Geological Digitizer

Team CSM Jobe

Marcelo Gonzales, Jessy Liao, Alexis Ludeman, Courtney Richardson
Client: Zane Jobe

June 2019

Contents

Introduction 3

Requirements 4

Functional Requirements . . . 4

Non-functional Requirements . . . 4

Nice to haves . . . 4

System Architecture 5

Graphical User Interface . . . 5

Image Reader . . . 5

Plotter . . . 5

Technical Design 7

PyQt . . . 7

Pruning Contouring Functionality . . . 7

Paint Functionality . . . 7

Plotter . . . 8

Quality Assurance 9

Results 10

Lessons Learned 11

Appendix 12

Product Installation Instructions . . . 12

Development Environment Description . . . 12

Figures . . . 12

Introduction

Graphic logs are the most common way of characterizing sedimentary geologic rock formations in outcrop and core data. The term graphic log originates from a geologist measuring and drawing graphically, or ‘logging’, a core or outcropping stratigraphic section. Graphic logs generally have thickness/depth on the y-axis, and grain size on the x-axis. Many geologists spend weeks in the field carefully measuring and logging rock formations by hand. The rock thickness and grain-size data that may have taken days or weeks to collect is often never captured digitally So, while tens of thousands of meters of graphic logs measured at fine-scale exist to quantify various geological parameters, the data contained in the graphic logs is rarely digitized and available for use.

While software solutions exist to collect graphic log data in the field, there are thousands of hand-drawn logs that need to be digitized. Zane Jobe, a research associate professor and geology and geological engineering director at the Colorado School of Mines, has reached out to the computer science department for help creating a new and better way to digitize data. Thomas Martin, a PhD student in the Chevron Center of Research Excellence who works closely with Zane, is also an important member of the team as he intends to implement this software in his future research. This project aims to create an executable, where a geologist can digitize the rock layers from a hand-drawn graphic log, called Geologic Digitizer.

The basic idea of the Geologic Digitizer is that an image of a graphic log can be uploaded into a user friendly GUI, edited through the use of a “pen” and an “eraser”, and then processed so that a CSV file exists containing the x and y coordinates of the trace of the log. There are thousands of hand-drawn logs lying around from years of research. This project will help convert the legacy data to digital form so that it can be used for research. The hope is that the open-source release of this executable will spur the collection of quantified, structured, and comparable data that, with continued advances in the accessibility of machine-learning to geologists, will lead to new discoveries in sedimentary geology.

Requirements

Functional Requirements

When the executable is run, the GUI (graphical user interface) is created and prompts the user to select the image file of a geologic log to import. Once the image successfully loads, it is sent to the pruning class of the project, which uses a computer-vision python library named OpenCV. This edits the image to remove excess marks and takes enough coordinates that it accounts for the curvature in the layer lines, returning it to the GUI as a CSV file which is separated by layers. The plotter class imports the CSV and graphs the data split into layers onto the same plot in green. The plotter exports the plot as a .png file with a transparent background. The GUI's canvas object draws the plot image on top of the original image to show the user how they line up. If the user isn't satisfied, they have the ability to draw and erase different lines before sending it to the pruner class again.

