Josh Gilley, Alex Wizes, Clae Wyckoff

CSCI 362 - Software Engineering

Professor Bowring

11/14/2020

Chapter 4

For deliverable 4, we had to implement our full automated testing suite with 25 test cases, 5 drivers and a single script. This should have been an easy task after having developed our initial driver and script, but we had to do quite a bit of refactoring before we even started expanding our initial testing suite. This was due in part to us wanting to optimize our script and drivers to make everything as efficient as possible, but also to bring it in line with the specifications provided to us by professor Bowring. In the sections below, we will first discuss our drivers and the changes we made to them before moving on to discuss the new and improved script. We will next discuss some of the other, smaller hurdles we faced before giving a brief reflection on our experiences developing this deliverable.

The Driver

Just after our submission of deliverable 3, we realized that we had made an error. Though our program was functional, the roles of our script and driver were essentially mixed up. Initially, our driver would read our test case info, instantiate an object for our test class, pass the testing inputs to the test method within the object, compare the actual output with the expected output, and return to the script a very long string with all of the info for our test report. If that sentence seems long to you, then you can understand why we needed to refactor. In short, we realized that the vast majority of those actions were supposed to be handled by the script, not the driver. This actually simplifies things and renders some of those actions unnecessary.

So the refactoring process began and after many late nights, pots of coffee, cry breaks, and motivational talks in the mirror, we ironed out a new driver that we could easily modify for each test method. This new driver (seen in **Figure 4.1**) is much simpler as it doesn't include our driver object and it leaves most of the work we discussed above to the script, as it should. Now our driver simply requires the proper files, instantiates the object containing the test method, passes the test method the test input(s), and returns the output. Aside from some simple error handling, that's the extent of our refactored driver's duties. The script calls each driver 5 times, once per test case. While every input will be different in each call to a driver, the only differences between each driver are the files they require, the object they instantiate, and the method they test. As a result, it was very easy to write 4 more drivers for our 4 other test methods (another driver can be seen in **Figure 4.3**). We were really making things harder on ourselves than we had to

Figure 4.1 - parse_charset Driver

```php
<?php
//DRIVER FOR PARSECHARSET METHOD
define('CLI_SCRIPT', true);
require_once("../project/moodle1/config.php");
require("../project/moodle1/lib/classes/text.php");

if (!$argv[1]) {
        throw new Exception('Inputs are invalid...');
}
try {
        $text = new core_text;
        $output = $text->parse_charset($argv[1]);
        print_r($output);
} catch (Exception $e) {
        print_r('Caught exception: '.$e->getMessage()."\n");
}
?>
```

Figure 4.2 - parse_charset Method Signature in Moodle

```php
public static function parse_charset($charset)
```

Figure 4.3 - break_up_long_words Driver

```php
<?php
//error_reporting(0);
//TEST STRING TO LOWER & REGULAR EXPRESSIONS

define('CLI_SCRIPT', true);
require_once("../project/moodle1/config.php");
require("../project/moodle1/user/lib.php");

if (!$argv[1]) {
        throw new Exception('Inputs are invalid...');
}
try {
        $splitStrings = explode("*", $argv[1]);
        $output = break_up_long_words($splitStrings[0], intval($splitStrings[1]), $splitStrings[2]);
        print_r($output);
} catch (Exception $e) {
        print_r('Caught exception: '.$e->getMessage()."\n");
}

?>
```

Figure 4.4 - break_up_long_words Method Signature in Moodle

```php
function break_up_long_words($string, $maxsize=20, $cutchar=' ')
```

The Script

After realizing our error and rebuilding our driver, we knew we had to put all of the functionality we stripped from out of the driver back into our script to make a functional testing suite. Our old script would call our driver as a subprocess, format our html file for our test report, then split a very long string of test cases and attributes before writing them to the html file and displaying it in a browser. If this sounds a little jimmy-rigged, we share that sentiment as well. Fear not however, our new driver is much improved.

Our new script is itself actually very simple (seen in **Figure 4.5**). It's nothing more than some json reading/variable creation, some method calls, and some html opening, closing, and displaying. In fact, most of the work is done by methods imported from a separate python file. This second file we wrote (part of which is visible in **Figure 4.6**) contains methods to get the testing output from the drivers, add them as an attribute to each test case for the final report, compare the actual and expected outputs, add the test cases to the html file, and 2 sort methods (1 for drivers, 1 for test cases). Because of this design decision, our script is very clean and our program itself has a great deal of modularity to it. Aside from calling these methods, our script simply opens an html file, calls our method to add the test cases to it, then closes and displays the html file in a browser (as can be seen in **Figure 4.7**). Again, we were really making it much harder on ourselves than it had to be.

