Team Term Project Final Report

Table of Contents

Chapter 1………… page 3

Chapter 2………… page 5

Chapter 3………… page 9

Chapter 4………… page 10

Chapter 5………… page 11

Chapter 6………… page 12

Chapter 1

Wiki was formally hosted at <https://plague.fandom.com/>

Candidate 1: Epidemiological Modeler (STEM) Language: Java

* Simulates the spread of various infectious diseases throughout different populations
* Huge library, will take some time to learn
* Well documented with tutorials on how to get started

Candidate 2: Drone 4 Dengue Languages: Javascript, HTML

* Is image processing software that detects mosquito breeding sites.
* The images are grabbed by a drone in the area.
* Not well documented
* Looks incomplete.

Candidate 3: Nightscout Languages: C, Java, HTML

* CGM (Continuous Glucose Monitoring) software.
* Compatible on many Operating Systems such as Windows, IOS, Android, Linux
* Setup guarantees errors with too many dependencies to properly demonstrate from a fresh setup.
* Formal setup guides, however error bound.
* Many applications developed for the Web Browser, Android, IOS, and even several smartwatches.

Team Exercise: <https://github.com/csci-362-01-2019/Team3/blob/master/myList.sh>

This script will list all of the files in the current directory as a bulleted list in HTML. Run with the command "./myList.sh >> fileName.html" to create the HTML file. For example, if the script is in the Documents folder along with two folders named "one" and "two", the output will be:

* myList.sh
* one
* two

Project: STEM (Spatio-Temporal Epidemiological Modeler) Following instructions from <http://wiki.eclipse.org/STEM_Eclipse_Setup> to build the STEM project, kept getting errors on the final build. Some of the plugins in the tutorial are outdated, and presumably covered by newer plugins. However, upon attempting to launch STEM, a Framework error stopped the launch. The developer build is still a work in progress. The standalone STEM release should work for a number of pre-programmed scenarios, however the program hangs up when starting a scenario. Currently troubleshooting.

Looking Forward:

Ultimately, once one machine can properly install the packages and sort through the dependencies of STEM that contain bugs, we can begin developing scripts to start testing our project.

Chapter 2

**Introduction**

Since our previous Deliverable, our group decided to switch the HFOSS project we focused on. We decided to switch to OpenAQ because STEM included many problems for our group including:

* Issues finding all packages of the source code.
* Building what little of the source code we actually could obtain.
* Not being able to successfully build the project on multiple machines running Ubuntu Linux.

With a lack of running software to develop or run tests on, our group could not continue making progress within the overall scheme of the project. Therefore, our group has decided to work with OpenAQ as our new #1 candidate HFOSS project. OpenAQ includes:

* All source code needed to properly build from scratch.
* Guides to build the project from the source code.
* Project being written in JavaScript.
* Successfully built and inspected on a few machines already (more on this later).

**Accomplishments**

During project development for Deliverable 2, our group has accomplished writing a test plan, which includes the hierarchy of how we are to proceed with testing different aspects of OpenAQ throughout the rest of the semester. Our team has also written how we would test certain operations within OpenAQ, 5 out of 25 is what we wrote so far.

**Learning Outcomes**

Our team has learned how to develop a test plan for a detailed method of testing different parts of the software from bottom to top. The work done helped develop a further understanding of how to write test cases for a complex software which includes multiple components and interfaces.

**Problems Experienced**

Even though the guides for building OpenAQ appear to be well written, the steps to successfully building OpenAQ sometimes appear to be out of order. At particular steps, a software package would not install because prerequisite software was instructed to be installed afterwards. Installing all software components when needed resolved many of these issues.

**Going forward**

The next goal is to keep uncovering more information of how the software functions in order to develop further relevant test cases. As our team learns more software engineering principles to apply, we can further strategize our resources efficiently towards completing future Deliverables for OpenAQ.

Below is the Test Plan for the OpenAQ project.

The Openaq project is an open-source project that gets air quality data from different cities and countries and makes them available to the public. The pollutants Openaq tracks are PM10, PM2.5, sulfur dioxide (SO2), carbon monoxide (CO), nitrogen dioxide (NO2), ozone (O3), and black carbon (BC). It gathers this data from official-level outdoor air quality source. The readings must meet the criteria of Openaq-fetch in order to be added to the database.

This is the test plan for openaq-fetch, the main data ingest pipeline for the [OpenAQ](https://openaq.org/) project.

**The Testing Process**

During the Testing of OpenAQ-fetch, an order of phases will occur. The first will be Unit-Testing where individual operations are tested with a set of test cases to guarantee the operations meet required specifications. The second phase will be Component-Testing where multiple operations are tested together using an interface (forming a component) to guarantee that a particular set of criteria is met. The last phase will be the overall System-Testing in which a select set of components are integrated together and then tested for expected conditions to be met from the software.

**Requirements Traceability**

The software is required to function as expected which entails:

* The software performs under expected conditions
* The software receives all necessary parameters within every operation during any set of tests
* The interactions between the components during system testing must match with what the oracle predicts.
* A report is made after a set of automated tests have completed.

