Final Report

Team: sdmay20-43 Client: Dr. Simanta Mitra Advisor: Dr. Simanta Mitra

Analysis of GitLab Project Using Boa

Team Member – Roles:

Diego Realpe – Team Lead

Adrian Hamill – Test Engineer

Benjamin Carland – Test Engineer

Megan Miller – Test Engineer | Website Manager

Yi-Hsien Tan – Test Engineer | Website Manager

Team website: http://sdmay20-43.sd.ece.iastate.edu

Table of Contents

List of figures/tables/symbols/definitions

Definition

1	Boa – A DSL developed by ISU to data mine software repositories.
2	Data Pipeline – One of the sub-team that specializes in developing the data pipeline of the software.
3	Boa Research – One of the sub-team that specializes in understanding and creating Boa queries.
4	Boa expert – The experts who created/are professional in Boa.
5	DSL – Domain Specific Language
6	CLI – Command Line Interface
7	OS – Operating System
8	RID – Requirement ID
9	FR – Functional Requirement
10	NR – Non-functional Requirement

Figure

Tables

1 Introduction

1.1 Acknowledgement

Special thanks to the Boa team experts for providing their professional help throughout our development, as well as our advisor Simanta Mitra for giving us valuable feedbacks, and providing us resources we need.

1.2 Problem and Project Statement

This project aims to integrate an existing Domain Specific Language (DSL) Boa, developed by a team of experts in Iowa State University, with GitLab. This project can help our client in evaluating the repositories of his classes on GitLab efficiently.

The project consists of three major tasks:

- Integration of Boa with GitLab.
- Analysis of GitLab projects with Boa.
- Generation of analysis report using R with data from GitLab.

The output of our design:

- Command Line Interface (CLI) program that does analysis on GitLab repositories.
- Output analytic reports with R programming language.

1.3 Operational Environment

The program should run on both Windows and Linux OS command line. No support for mobile application.

1.4 Requirements

RID	Requirement Name	Description
FR1	Analyze Repositories	The program shall analyze repositories from GitLab.
FR2	Analytic Reports in R	The Program shall display analytic reports with R programming language.
FR3	Report Partitioning	The report shall contain two sections – commitment and code quality analysis.
FR4	Backend Automation	When the program runs, backend work shall be done automatically without manual connection.
NR1	CLI	The program shall be a CLI.
NR2	Ease of Learning	New users shall be able to learn the use all features of this program in less than 10 minutes.

NR3	Ease of Use	Analytic reports shall be displayed with just one click.
NR4	Security and Confidentiality	All information of repositories shall remain confidential.
NR5	Data Integrity	All software repositories' metrics shall remain original and unaltered after running the analysis.

Table 1: Requirements

1.5 Intended Users and Uses

This program is intended for:

- SE/ComS309 evaluators/graders.
- Possibly any users who are interested in data mining software repositories, depending if our client wants to publish the program.

1.6 Assumptions and Limitations

Assumption:

- There are no multiple user running this program at the same time.
- The primary users will be our client/his classes' graders.

Limitation:

- There are some backend functions that must be done manually occasionally.
- Only professionals who understand the program can maintain the program.

2 Implementation and Design Analysis

The foundation of our project was well implemented during the first half of the year. We separated our implementation into the front-end (handled by the Boa Research sub-team) and the back-end (handled by the Data Pipeline sub-team) during the first semester, and we combined both sub-teams' work into the final implementation we have now.

2.1 Proposed Design

The design of this project calls for a CLI front-end to ensure minimal computational memory consumption to run the software.

The software repository analyzing language of this software is built on top of an existing DSL named Boa, developed by a team of experts in Iowa State University. This involves members from the Boa Research sub-team to run many created queries to understand the code structure of the language in order to create our own Boa queries.

The program is created to be user friendly and as automatic as possible; the initial section of the program consists of configurations for the program. We collect the target repositories to be analyzed along with the queries to be run. From here the program takes off on its own, it requests the entire repository from GitLab and makes a replication of it in GitHub, this since the BOA analysis tool is only compatible with GitHub. This part juggles permissions and remote commands to git that create replica repositories from the name to the metadata and code, conserving it exactly from the original.

