Drew Rodman
Chad Hobbs
Jason Daniel
Jake Wisse
Brett Ostwalt

Schfifty Five Testing Framework Report

Team 5 Deliverable Number 3

Framework Architectural Overview

The testing framework as currently implemented adheres to the directory and file structure as defined by the project specifications in deliverable number two and the initial guidelines. Test cases are defined in text files located in a dedicated directory named ‘testCases” and must conform to a specified template in order to be validated by the testing script, which is written in Python. Each template defines a number of fields in a key-value format, including the source file being tested, a test case executable, a test identifier, the component being tested, and so on. All of these are parsed by the script and stored in a dictionary for later inclusion in the test report and test execution. All Java source files included in the “src” directory and all test cases in the “testCaseExecutables” directory are recursively pre-compiled by the script and any failures are reported and then skipped. For each test case, the template is parsed and the test executable filename is read from the dictionary and used to locate the Java file to run. Each test case returns an output which is read by the script and then written to a results file, which is in turn compared to an associated oracle using the “diff” shell command. A record is kept of the output of diff, indicating whether each of these tests pass, fail, or encountered an error. This result is appended to the template data gathered from each of the test cases and combined into a test report document in “reports,” which is viewable in any web browser. These reports are time stamped and the user may accumulate as many as he or she

wishes before removing them. The contents of the temporary directory are cleared after every run, so the testing framework may be run an arbitrary number of times without modifying any files or the directory structure and achieve the same result.

Guidelines for Running the Testing Framework

The testing framework is intended to be completely automated, not requiring any user interpretation of test output to determine whether a given test passed or failed. That said, the testing script is named “runAllTests.py” and is located in the “scripts” subdirectory; simply executing this script will trigger the compilation, execution, and analysis of all tests. It is required that you open a terminal window and *cd* (change directory) into the project directory before running commands. If you have never run the script on your system before and you attempt to do so and get a “Permission Denied” or similar error, all you have to do is run *chmod +x ./scripts/runAllTests.py* in order to make the script executable. After this step, simply type *./scripts/runAllTests.py* and press enter to run the script. You can follow what the program does at a high level in Terminal and your browser will open automatically to view the test report.

Development During Design and Build

Our primary development model started as a prototyping model and switched into an incremental development process once a basic framework was established. The initial prototype demonstrated most of the functionality that was desired in the testing framework and helped identify required methods and parameters of the scripts and drivers. Once the requirements were mostly developed, the prototype was cast aside and the first increment of the final product was created. Individual components of the script were completed individually, reviewed for functionality, and then committed to the project. This would in turn lead to another cycle where either a different component was developed or an existing component was refined. This

incremental method of development lead to a steady march towards project completion. Further development of the project will follow the same model until the final product is delivered.

Obstacles Encountered

A difficulty we predicted revolved around the fact that one of our team members was across the country for a week. However he was able to communicate via email and still work on the project, so this was easily resolved. However, an unforeseen obstacle stemmed from the nature of the OpenRemote project, which is a highly integrated and coupled system that requires substantial configuration and communication between classes. That said we are evaluating potential tests that require as little scaffolding as possible and do not expect to encounter any further obstructions to meeting the required deliverables.

Timeline

Our script and framework are fully functional, leaving us with approximately thirteen days to implement the remaining test cases. The remaining test implementations will be derived from the project requirements and developed by all team members using the completed test cases as a template. It may be necessary to set light inter-team deliverables or deadlines in order to keep everyone on track and communicating any issues or successes.

Current Tests

In order to have something to test, we have primarily focused on the utility classes for the project that provide convenience methods such as binary to decimal conversions and an object pools. As previously mentioned, now that the framework itself is fully functional we will be improving upon existing and deriving more test cases from both the project requirements and non-trivial methods in the OpenRemote Controller source files.