real time, this a big concern and more or less is the reason for which we are opting to not use JBoss.

The last OS to discuss is Android. Android has numerous advantages that are very pertinent to the team and its project. Android is by far one of the most prevalent operating systems in the world, and as such has a wealth of easily accessible information pertaining to app development. One can easily find various API's readily available, online tutorials, and discussion forums in which one can find the answer to their problem. Additionally, app interfaces are easily developed using tools such as Android Studio, which also doubles as a simulator for application testing. Play Store, the platform on which Android apps are hosted, has a very lenient process by which apps are submitted and accepted. Uploading and updating the app will be simple and easy. This saves a lot of time, especially during the deployment part of the development cycle.

Although there are many advantages to Android, it does have its fair share of downsides. The most glaring is its sheer number of devices that run it. As far as tablets and smartphones go, there are a multitude of different screen sizes, resolutions, and other hardware specifications that serve to make UI and the app development in general much more complex and laborious.

3.1.2 Chosen Approach

Ultimately we decided to go with Android as the platform on which we will develop our app. The data visualization libraries available in Java are easy to implement and provide all of the features we need. The roadblocks presented in the iOS route appear to be very large concerns for the future that can become costly in both time and money. Although JBoss has its perks, having to conform the project into the constraints of a web application created too many inconveniences to the client and complications to the development team. In the table below, the ratings are on a scale out of 5. The most important factors in making our decisions are listed on the left column, with their accommodations from the respective operating systems being rated across their row.

	iOS/Swift	**Android**	**JBoss**
UI Development	3	4	2
Accessibility	1	5	4
Performance	4	3	3
Customizability	1	4	4

Categories rated on a scale of one (worst) to five (best)

3.1.3 Proving Feasibility

To prove that Android is feasible we have downloaded drivers that allow for the Android device to communicate with and receive data from the gas analyzer. We have successfully connected a device to the gas analyzer and transferred data from it in real time, as can be seen in the screenshot below. The screenshot displays the output from the gas analyzer being uploaded to

the mobile device, complete with timestamps and labels. With this data, we can simply develop an application that utilizes data visualization and statistical analyses libraries in tandem with a clean user interface in order to satisfy the requirements of the project.. This shows that the Android OS is a feasible platform on which to build our project.

3.2 Graph Library

One of the major benefits of our application is that data will be presented in a convenient and easily readable way. In the field, it can be difficult to do statistical work on large sets of data. It can often be difficult to determine quickly if the readings being taken are viable data sets. For the end user, having the data presented in the form of a graph, with real time reporting would lead to more meaningful data, and a quicker process. Instead of writing our own data visualization library, we have determined that we should instead research libraries that can fit our needs and be integrated into an Android application.

3.2.1 Alternatives

We have researched many different graph Libraries and have narrowed the list down to three choices: GraphView, AndroidPlot, and MPAndroidChart. These are all open source libraries with full documentation on Github.

AndroidPlot is a good library because of its wide selection of aesthetically pleasing graph presets. The problem with this library is that aesthetics is the only area it excels in. It is not a very robust or capable library. It is very short on documentation and has the least features of the three with no realtime or customization.

MPAndroidChart was another strong contender because it is a good library for situations where the user needs graphs displayed in visually appealing ways. It has many graphs and can handle a lot of data. It also is very customizable from the colors to the fonts to the layout. MPAndroidChart has many functions but it lacks real time display. This is not acceptable as it is a crucial part of our project.

3.2.2 Chosen Approach

After we finished our research, we decided on using GraphView. Of all the libraries, graphView definitely has the most functionality. It is the most customizable library by far with the most features. It also has the most important feature that the others were missing and that is real time display. On top of that you are able to have a view of only specific parts of data and it has zoom and scaling capabilities. The following table lists the three graphing libraries we considered and whether or not they contain features we deem to be essential.

Screenshot of GraphView

	GraphView	**AndroidPlot**	**MPAndroidChart**
Real Time	X		
Customizable	X		X
Zoom/scale	X	X	X

3.2.3 Proving Feasibility

To demo GraphView, we will set up the gas analyzer and blew into the tube to change the CO2 levels. Since one of the main goals of our project is to have real time display, this demo will clearly demonstrate all the functionality of our graph library. When the data is plotting we will be able to show the scaling capabilities and how we can manipulate the graph to change the x and y views.

3.3 Android Serial Connection Library

Our solution hinges around the ability for the LI-840A to accurately communicate data to a portable Android device. While we could write our own communication library using the Android USB protocol, we see it as a better solution to use an already written and tested Android serial connection API. This will be simpler to implement, less work overall, and most likely more stable than having to write our own library.

3.3.1 Alternatives

We have chosen three libraries to focus our research on. These three libraries are USBSerial, usb-serial-for-android, and Physicaloid. Both USBSerial and usb-serial-for-android are very promising, with a rich feature set and well documented code. If one library shows to be problematic then the other will be an excellent runner up. Physicaloid is mainly focused on use with Arduino devices, and the online documentation does not supply must information on how to use it with a more generic serial connections. For these reasons, we have decided to also rule it out.