After this replication process that we have come to call "pull down, push up", we request the metadata from the GitHub API and store it locally so we can run queries against it to determine project quality. These are run in series and look and measure multiple aspects of the metadata obtained for each project.

Then the output is collected by a background daemon that will take this output and parse it in a clean format and alert the system when it is complete. When it is done, the final parsed output will be a clean compilation of the results for each analyzed repository from which we use R scripts to display graphically for our user.

2.2 Design Analysis

The team set up a basic implementation of the Boa framework for GitLab, which was built referencing the current implementation of Boa for GitHub. This has been a very effective strategy; the team can free up more time to focus on creating useful queries for better analysis, as well as creating our own “version” of Boa for GitLab.

At the beginning of the program we were bound by some constraints to make the Boa tool work alongside us. This tool was not compatible with any other sort of repository that was not GitHub and it was a monolithic piece of code that was complicated to predict on its behaviors, addedly that it had little documentation. Backend team's design settled on making a transition setup so this way the Boa tool that collected the repository information could have its desired input. The repositories found in GitLab needed to be completely downloaded and then replicated exactly as they are now in GitHub. For this we found several custom Bash tools that could allow us to do these actions automatically without needing to use the Web-Based UI that GitHub has to create remote repositories. The primary one we used was called Hub. This tool acts as a CLI front to do actions not available with the native git program like creating a GitHub remote repository for an account.

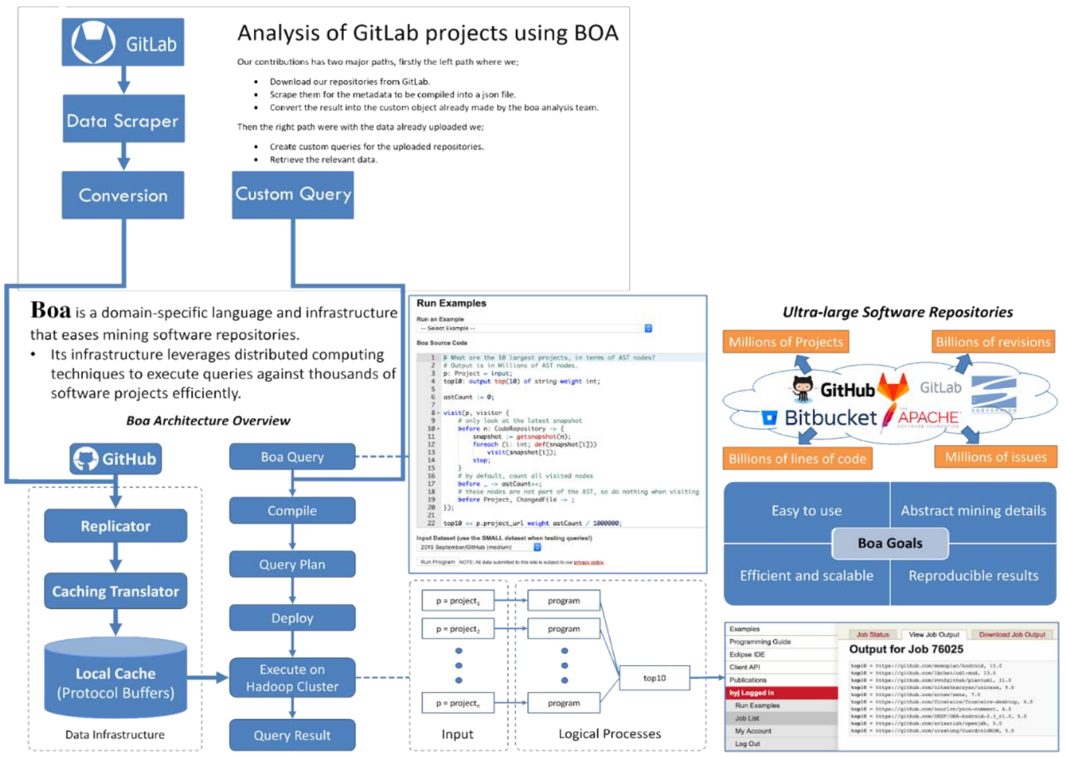
The program's final end goal is to create "SequenceFiles" which contain the metadata for the entire repository being analyzed. Think of this as the raw data at this point. From these we can feed them into the Boa interpreter which will run any given Boa query against the SequenceFile and return the result in a .txt file with the field or fields of the information requested by the user.