Figure 4.5 - The Script

```
5   import importlib.util
6   import sys
7   import webbrowser
8   import time
9   import glob
10  import functools
11  from runTestsFunctionsAndVars import *
12
13  # read in the test cases
14  arrayOfCases = (glob.glob("../testCases/*.json"));
15
16  # globals
17  cases = []
18  outputs = []
19  driversUsed = []
20  sortedCases = []
21
22  # run subprocess to drivers to get outputs
23  getOutputsFromDrivers(arrayOfCases, cases, driversUsed, outputs)
24
25  #add outputs as attribute to cases
26  addOutputsAsAttr(cases, outputs)
27
28  #sort cases according to drivers
29  sortDrivers(driversUsed, cases, sortedCases)
30
31  # write html opening to html report
32  f = open("../temp/testReport.html", "w+")
33  f.write(htmlOpening)
34
35  # add cases in html format to the html report
36  addCaseToHTML(sortedCases, f)
37
38  # add html closing to html report
39  f.write(htmlClosing)
40  f.close()
41
42  #open temp file in browser
43  webbrowser.open("../temp/testReport.html")
```

Figure 4.6 - A Piece of the Imported Class We Wrote

```
7   def addOutputsAsAttr(cases, outputs):
8           throughOutputs = 0;
9           for case in cases:
10                  case['actualOutput'] = outputs[throughOutputs]
11                  throughOutputs += 1
12
13  def compareExp(actualOutput, expectedOutput):
14          if (actualOutput == expectedOutput):
15                  return "Pass"
16          else:
17                  return "Fail"
18
19  def getOutputsFromDrivers(arrayOfCases, cases, driversUsed, outputs):
20          for case in arrayOfCases:
21                  f = open(case);
22                  data = json.load(f)
23                  cases.append(data)
24                  testingInput = ""
25                  driverName = ""
26                  for key, val in data.items():
27                          if (key == "driver"):
28                                  driverName = str(val)
29                                  if driverName not in driversUsed:
30                                          driversUsed.append(driverName)
31                          if (key == "testingInputs"):
32                                  testingInput = str(val)
33                  driverName = '../testCasesExecutables/' + driverName
34                  result = subprocess.run(
35                  ['php', driverName, testingInput],
36                  stdout=subprocess.PIPE,
37                  check=True)
38                  outputs.append((str(result.stdout).replace("b'", ""))[:-1])
```

Figure 4.7 - Final Report Displayed in a Browser

The Hurdles

We faced several hurdles during this phase of the project. Obviously the most glaring was the need to completely refactor our code after realizing we were doing some things incorrectly. However, we would have needed to refactor a bit anyway to enable our script to handle multiple drivers. Before the refactor, our program had some info hard coded that while functional, would've had to have been duplicated and adjusted should the user want to add additional test cases. Now, our program has nothing about any test case hard coded anywhere, and our script can handle as many drivers/test cases as the user could write without any adjustment. This was just one of the hurdles we faced while tackling this deliverable though.

One of our other issues was simply finding methods we could use to test. Moodle is a somewhat large project with some very complex functions, and it took us several hours of looking to find enough methods to use. We had a lot of success finding methods in a php file called weblib.php, but we thought it to be a cheap move to use that class more than necessary. Then again, many methods in weblib.php had to do with converting html elements into entities which made for tricky testing, so we ultimately decided to move on. This was likely for the best as the methods we found work very well for our purposes.

The final issue we had was trying to figure out how to call our script from the correct place and having it run. This turned out to be a simple fix. Now we have a bash script in the scripts folder that runs all of the scripts in that folder, including out runAllTests script. Having

remedied that final error, we feel that we have adequately completed deliverable 4 and we are confident in our ability to accomplish everything for deliverable 5.

Reflections/conclusion

This deliverable was particularly difficult for the team. We thought that most of the work would be done after deliverable 3, that we would just need to make some minor adjustments to the script and the driver and then we could easily write the other 4 drivers and 20 test cases. That clearly was not the case as we had to not only refactor our code for 2 reasons, but also that we had to spend hours on a scavenger hunt for a method we might be able to test. It took a lot of work and patience to get everything done, yet we believe we managed to do so with some elegance and some class. We are happy with our project considering that at this point we believe we've done everything we need to and it's working as specified. Nonetheless, we've become weary in our experiences with open source projects, and we are fully prepared for whatever curveballs deliverable 5 has in store for us.