Matplotlib Testing Framework

By: Jan-Michael Carrington, Vincent Nguyen, Phillip Wilson

Table of Contents

Choosing and Building a Project

The Early Birds team at first chose Sugar Lab was the initial open source project to test, but due to its complexity and the team's lack of experience in working with virtual machines and Ubuntu, the team switched to Amara. Amara is an open source program used to insert subtitles to videos and depended on the open platform Docker. Installing Docker and Docker Compose took everyone in the team a fair amount of time due to Docker itself needing other components to correctly function. The team underestimated the space needed to install the whole system, so they had to delete their current virtual machine and create a new one. After installing the necessary components, each group member managed to compile and build the program and run the provided tests. The tests provided data, but the team was unsure on how to view it. The progress made looked promising.

Difficulties with Dependencies

The progress quickly died down. A test plan was made for Amara, but as the team attempted to test the methods, they found the already included test files had multiple dependencies with different python files which also had different dependencies. Once getting all the necessary files, they found out Amara was based on the Django web framework and quickly sailed ship elsewhere. The team was eyeing pixi.js to work on, but it was made in javascript and difficult to create a suite for it. The team decided to switch to Matplotlib. It is a python 2D plotting library able to generate plots, histograms, power error charts, etc. Getting the open source software to build was relatively easy compared to the previous two projects mentioned in the previous chapter. Due to the switch, the team was behind one deliverable and had to catch up on their

work. Their test plan was remade to fit the new open source software. The methods will be manually tested to verify they compile correctly and are able to be executed from the command line. Once clear, a testing framework will be built to test each individual methods automatically. The projected schedule was:

> October 25, 2016: Revised test plan to fit the new project and design a testing framework
>
> November 15, 2016: Complete the design and implementation of the testing framework from deliverable #3. The twenty-five (25) test cases must be ready.
>
> November 29, 2016: Design and inject five (5) or more faults into the code being tested to cause the tests to fail. Test out the framework and analyze results.
>
> December 1, 2016: Complete final report and presentation.

The constraints of the team were not minor. All three members of the team were full time students and part time workers. One of which worked more than 20 hours per week. Physically meeting outside of class was a challenge and in fact never happened. They used online communication to keep each other up to date about their work on the project. Each test cases will output to a file their description of the process, the method being tested, the input, and the expected output. The first five test cases went smoothly. The methods colors.to_rgba(), colors.to_hex(), colors.is_color_like(), dates.num2date(), and dates.date2num() were successfully tested with little to no issues. The team looked ready to finish it all in one stride.

Testing Framework and Misunderstanding

The testing framework was built and centered around the python script runAllTests.py. The framework makes use of python's testing suite. The script runs through each test case file in the test cases folder. The template for each test case file includes a test case number, requirement, component tested, method tested, input, expected output, and the test case executable file. The script grabs the method, input, and expected output and runs a test on the method using the executable file. The test will either return passed or failed. This data is stored in an array and kept for later. Once all the test cases have been tested, the script exports the data to an html file and a text file. The html file contains a simple table with the test case number, requirement, component, method, input, expected output, and whether the test passed or failed. The text file contains extra information on methods that failed, such as where in the executable file the test failed at.

In order to run the testing framework on one's own computer, simply clone the repository at https://github.com/CSCI-362-01-2016/The-Early-Birds and extract it to the desired folder. The framework does require the following dependencies: Python 2.7, 3.4, or 3.5, numpy 1.6 or later, setuptools, dateutil 1.1 or later, pyparsing, libpng 1.2 or later, pytz, freetype 2.3 or later, and cycler 0.9 or later. To install matplotlib download a zip of it at https://github.com/matplotlib/matplotlib. Extract it, then inside of it run "sudo python setup.py build." This command will alert you of any missing dependencies, if none, everything went fine. Then run "sudo python setup.py install." The last this required to run the testing framework is to

go inside the frameworks scripts folder and run the command “chmod +x runAllTests.py.” Now from the TestAutomation folder, run “./scripts/runAllTests.py” to run the testing framework.

The team misunderstood the required deliverable October 25, 2016. The testing framework was supposed to test at least five of the eventual 25 test cases. The team believed they needed all 25 test cases. They rushed to get the work done, and as a result, the work was not of the highest quality they could offer. A third of the 25 test cases delivered that day were setters and getters. This was not all negative however, the work for the next deliverable was cut down.

Refining the Testing Framework

Thanks to the team’s misunderstanding of the previous deliverable, the amount of work to be done for November 15, 2016 was significantly lower than it should have been. Most of the test cases were working correctly. Some formatting was changed as per the suggestions of Dr. Bowring. Numerous test cases that were testing setters and getters were changed. One of the group member was unable to properly work on the test cases for that deliverable, but the group came through with quality work nonetheless. The group was on schedule to deliver the fault injection on November 29, 2016.

Fault Injection

The matplotlib source code was found by searching the whole system for “matplotlib” and see which one was inside the python2.7 directory. It was found at this path:

/usr/local/lib/python2.7/dist-packages/matplotlib-2.0.0b4_2382.g042ebc7-py2.7-linux-x86_64.egg/matplotlib

The files were read only so to open them the team used sudo gedit [filename].py to edit them.

The test cases 03, 18, 19, and 22 were projected to fail for the first fault injection since they all used the same method.

The colors.py file was altered on Line 246:

```
return "#" + "".join(format(int(np.round(val * 255)), "02x")
    for val in c)
```

was changed to:

```
return "#" + "".join(format(int(np.round(val - 255)), "02x")
        for val in c)
```

All the test cases related to the dates class were tested by changing a constant in the file.

The dates.py file was changed on line 186:

HOURS_PER_DAY = 24.

was changed to

HOURS_PER_DAY = 22.

Test cases 09, 10, 11, 12, 13, 14, 15, 17, 20 all failed as expected. However, test cases 07 and 09 still passed their test cases.

Test cases 07 and 09 are not using the constants after inspecting the code.

Reflections

This was the team's first time working on open source software, and they have finally realized the importance of the material taught in their software engineering class. The amount of software

needed to be able to properly work any of the four open source software they picked was ridiculous to them which encouraged them to build code with low coupling. They have always been taught to write documentation to their code, but it was not until now they realized the importance of proper documentation. The documentation provided on the Matplotlib website helped them tremendously in building their testing framework for the open source software.