When the job is submitted by the user, which kickstarts the process of readying the repositories into being turned into SequenceFiles. It also passes information like input and output directories and credentials from where to take the repos from and to. Lastly it prepares the end part of the program to wait for these results and parse them. The program turns on a daemon using the package fswatch whose job is to look for newly created text files that appear in the output directory and when it sees these, it passes its name and other information like the query type that this output file contains so we know how to parse the information correctly. This file parser creates comma separated value type files from the results of the Boa query where each new row is a new parsed SequenceFile and hence a different project to be graphed by R.

The R programming language typically works using an IDE of some sort and manually creating scripts that graph the input data, however we needed to make our process automatic and couldn’t rely on our user having to manually look for the finished csv file and select the query to run, so instead the backend team found another custom package that can be used inside of our CLI which invokes R and gives it the csv file and an R script to use against it. Finally, the command is run and the R graph is displayed.

2.3 Design Plan

Figure 1 shows the visual plan of our design.



Analysis of GitLab projects using BOA

Our contributions has two major paths, firstly the left path where we;

- Download our repositories from GitLab.
- Scrape them for the metadata to be compiled into a json file.
- Convert the result into the custom object already made by the boa analysis team.

Then the right path were with the data already uploaded we;

- Create custom queries for the uploaded repositories.
- Retrieve the relevant data.

Boa is a domain-specific language and infrastructure that eases mining software repositories.

- Its infrastructure leverages distributed computing techniques to execute queries against thousands of software projects efficiently.

```
# What are the 10 largest projects, in terms of AST nodes?
# Output is in Millions of AST nodes.
p: Project = input;
top10: output top(10) of string weight int;

astCount := 0;

visit(p, visitor {
    # only look at the latest snapshot
    before n: CodeRepository -> {
        snapshot := getsnapshot(n);
        foreach (i: int; def(snapshot[i]))
            visit(snapshot[i]);
        stop;
    }
    # by default, count all visited nodes
    before _ -> astCount++;
    # these nodes are not part of the AST, so do nothing when visiting
    before Project, ChangedFile -> ;
});

top10 << p.project_url weight astCount / 1000000;
```

Input Dataset (use the SMALL dataset when testing queries!)
2015 September/GitHub (medium)

NOTE: All data submitted to this site is subject to our privacy policy.

Output for Job 76025

```
top10 = https://github.com/momoplan/Android, 15.0
top10 = https://github.com/lbchen/odl-mod, 13.0
top10 = https://github.com/svn2github/plantuml, 11.0
top10 = https://github.com/nitashnarayan/uxlcase, 9.0
top10 = https://github.com/arnaw/sasa, 7.0
top10 = https://github.com/frostwire/frostwire-desktop, 6.0
top10 = https://github.com/nourlcn/yarn-comment, 6.0
top10 = https://github.com/CESF/OBA-Android-2.3_r1.0, 5.0
top10 = https://github.com/sriastish/openjdk, 5.0
top10 = https://github.com/ursatong/GuardroidROM, 5.0
```

Figure 1: Visual Design Plan

2.4 Implementation Details

The entire process can be divided into three main pieces, along with setup and input. The entire process is shown visually in Figure 2.

Setup:

- Before using the software, the user has to designate the target GitLab repositories that are to be analyzed.

1. Repository Processing (Pull Down, Push Up)

- The credentials and addresses are used to fetch the repository images from GitLab and store it in local. Then, a script will replicate equivalent repositories on GitHub for an API to access it.

