Tushar Narayan
tnarayan@wpi.edu

CS562 Advanced Topics in Software Engineering

Testing for Developers

Assignment 2- Build a Regression Test Harness

This is the submission for the assignment “Building a Regression Test Harness” for course CS 562, taken A term 2014 at Worcester Polytechnic Institute.

Capabilities

The following capabilities are implemented:

Basic

Initialize the test harness.

This was implemented in the file “initialize-harness”. I remove any existing results, tests, data, history, and temporary files from the harness, and start a new history log.

Reinitialize/restart the test harness.

This was implemented using the provided “restart-harness” file. I remove existing results, and move all *passed*, *failed*, and *unclassified* tests to the *new* state. Depending on the arguments, I also move *inactive* tests to the *new* state, and possibly remove all the historical logs. I also start a new history log.

Run all tests.

This was implemented in the file “run-all”. I run all *new* tests, *passed* tests, and *failed* tests. The file uses separate files “run-new”, “run-passed”, and “run-failed” for those particular functions.

Run one or more specific tests.

This was implemented in the file “run”. It can run one or more specific tests when the paths to the tests are provided. It has a verbose option as well, that prints out the details of each test run (either way, the results do get saved to the appropriate result directory).

Classify tests, that is specify whether one or more tests should be classified as passed, failed, etc.

This was implemented in the “pass”, “fail”, and “deactivate” files. The “pass” and “fail” scripts move all *unclassified* tests to the *passed* and *failed* directories respectively. The “deactivate” script moves all *failed* tests to the *inactive* directory (and also deletes their results).

Intermediary

Report on the last execution of the test harness (e.g., how many tests passed, how many failed, how many are unclassified, etc.).

This was implemented in the “generate-report” file. I find the number of *passed*, *failed*, *new*, *inactive*, and *unclassified* tests, and then print them out along with percentages.

Tushar Narayan
tnarayan@wpi.edu

Allow the classification of all unclassified tests with one command. For example, if you have 50 tests and 8 have changed and are unclassified, you should be able to run a command such as "pass all" that would classify all of the unclassified tests as having passed.

This was implemented using the files "run-new", "run-passed", and "run-failed"; which run all *new*, *passed*, and *failed* tests respectively.

Advanced

Provide a report of all test runs that have been run since the initialization, or reinitialization, of the harness.

This was implemented in the "get-run-report" script. I log all the actions for the respective scripts that run tests, or change their states, to the history. The history gets refreshed on both initialization and reinitialization; thus, I simply print the current history for this.

Provide a command that clearly shows the difference between a previous result and the current result.

This was implemented in the "result-difference" script. The user specifies a test name, and if there is both a new result and an old result for the test (either *.passed* or *.failed*), we print the results out in succession to allow easy comparison; else we display appropriate error messaging.

Provide a command that will "print out" a specific test for viewing.

This was also implemented in the "run" file. Using the '-v' parameter, the user can specify that the test should be run in verbose mode. The harness will then print out the details of the test run on the console while also saving the results to the appropriate result directory.

Installing and Running

To install, simply unzip the *harness.zip* file. The resulting *harness* folder contains the directory structure and files required to run the harness.

On the Vagrant box, please make sure to move the harness **out** of the shared folder, otherwise there are "text file busy" errors when initializing/reinitializing the harness that are related to moving the *current* history into the *historyx* file. It seems to be a known issue – I found http://stackoverflow.com/questions/19124367/rails-app-on-percise32-vagrant-box-assets-get-text-file-busy-error-errnoe that related to this.

The scripts are in a **harness** folder within the top level **harness** folder. The two tests provided are in the **new** test folder.

To run all the scripts, please use **ruby scriptname arguments**. I tried to convert the scripts into standalone executables using the hashbang at the top and the appropriate file permissions, but I repeatedly got a *no such file* error when trying to use the executable version. I suspect it's because I created the files on Windows and then copied over the Linux, but I didn't spend too much time investigating because the harness works this way as well.

For example:

```
vagrant@precise64:/vagrant/harness/harness$ ruby restart-harness
```