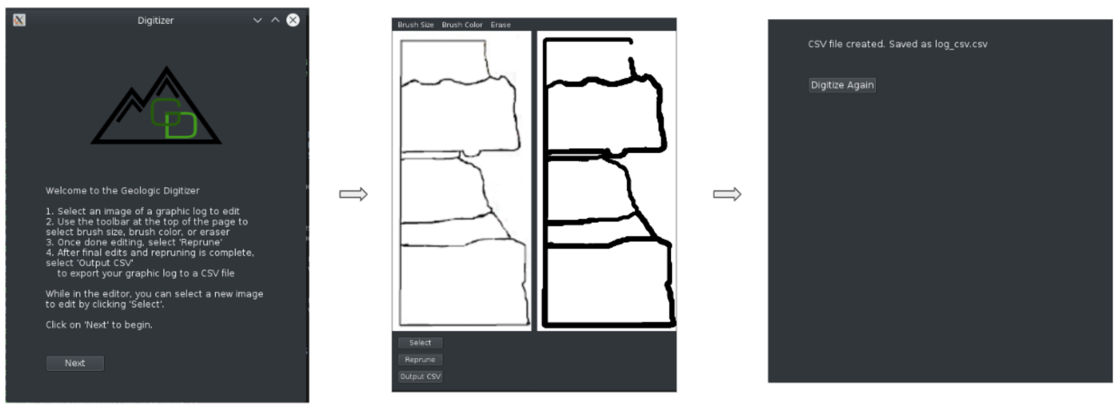
Non-functional Requirements

The program's GUI needed to be aesthetically pleasing and easy to navigate, as this is the only part of the code that the user will be interacting with. As the client base that will be using this software is geologists and not just computer scientists, it is important that the GUI be designed to be relevant for the needs of geologists. The client wants the program to be open-source so that it can be improved upon in the future, so the software is hosted in a public GitHub repository.

Nice to haves

Aside from having the base of the program and just collecting the CSV file with the correct coordinates, the client requested, if there was time leftover, to have a more complete GUI with additional options for the image. Giving the user the ability to indicate where the axis labels should go and refactoring the coordinates in the CSV accordingly and allowing the user to indicate layer lithology were the main two desires. If there was even more leftover time, the client wanted us to implement a way to indicate where geologic structures, such as waves and bedding level, by allowing the user to give the layer the appropriate sedimentary structure icon.

System Architecture



Welcome to the Geologic Digitizer

1. Select an image of a graphic log to edit
2. Use the toolbar at the top of the page to select brush size, brush color, or eraser
3. Once done editing, select 'Reprune'
4. After final edits and repruning is complete, select 'Output CSV' to export your graphic log to a CSV file

While in the editor, you can select a new image to edit by clicking 'Select'.

Click on 'Next' to begin.

CSV file created. Saved as log_csv.csv

Figure 1: System Architecture Flowchart

Graphical User Interface

The graphical user interface (GUI) is implemented to bridge the gap between the software and the user. After speaking with the client, he made it clear that there are both geologists who are unfamiliar with programming, and others that maintain a high-level of knowledge of computer use in general. The goal was to create a design that would contain all of the necessary functionality while remaining simple and easy to use.

The GUI is comprised of 3 different pages; 1) Welcome Page, 2) Log/Edit Page, and 3) CSV Output Page. The welcome page lists all the instructions to properly use the software. The log/edit page allows the user to upload a log and shows the original imported log on the left side of the screen, and the image to edit on the right. An example of this is shown below in figure 2. This allows the user to compare the pruned image with the original. Three buttons exist on the bottom of the screen labeled “Select”, “Reprune”, and “Output CSV”. “Select” allows the user to choose any image from their computer to prune and edit. “Reprune” allows the user to prune their newly edited image. “Output CSV” is clicked once the user is done with their editing, creating a CSV file to match their image.

The purpose of the GUI is to implement all of the created functionality into one easy to access spot, making it so the user is able to use a function without the need of understanding the software itself. However, the GUI is designed in such a way that if someone in the future wanted to add additional functionality to the software, the GUI could be expanded upon and the user could be presented with additional editing options.

Image Reader

The pruner class is called when the initial image is uploaded by the user. The image is uploaded and processed by the OpenCV library for python. The first step in its journey is through the pruning class, which removes any markings on the page that are below a certain threshold, making the page clearer. The software then outlines the leftover shapes, separating them into different sections. This data can be split up based on which shape it is, allowing separation between the different geologic layers. The program then takes each shape and records the x and y coordinates of the associated pixels in the shape's outline.