Input:

- When the user runs the software, they will be prompted to select a query to run on all the designated repositories.

2. Repository Analysis (Boa querying)

- The data is then scrapped by Boa using the GitHub API into the SequenceFiles with all the metadata that we need to analyze and run it against the selected Boa query. After the analysis is completed, the software will output the query result in text file.

3. Result Processing (R scripting)

- The software runs a daemon in the background which takes the outputted query result and automatically parse it in a .csv file that can be translated into graphical report with R scripts.

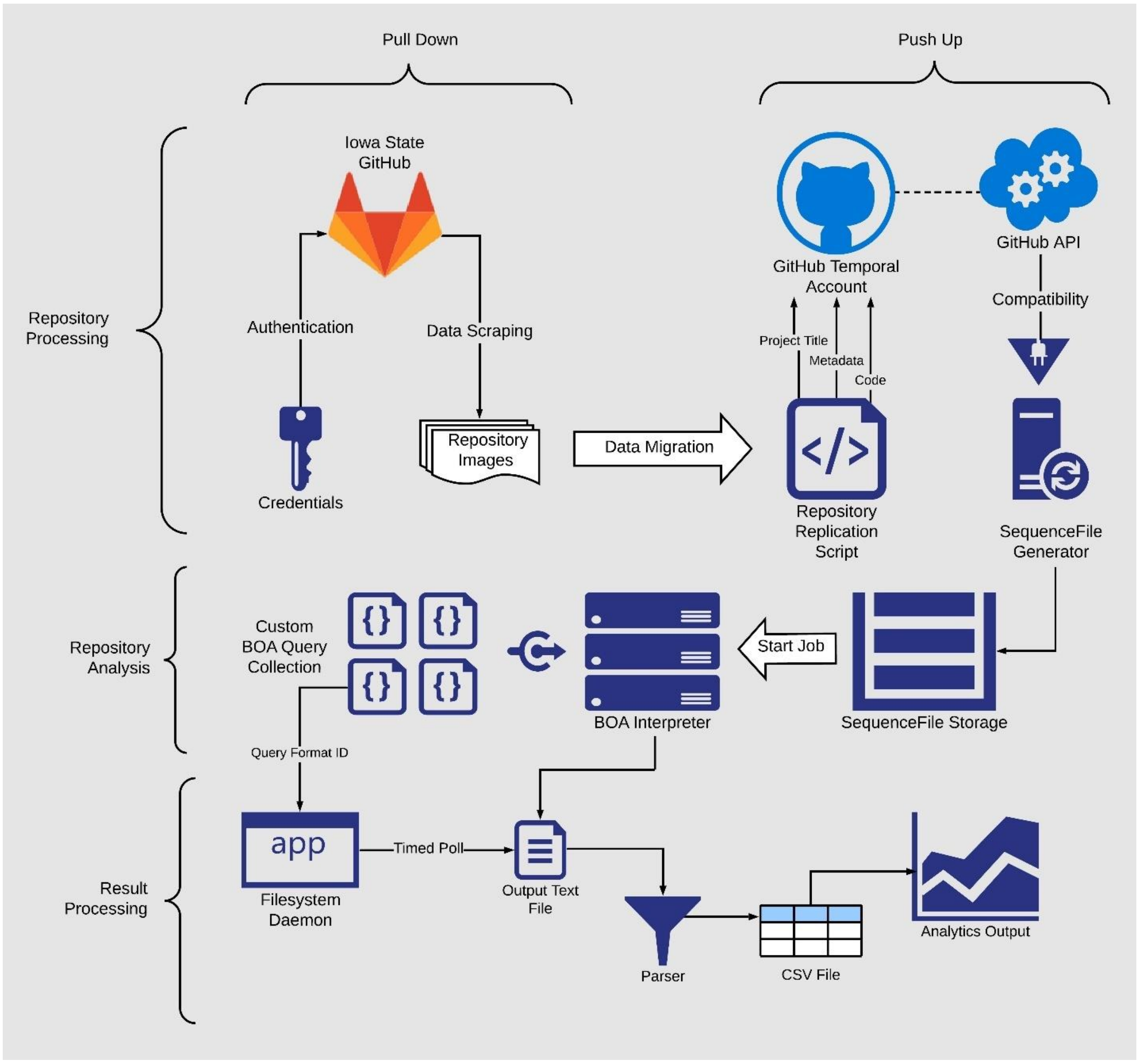


Figure 2: Process Diagram

3 Revised Project Design

The design of our software has changed since the Fall 2019 semester. While things are changing in our design, we had to come up with creative ways and solutions to accomplish the requirements we initially set out to fulfill. This section contains all the changes we have made since our last design document.

3.1 Data Pipeline Revision

During the year started working on the backend design we had many different ideas of how we could implement this, not too long after some of the constraints we had by our client, like using specifically the Boa tool, which in turn also subjected us to the constraints of the Boa tool itself.

At the very beginning of Fall 2019 we believed we could come up with a solution that would involve directly pulling the repository information from GitLab itself without needing our existing transition through GitHub. We believed we could parse the GitLab repositories manually and obtain the same end result. The SequenceFile raw data. This was not the case as both our attempts to adapt the SequenceFile Generator to use GitLab and our attempt to make the SequenceFiles themselves were unsuccessful. This mainly because of the monolithic nature of the Boa tools we were subjected to use and little documentation to go by off.

We also experienced problems with authentication towards both GitLab and GitHub. The repository analysis in Boa is still being worked on by the ISU graduate researchers and it is not yet a commercially ready product. Because of this, there seems to be no protocol native to Boa to authenticate with the remote repository that is to be turned into a SequenceFile. Since no authentication and identity verification can be made to GitHub then it means that Boa is not compatible to turn private repositories into SequenceFiles. This was a major hole in our design which counted that we could incorporate some type of solution which could allow us to keep any analyzed repo private for the entirety of the process. This was immediately notified by our client along with a compromising solution for it. We would maintain the repositories private in GitLab and handle them as much as we could locally, but when they got uploaded GitHub it only would be a temporary replica which would be alive publicly for a very short amount of time (minutes) until the repo could be uploaded and converted into a SequenceFile by the GitHub API + the Boa SequenceFile creator. Afterwards the content and the project itself would be deleted leaving no trace.