**Tested Items**

An operation within OpenAQ known as, MakeSourcefromData(), which adds air quality data to the database. The operation uses parameter unit values such as coordinates, dates, value of ppv (particles per volume), and units of measurement.

**Testing Schedule**

The schedule will be as follows:

* The first set of Unit-Tests will occur during the development/understanding of code for a couple of weeks from October 1st to October 29th.
* The next set of testing will include the development and execution of an automated testing framework for 25 test cases from October 29th to November 12th.
* The final set of testing involves fault testing where components within the framework will be tested with malicious input for expected results from November 12th to November 19th.

**Test Recording Procedures**

The software will be running tests from a script file that records the results of the tests into text file with a particular format depending on what part of the system is being tested. The results are then compared to the oracle's text file for correctness & expected output. Lastly, a report is made onto an HTML file of the results from testing our project.

**Hardware and Software Requirements**

Software & Hardware Requirements include:

\* Windows, Mac, Linux system

\* Approx 200MB of free disk space

\* At least 1.8GB free memory

\* 2 core CPU is recommended

\* node.js >= 8.6

\* Either of:

\* docker or

\* postgresql >= 10 with postgis >= 2 extensions.

**Constraints**

Constraints include:

* **Lack of Time** due to the testing portion of the project being limited to 2 months.
* **Lack of Staff** due to amount of students working on the project being limited to 3 members with diverse tasks.
* **Lack of Knowledge** due to every student working on the project still learning methods for testing.

**System Tests**

Test cases will include tests ran through scripts upon the operations of OpenAQ-fetch for pass/fail conditions. Tests will be ran on the inputs for several particular operations for stability, and expected output. Tests will be ran on the outputs of several operations for correctness. Tests will be ran on order of execution for the system components as well as the overall system for expected goal completion.

Each test case is created in javascript and run from the command line using: node index.js --dryrun 'test'

--dryrun, -d Run the fetch process but do not attempt to save to the database and instead print to console, useful for testing.

Test case 1: Input: parameter: 'pm25', unit: 'ppq', //Error: Should be ppm value: 10, location: 'test1', coordinates: { latitude: -20, longitude: 34 }, country: 'US', city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'instance.unit is not one of enum values: µg/m³,ppm'

Test case 2: Input: parameter: '01', //Error: Not a valid parameter unit: 'ppm', value: 10, location: 'test2', coordinates: { latitude: -20, longitude: 34 }, country: 'US', city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'instance parameter is not one of enum values: pm25,pm10,no2,so2,o3,co,bc'

Test case 3: Input: parameter: 'no2', unit: 'ppm', value: Q, //Error: Value must be a number location: 'test3', coordinates: { latitude: -20, longitude: 34 }, country: 'US', city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'instance value is not of a type(s) number'

Test case 4: Input: parameter: 'pm25', //Ideal case unit: 'ppm', value: 10, location: 'test4', coordinates: { latitude: -20, longitude: 34 }, country: 'US', city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'Data source accepted.'

Test case 5: Input: parameter: 'pm25', unit: 'ppm', value: 10, location: 'test5', coordinates: { latitude: -20, longitude: 34 }, country: , //Error: Country field required city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'instance requires property "country"'

Chapter 3

**Introduction**

Since our previous deliverable, our group has worked on implementing our automated testing framework, while developing 15 of 25 test cases.

The goals for this deliverable were to:

* Rework the test plan written during Deliverable 2 (Chapter 2).
* Develop an architectural diagram or description of our testing framework.
* Document a full how-to on deploying our test environment and project.
* Developing at least 5 of 25 eventual test cases within OpenAQ-Fetch.

Plenty of time has been spent writing code to fulfill the core requirements of our automated testing framework.

**Accomplishments**

During project development for Deliverable 3, our group has accomplished rewriting the test plan previously mentioned, which includes the hierarchy of how we are to proceed with testing different aspects of OpenAQ throughout the rest of the semester. 15 test cases were written for OpenAQ-Fetch to execute. The automated testing framework was written in the BASH programming language. Documentation was provided for deploying and executing a full clone of the project, as well as our automated testing code. Lastly, an html file was written to report the outcomes of the testing framework based on the test cases used within OpenAQ-Fetch.

**Learning Outcomes**

Our team has learned how to develop a proper test. Our team has learned how to write code in BASH, as well as HTML. The work done helped develop a further understanding of how to write proper test cases for a complex software which includes multiple components and interfaces. Knowledge of a “Layered Architecture” was discovered for designing the description of our automated testing framework.

**Problems Experienced**

During development of the BASH code for OpenAQ-Fetch, we were having trouble discovering the proper syntax and method to parse and execute a string of arguments from a separate text file which contained our test cases.

**Going forward**

The next goal per specification of our test plan, is to one hundred percent finish the design and implementation of our testing framework. As well as finish developing a full set of 25 test cases for OpenAQ-Fetch.

version of our Layered Architecture can be found below