Plotter

The plotter class of the program takes in a CSV file that is created by the pruner class and passes its output to the GUI. Using the Pandas library, the CSV is read into a data frame inside the initialization of the class call. The data frame is ran through a loop to register the total number of layers. Once per layer, the plotting function is called and passed the data frame for that specific layer. The MatPlotLib library is used to create a single plot where

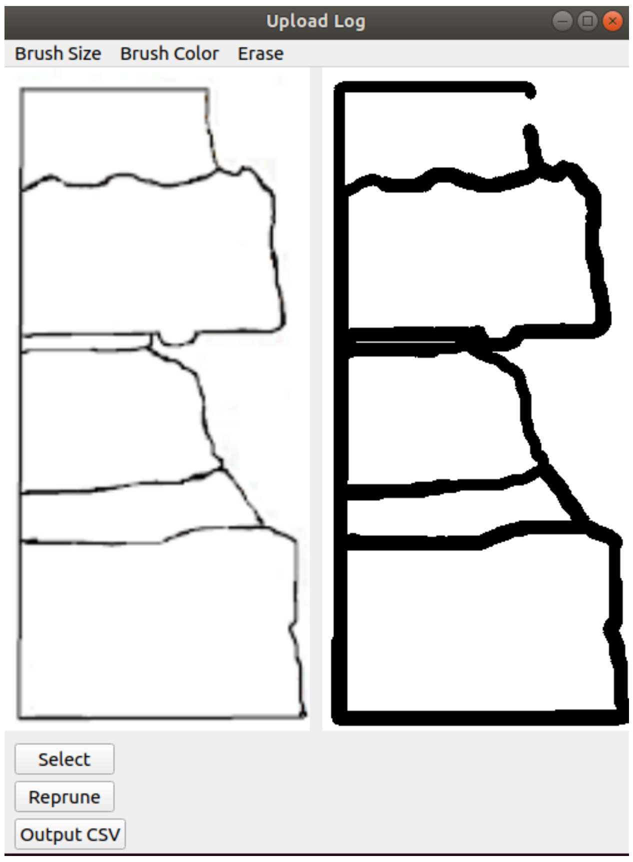


Figure 2: Graphical User Interface

the individual points are plotted. The points are connected by green lines, giving a complete outline of the layer. The other layers are then called one after the other and plotted on the same plot, also in green. When all the layers have been plotted successfully, the export image function is called. This function gives the plot image a transparent background and then exports the image into a png image. The exported image is used again in the GUI to allow the user to identify any mistakes. The GUI provides the user with tools to edit and erase different lines. The image will then be pushed into the pruning class once again, to repeat the process as many times as desired.

Technical Design

PyQt

PyQt was the main library used to create the GUI, specifically, the GUI was made with PyQt version 5 because the backend of the code was written in Python 3 and the two needed to interact with each other.

As mentioned previously, the GUI is made of three different pages. Since each page is different, they were organized into different classes. The welcome page class was relatively simple and only contained functions to display and format text. The log/edit page has two core functionalities: pruning/contouring, and painting/editing the image. This portion of the code is mostly dynamic and therefore the log/edit class consists of multiple methods that handle all the different callbacks.

Pruning Contouring Functionality

The pruning and contouring functionality in the code is mainly located in the backend. It was mostly written with the OpenCV library. OpenCV is a library in which its main purpose is to handle real-time computer vision. First the image is uploaded and scale to a certain size. It is then pruned, meaning that it detects all the shapes in the image, determines the area of each shape, and either ignores or saves them depending if they meet the threshold area. This is done because OpenCV will run too slow if it were to try to save all the information of every little shape (text are basically treated as tiny shape and take up a lot of memory). The code then sends the image to the contouring functionality which is built into OpenCV. The contour function will draw lines around each of the shapes. This is used later on in the code to determine the different layers in the graph.