Currently this is the current version of the backend which our client Dr Mitra has approved of.

4 Testing

4.1 Interface Specification

The software interfaces we will need to test our software are minimal. A series of a few GitLab repositories with dummy code of good and bad quality and various amounts of commits that we wish to set up so we can test the efficiency of the suite of queries we have created and determine if they were able to provide useful insights to each of them.

4.2 Hardware and Software

- CS309 Old Repositories: These will be used to perform load testing on our system, filled with code from past 309 semesters
- Dummy Repositories: These will have mocked code with edge cases and various types of errors.
- Mac Mini: This computer will be our test server where the first prototype will be tested.

- Boa Website: Functional website from the Boa labs where isolated queries will be compiled and tested.

4.3 Functional Testing

- Unit: We are going to make each of the queries we develop to go through unit testing by running them in isolation on the Boa website which has an online compiler of the Boa language. This to test the correct functionality of each before integrating them in the query suite
- Integration: We are going to use some dummy repositories developed to observe if the transition between the backend pipeline to the analysis piece of the system transitions the information accurately
- System: Our final tests for the system will be performed close to the end of Spring semester, using the repositories from CS309 of the current semester. If successful we should see analysis and insights that correspond to those student's final grades.

4.4 Non-functional Testing

- Load: Using access to the large amounts of code found in the archives of CS309 we will perform a load test to determine that the final product is fully able to handle the volume of an average 309 class for our client.
- Compatibility: Unfortunately, the overarching project that we are using can only handle MacOS. However, we will test compatibilities on both the Client's machine and our Mac Mini (which have different versions of MacOS) to ensure that the transition between one and the other has no bugs or issues.
- Usability: At the end we will allow Dr. Mitra to have a period of test with the final product and address any usability features he would like changed or added.