3.3.2 Chosen Approach

We have decided to go with the usb-serial-for-android library. We chose this option for a few different reasons. First, the library is certified to work on devices running Android 3.1 or higher, and because all Android devices currently run Android 4.0.1 or higher, it should be fully functional on any device the end user has available. Second, the library is well documented. The Github page hosting the project is freely available, and even features code snippets showing how to implement the library in your application. This will make integrating it into our mobile app fast and easy. Third, the library is released under the LGPL license. This allows us the freedom to use it in our project without the obligation to also make our project open source.

3.3.3 Proving Feasibility

There are many different applications in the Android Software Center which communicate with devices over a serial USB connection. In order to ensure that communication with our particular device can be done, we downloaded one of these apps and attempted a connection. The mobile app ran perfectly, reporting data to the phone in an organized and pre formatted manner.

3.4 Data Transfer Out of Application

Once the data is collected and organized into files, it will need to be transported. As useful as it is to see the data on the device, the user will want to use the data in other places. Transferring the data will also serve as a way to back it up online since the user may not want everything saved locally on the Android device. The user will want to view the data in other places besides the device it was collected on.

3.4.1 Alternatives

Google Drive is an alternative that is very close to email. Google Drive has the advantage of being able to organize data files as well as being very well documented. However, integrating Google Drive, especially when one is required to link their individual accounts whenever they would want to send out data from the app, would be a very time consuming endeavor. Google Drive does not have a very user friendly interface if one's intention is to store a large amount of varying data. Since that will be a very large and vital aspect to this application, this is a drawback that inspires investment into other alternatives. Another alternative is to pull the data off the device over a usb connection. This option is very simple and would likely require minimal effort to implement, but would undoubtedly be time consuming for the user. Furthermore, this application is intended to be open-source and to be potentially used by anyone in the field who wishes to utilize its resources.This alternative would require the user to have the proper cable to connect their device, as well as a device that supports that kind of connection. Cloud or email based alternatives allow for the open-source status of the app to be much easier realized. The third option is email, where the application would prompt the user for an email address to send the data sets to.

3.4.2 Chosen Approach

We have chosen to email the files off of the device. One of the primary benefits to email, is that it is ubiquitous, and users would most likely not have to download any additional software, or learn any new applications to later access the data sets. The phone will have the data files stored locally until it has access to an email connection. Once connected the user can send an email to any desired address, and the selected files will be attached to it.

3.4.3 Proving Feasibility

Sending an email from an Android application has sufficient documentation and will be easy to implement. This includes validation and sending the email to any desired email address. A quick mockup was made in Android Studio to test that a mobile app is able to not only send an email, but also attach files to that email.

4. Technology Integration

All of our technologies will come together to produce a single, functional application. How they interact with each other can be seen as a four step process, starting with the collection of the data, and ending with the exporting of the data off the device.

First, our serial connection API will establish a connection with the gas analyzer, and report it to the mobile device. Second, the data will be organized into a format that can be understood by the graphing library, and any necessary statistical computations will be performed. Third, the user interface will display the data, using graphs, as well as text output to show the statistics. Fourth, if desired, the user can choose to email a folder of the data readings to a specific email address.

We expect most of our challenges to take place in step two, where statistical analysis will occur, as well as the preparation of the data for the graphing library, and step three, where we will have to develop a mobile interface that the users will find aesthetically pleasing, simple to operate, and easy to navigate. Determining the optimal layout for the interface will require many user polls, and constant revisions until a satisfactory look and feel is found.

The following diagram illustrates the basic architecture of our plan, with each box representing a specific part of the flow, and the lines representing what kind of data is communicated between the various parts.

5. Conclusion

In conclusion, the problem we set out to solve is making the collection of CO_2 data from a gas analyzer portable and easy to use. By passing the need for a 3rd device between the Android device and the LI-840A, the setup used by Professor Richardson and Professor Carbone will be far more mobile and accessible. The way we set out to do this is by establishing a USB-to-Serial connection between the LI-840A and an Android mobile device and then using various Android technologies to analyze the input, and display the data, while also using email to allow simple transferring of readings out of the application. The purpose of this document is to lay out all research behind all our our technical decisions and explain how we intend to use it. The chosen technologies are as follows:

Technology	**Proposed Solution**	**Confidence Level**
Operating System	Android	100%
Graph Library	GraphView	95%
Serial Connection	usb-serial-for-android	80%

Data Transfer	Email	90%

Overall, our top priority in selecting technologies was portability and dependability. Our proposed solutions seem to be the quickest, easiest, and most viable ways to solve the problem, while also providing the end user with simple and familiar means of interaction. The goal of this project is to allow users to understand more about our changing environment. It is important that we make an effective, and open source way to collect data crucial to furthering the understanding of our planet.