Paint Functionality

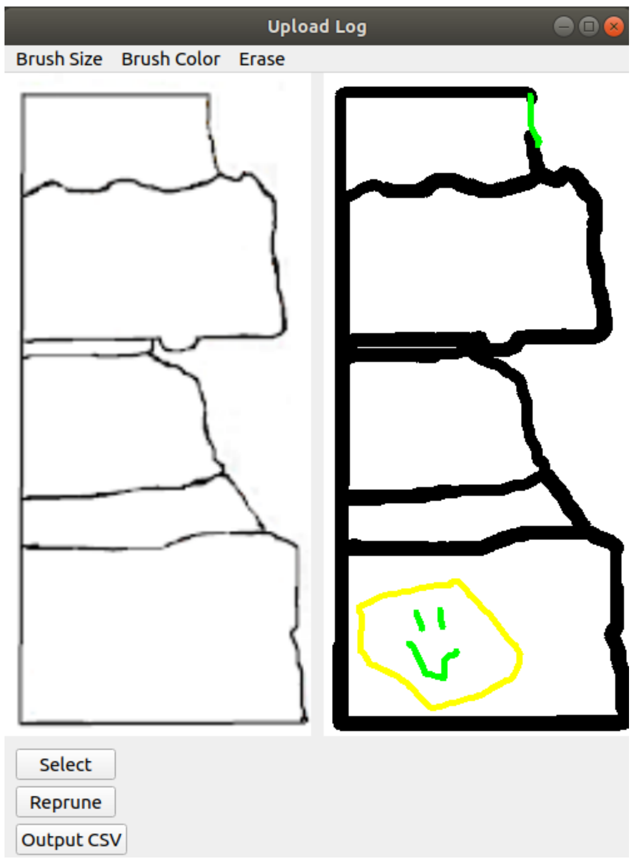


Figure 3: Brush Functionality

This feature gives the user three options: a pen, an eraser, and the ability to change the size of both. The pruned image located on the left is a canvas that has mouse listeners and callbacks built into the image object. This allows the user to edit the contoured image and ensure it is recognizing all the shapes within the graph log. Once the client finishes the adjustments, they can click on reprune to create another contour on the edited image.

Plotter

The plotter is a class that uses matplotlib and pandas to read in a CSV file which is organized with rows containing the layer number and the x and y coordinates of the pixels associated with the original image's layers. The class is initialized by importing the CSV using the pandas library and splitting it into a data container. Once the container is filled, the program identifies how many layers are in the container in order to split them up further.

A plotting function is called once per layer and makes a scatterplot of the x and y values. It then connects the points together in green, to make sure that the new image stands out from the original image that it overlays. After all of the layers are plotted on one graph, the class calls a function to export the newly-made graph to a .png file for the next step in its digitizing process. The flow of the class and its functions can be seen in Figure 4.