4.5 Process

The process for testing each of our methods will follow a semi manual testing. Since the technology we are using is very niche, there are no real ways of mocking it. Our overall plan is to perform blind experiments with our data to test the reproducibility of our insights. By feeding the pipeline unknown quality of code and obtaining our results, we can check the veracity of it by actually reading from the project afterwards and if the assessments of the system seem accurate

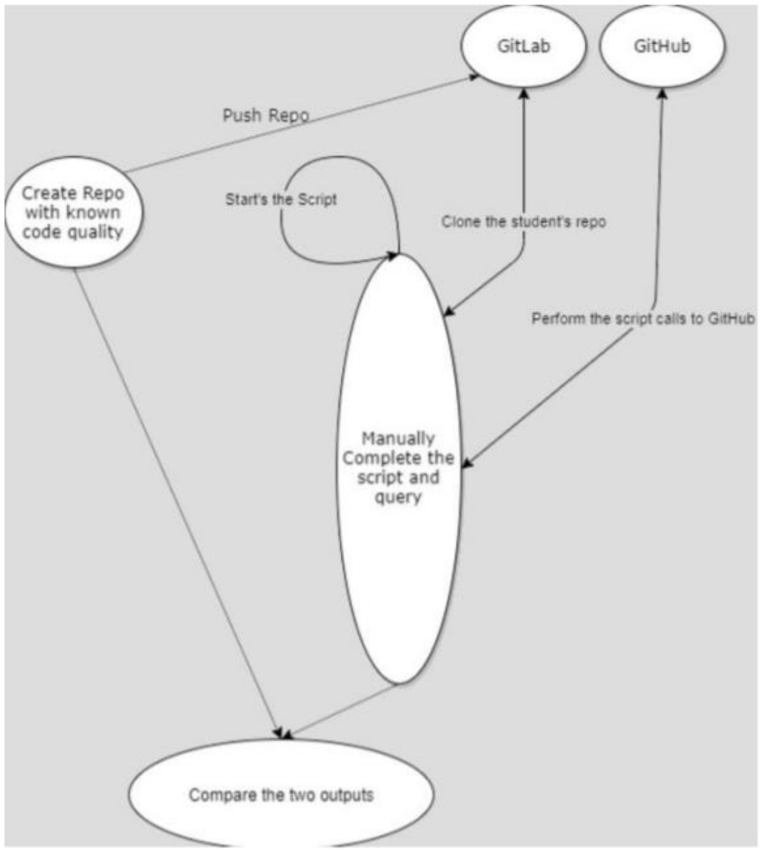

GitLab
GitHub
Push Repo
Create Repo with known code quality
Start's the Script
Clone the student's repo
Perform the script calls to GitHub
Manually Complete the script and query
Compare the two outputs

Figure 3: System Testing Flow

4.6 Results

4.6.1 Backend Results

The pipeline backend has yielded some results already regarding the cloning, scraping and automation of the data. Our first solution for this problem looked to pull directly from the GitLab repo using a government program designed for this. However, after conclusions that the program was not suitable and failed to provide the particular pieces of metadata, we required we moved onto a different one. This solution allowed us to make clones of the repositories in GitHub with all data intact (which is one of our requirements) and from there we can use the Boa Labs code which does work on GitHub and allow the pipeline to query a GitHub API to construct the metadata JSON files necessary for Boa to execute against. A diagram of the breakdown of the system can be seen to the right where the process is detailed.

