Deliverable 3: Testing Framework

Red Team

September 29, 2015

Architectural Framework

The testing framework is intended as an all-in-one solution that requires only addition of a single, structured, text-based test file to run the system. Using Python's scripting abilities, the system both sets-up tests and runs them within the main "runAllTests" script. To describe how the system works, we have the following ordered list of the functions of the runAllTests script:

1. The script sets up its local working directory, noting where tests are coming from (/testCases) and where to place its report (/reports).
2. The script opens the test case directory, lists all of the files, and then puts them in alphanumerical order.
3. The script begins to generate an HTML file to use as the report for the tests.
4. The script then runs through all the files in the testCases directory in which it:
    a. Lists the identity, requirements, component, method, input, and expected output for a given test case (which is printed to the HTML report file)
    b. Parses together the component, method and input into a string
    c. Executes the string using an exec method within Python.
    d. The system then prints the outcome of the statement to the HTML report file
    e. If the system encounters an exception, this is then printed to the HTML file and the script moves on.
5. At the end of running through the tests, the script then lists how many tests it ran, how many passed, and how many failed, and prints this information to the HTML document.

How-To Documentation

The process for writing a test for the Mercurial test set is as follows:

1. Create a text file in the /testCases directory. The current naming convention is "testCase##.txt," and the system runs through the files in alphanumeric order, so if you don't follow this convention, be aware of where your test should run based on that order.
2. Within the test case text file, list information in the following order (information with a * indicates it will be parsed, so write it with the intention of that specific code being executed):
    a. Test Identification
    b. Requirement of the method being tested (what's supposed to happen or a quick description)

    c. *Component being tested (what you would import from mercurial, i.e. “from mercurial import **fmtremaining**)
    d. *Method being tested (only the method name, no parenthesis or any other symbology)
    e. *Input being tested
    f. *Expected output (exactly as you would expect the system to return it)
3. Once the test case is generated, you can execute “runAllTests.py” within the /scripts directory

Report

Working on creating an automated testing framework has been a very engaging process. The first push towards getting this deliverable done was getting a simple, purely Python test to execute from outside the source directory. After encountering mind-boggling errors where the Python script could not locate its import dependencies, we suspected the issue may lay in using the wrong version of Python (we were using version 3). Unfortunately, when working with SCM software that is made to house code, finding an adequate Google search to figure out which version of Python Mercurial is based off is a fruitless quest. We made the switch to 2.7, and things instantly began to work, and we called it a day after having a successful test built.

The second, and heftier push, exposed some of the common problems faced with working on a larger software project. After completing most of the requirements for the project, we went back over the specifications, and realized we needed to reorganize the files into their proper directories among other things. The next problem came at us as we realized that the automated structure required for the project was somewhat ambiguous. We rebuilt our scripting framework to accommodate the desired file types for the test cases and executables, but came away unclear of how to incorporate the oracle into our development. At this point, the test case text files contain both the inputs and oracles, which will be fixed as we receive clarification on how we are intended to implement the oracle structure. Finally, today we integrated command line arguments as the test inputs and oracle components for our unit tests. We realized with advanced data types we ran into many problems, so we have another problem we will look to fix in the very short term so we can have the test framework running in complete automated form.

Test Cases

Here are the first 5 test cases as they are written in their text files:

testCase01.txt:

1

Break a given integer into the largest increments of time it can be broken into
progress
fmtremaining
100
1m40s

testCase02.txt:

2
Given the range of two points, calculate the point of overlap between them
simplemerge
intersect
(0,100), (50,150)
(50,100)

testCase03.txt:

3
Counts the number of processors available to the operating system
worker
countcpus

1

testCase04.txt:

4
Return the length of the given string
templatefilters
count
"abcde"
5

testCase05.txt

5
Return elements in array concatenated into one element in a larger array
namespaces
tolist

"1" "2" "3" "4"
['1234']