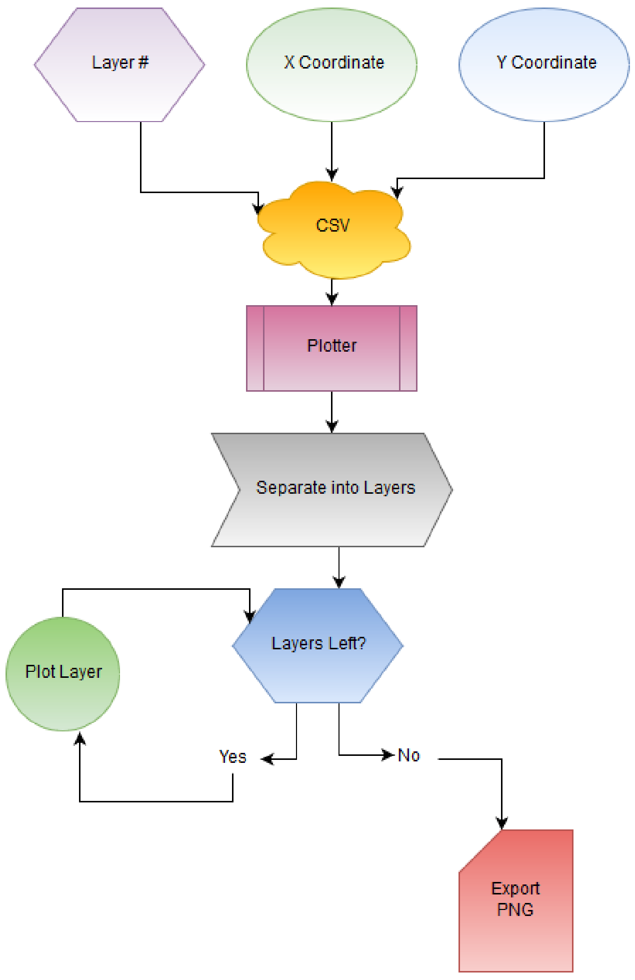


Figure 4: Plotter Flowchart

Quality Assurance

To establish an optimal and efficient product, several testing strategies were implemented to ensure the quality of the different functionalities of the digitizer. The software was created with an emphasis on test driven development. The following strategies were used to accomplish this development style:

Unit Testing: Pytest was the main framework used to test functions within the software. This was mainly done for the backend of the code which includes functions such as image handling, pruning, contouring, and CSV file generation.

User Interface Testing: User interface testing was accomplished by having our client, specifically Thomas, check the quality of the outputs that were being generated by the backend. The client also tested the GUI and how easily it is to interact with. This was important in developing the definition of done since initially there is a lot of information that can be extracted from the graphs. Through these user interface testing, the functions that extracted the most useful information, such as layer shapes, were prioritized.

Integrating Testing: After the front-end and back-end of the project were integrated, the team repeated multiple integrated tests. These tests were done by inputting a graphic log, which in turn outputted a CSV file in which the team could replot the points to ensure it resembles the original graph log. Plotting the point and comparing it to the original log will showed the digitizer is integrating the different components correctly.

Code Reviews: The team was divided into two subteams. One of these subteams were responsible for implementing the computer vision solution, and the other subteam was responsible for creating the GUI. As work continued on the project, the two subteams updated each other on how their code functioned, and reviewed the other subteams code.

User Acceptance Testing: The goal of the product is to meet the requirements of the client. The client was given all information and was able to interact with the code before anything was finalized.

Static Program Analysis: Before any code was written an UML diagram was created to structure how the architect will look like. This kept the code organized for the most part. Changes were obviously still made as the project continued, but the overall architect made these changes easy to implement.

Dynamic Program Analysis: For dynamic testing, the client was able to edit the images during run time within the GUI. This allowed for the outputs to undergo specific changes that will meet the clients needs before generating the final output.

Results

The goal of this project was to create an easy-to-use digitizer for the geology department at the Colorado School of Mines. The client's desire was that an image of a hand-drawn geographic log could be uploaded into the program, and the program would be able to detect each layer and provide a skeleton outline of the graph that the user could either continue to edit, or save and exit. Various tests were performed on each aspect of the project; details of these tests are listed above in the "Quality Assurance" section of the report. The program was then passed on to Thomas Martin for user testing. The feedback he provided gave guidelines for the final adjustments, and the software was edited to meet his expectations and needs. Due to the state of the current GUI, this program is best run on Linux operating systems. While the program has the ability to run on other operating systems maintaining full functionality, it has not yet been optimized for those systems and can present some layout and sizing issues. For example, on a Windows computer the buttons have a slight overlap and the image of the graphic log is presented smaller. Being able to run the program on any system without any issues is a feature that was restricted due to the time constraint. The goal would be to have the entirety of the software contained in a single click downloadable link that can be run on any computer with any operating system without any problems. Currently, the GUI contains three main tasks: Select, Reprune, and Output CSV. The descriptions of each of these functions are listed above in the "GUI" part of the "System Architecture" section of the report. There are also options that allow the user to change their brush size and brush color, or erase parts of the image. Future functionality of the program would include being able to click on each individual layer and add lithology and notes for extended documentation. Due to the open-source nature of the software, this implementation can either be added by the team at a later date, or built upon from other geologists or software engineers from all over the world.