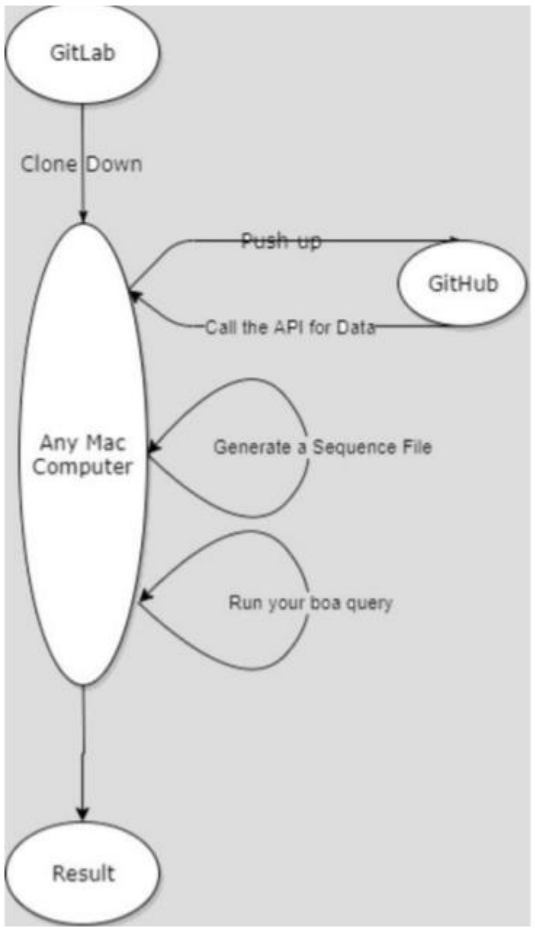


Figure 4: Data Pipeline Process

This system is also tested against the edge cases of a repository. Based on the trials and errors we underwent to bring a prototype of the scaping script and process we have gained tons of knowledge on how Boa operates and also has given us a time to partner with the experts that use Boa, allowing us to have a quick place where to go for extra resources or when we get stuck in a problem.

4.6.2 Boa Analysis

The Boa Research sub-team utilizes many existing software like FindBugs and SpotBugs to compare results against the results produced by our Boa queries. The Boa libraries also offer to make a tool that talks about code quality on each of the repositories. As we go into the development and writing of these queries, the directions we take to assess the quality of a project spreads out and multiple aspects of the metrics we are given will be compared. As mentioned before, Boa is a complex language to write high level analysis tools for and we expect a lot of trial and error in this section as lots of ideas will be tried and discarded while we push the limits of what Boa can and cannot do. Final results on this area will be a suite of queries that can be toggled on or off that analyze different parameters of the code, we also check for things like documentation and private variables to ensure readability, cohesion and coupling issues.

5 Appendices

5.1 Appendix I – Operation Manual

The project has been developed to require little manual control from the user. For running this project please start at the home directory and run ./setup.bash. Here the setup script will be started and ask the user for configuration specifications. It will ask about the system this will be run in. Either MacOS or Linux. Afterwards, the instructions will print and guide the user smoothly through the installation process. A couple of utilities will sometimes require additional info, like as an example, the utility "Hub", which will manage the push up, pull down solution will need an access token to be provided to it. Afterwards the user needs to specify the repositories that the program will be running against. You will be prompted to provide a Git username and a URL, then give an access token for the utility to to have permissions to automatically manipulate repositories on your behalf. Finally it needs a local directory where it will run the project data scraping in and store files in. Provide the program with this target and give it the path of the repositories you want analyzed too. The program will then proceed to use the Sed utility to insert your given configuration options into the data scraping scripts that are in the generics folder and run them. At the end of this process the SequenceFiles for the given repositories should be created and ready to be run queries against them. The program now is ready for you to choose Boa analysis queries from our given suite and repository SequenceFiles for you to run against.

5.2 Appendix II – Initial Version of Design

The initial version of our software design is mainly different in the backend part. Initially our design is to build an entire Boa framework on GitLab since the original Boa only works on a handful of software development version control site like GitHub and SourceForge. It is proven that our initial idea is very difficult to be achieved as GitLab does not provide the same API we need to run Boa as GitHub does.

In our current design, instead of building a Boa framework on GitLab, we decided to clone all the repositories we want to analyze on GitLab onto GitHub. Not only this will save us tons of development time by using existing resources, we can devote our time on perfecting the transition of “pull down, push up”, as well as our script to run the process.

In terms of the frontend, there isn’t any changes since how the team planned out in the beginning. To understand more about the frontend, visit section 2.