Lessons Learned

This project contained a lot of trial and error as the team was trying to become familiar with Python and all of its options. There exists a vast amount of libraries that can be imported and used in Python. An abundant amount of time was spent experimenting with these different libraries and researching which would be the best fit for the project. It was decided that OpenCV would be used for the tracing of the image and exporting the CSV file, and Tkinter would be used to create the GUI. However, when trying to integrate the background part of the program with the foreground, it was discovered that Tkinter did not provide the optimal solution and the team encountered many problems while trying to make it work. After further research and tinkering with the Python user interface libraries, the final GUI was created using PyQt. This adjustment provided the team with extra knowledge of the programming language and an insight of how to adjust code in a timely manner to meet the clients expectations.

An extension of learning existed as the team strove to understand the specifics of a graphic log. It became clear that knowledge outside of the code was necessary to ensure that the software did exactly what it was supposed to do. The team learned where the data for a graphic log came from, how to collect it, and how it is used for various research explorations. It was important to have these geologic understandings before continuing with the code.

Appendix

Product Installation Instructions

1. The Github repo is hosted at https://github.com/magonzal/CSMJobe. In a Linux environment, open the terminal and change directories to where you want the packaged cloned using the command: git clone
2. The next step is to simply type in the same folder the repo was forked "pip install -r requirements.txt" without the quotation marks.
3. The final step is to simply type in the same folder python3 digitizer_v2.py to run the software.

Development Environment Description

A lot of the development for the project happened using several different IDE's. A lot of these IDE's were good for static testing since they highlighted any errors related to syntax or conflicts. PyCharm, for example, was designed specifically for the Python programming language. You can use the terminal to run several Python specific commands such as pip or even if the Anaconda package manager is being used. Jupyter Notebooks was also used in the sense of prototyping. Jupyter Notebook is a wrapper that shows the code in a browser such as Chrome or Firefox. Half of the team used the Windows environment and the other half used the Linux environment. Python is a great multi platform language as it can run in any operating system.

Figures

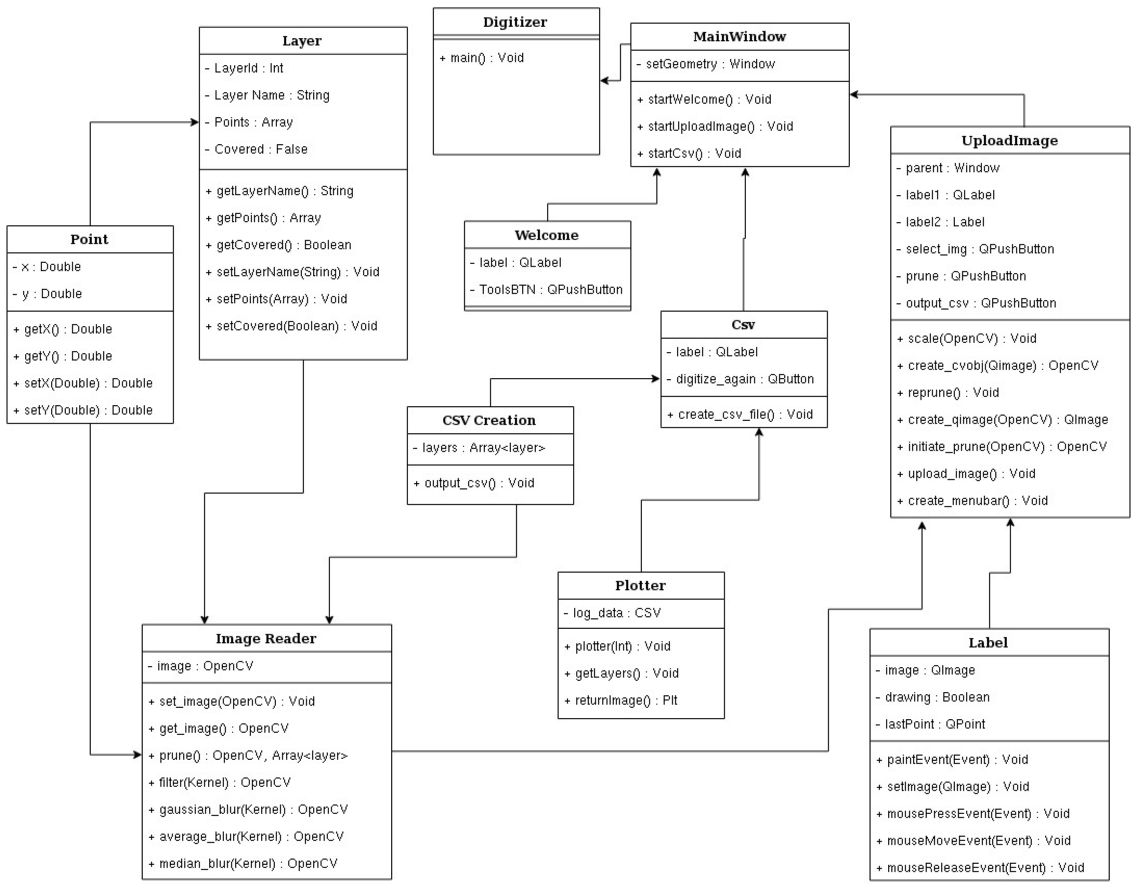
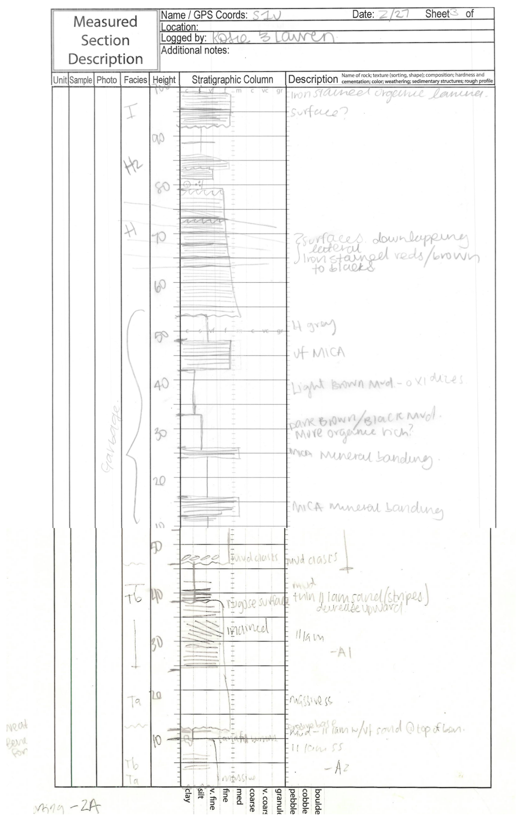


Figure 5: UML Diagram of software architecture



Measured Section Description

Name / GPS Coords: S1V Date: 2/27 Sheet 3 of

Location:

Logged by: Katie 3 Lauren

Additional notes:

Unit	Sample	Photo	Facies	Height	Stratigraphic Column	Description Name of rock; texture (sorting, shape); composition; hardness and cementation; color; weathering; sedimentary structures; rough profile
			I			Iron stained organic lamina. surface?
			H2			
			H			?surfaces downlapping lateral; Iron stained reds/brown to blacks
			Garbage			Lt grey
						vf MICA
						Light Brown Mud. - oxidizes.
						Dark Brown/Black Mud. More organic rich?
						MICA Mineral banding.
						MICA Mineral banding
						mud clasts
			Tb			mud; thin ‖ lam sand (stripes) decrease upward
						‖ lam; -A1
			Ta			massive ss
						erosive base; mud - ‖ lam w/ vf sand @ top of lam.
			Tb			‖ lam ss; -A2
			Ta			

Figure 6: Log Graph Example