

Josh Gilley, Alex Wizes, Clae Wyckoff

CSCI 362 - Software Engineering

Professor Bowring

College of Charleston

9/22/2020

Experiences Working with Opensource Software

The Another-One-Bytes-the-Dust Team

Table of Contents

- **Intro.....3**
- **Chapter 1 - Project Choices.....5**
 - **Sigmah**
 - **Glucosio**
 - **Moodle**
- **Chapter 2 - Test Plan.....16**
- **Chapter 3 - Designing & Building Our Testing Framework.....18**
 - **Building the Driver**
 - **Building the Script**
 - **Hurdles Faced**
 - **Reflection**
- **Chapter 4 - Full Test Suite Implementation.....28**
 - **The Refactored Drivers**
 - **The Refactored Script**
 - **The New Hurdles**
 - **Reflection**
- **Chapter 5 - Error Insertion and Analysis.....36**
 - **normalize_version**
 - **round**

- **trusttext_strip**
- **break_up_long_words**
- **parsecharset**
- **Reflection**
- **Chapter 6 - Overall Experiences and Lessons Learned.....45**
 - **Reflections**
 - **Conclusion**
- **Chapter 7 - Evaluations.....47**
 - **Team Self Evaluation**
 - **Project Assignments and Suggestions**
- **Presentation Material.....48**

Intro

The following document is a report on the experiences of Josh Gilley, Alex Wizes, and Clae Wyckoff working as a team to complete a project given to us for our software engineering class. For this assignment, we had to build an automated testing framework for an open source project according to specifications provided by our professor. The chapters were written as we reached various checkpoints within the assignment and between major breakthroughs and setbacks. As such, some things mentioned in one chapter may not necessarily be consistent with something mentioned in a later chapter. We've taken efforts to prevent or explain that sort of thing when it

occurs, but we're only human and may not have been as thorough as we would have liked to have been. Should such a tragedy arise, the later chapter should be deferred to as the way we ultimately did things. This unfortunate possibility arose because there was so much to do for this project, but without further ado, let us begin the process by talking about some of the projects we considered.

Chapter 1 - Project Choices

In the beginning there was Josh, Alex, and Clae on a quest to build a magnificent project from a deprecated open source project. While we first thought this would be an easy task due to the sheer number of project choices before the mighty Another-One-Bytes-the-Dust team, we were quickly proven wrong when our first, second and third project choices failed to bear fruit. As a result, we switched to Moodle, and immediately saw the benefits. But first, let's talk about our failures.

Section 1 - Sigmah

Our original choice was Sigmah, a free software developed to help international aid organizations manage the information from their projects including reports, indicators, schedules, documents, etc. We chose this project because it was written in java and seemed to have some pretty good documentation. For a while we had success in the first step of building the project, which involved building the postgres server that handled our database. But after this we were given the task to copy and adjust the pom.xml and settings.xml files of the Maven build. Having tried to run it with the command "mvn install -Psigmah-dev" we received a fatal error message that included the following message: "org.apache.maven.plugins:maven-compiler-plugin:3.1:compile." **(Figure 1.1)** We tried searching stack overflow for a solution to this issue, but there wasn't very good documentation on what we were experiencing. We did find one article suggesting to run a Maven clean install which did not work. We later added the <build> tag below to the settings.xml file which also did not bear fruit **(Figure 1.2)**.

Figure 1.1 - Fatal error with Sigmah and Maven

```

BUILD FAILURE
-----
Total time: 2.591 s
Finished at: 2017-03-01T13:09:47+05:30
Final Memory: 21M/347M
-----
Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.1
/Users/vshukla/git/Prism/src/main/java/main/MainUITabbed.java:[26,22] pac
/Users/vshukla/git/Prism/src/main/java/main/MainUITabbed.java:[93,41] can
symbol: class Application
/Users/vshukla/git/Prism/src/main/java/main/MainUITabbed.java:[93,67] can
symbol: variable Application
-> [Help 1]

To see the full stack trace of the errors, re-run Maven with the -e switch
Re-run Maven using the -X switch to enable full debug logging.

For more information about the errors and possible solutions, please read
[Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureExce

```

Figure 1.2 - Second error involving Sigmah and a settings.xml file

```

</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.5.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>

```

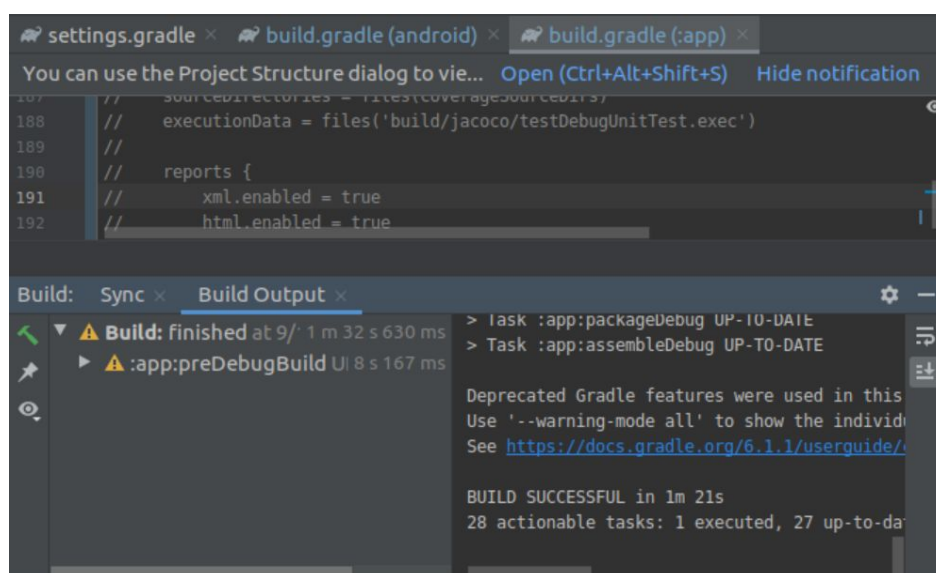
We came to a point where there were still other issues we were anticipating, so we felt it was better to cut our losses and try another project, namely Glucosio.

Section 2 - Glucosio

After having exhausted ourselves attempting to remedy the issues with Sigmah, we reluctantly turned our attention to Glucosio. We chose Glucosio for a number of reasons including that it could be worked on either as an android project or an ios project. We thought that working on Glucosio would be a good opportunity to learn some objective-c, an apple based language becoming more frequently used. Unfortunately, like Sigmah, we also ran into a number of issues with Glucosio.

Glucosio failed in the mobile app build process due to one of the Jocosio dependencies no longer being supported, and when searching for a solution we came up without a fix. So in a desperate attempt to keep the Glucosio train rolling, we commented out the Jocosio lines of code, and proceeded to run the build. We encouragingly had some success where the build would complete successfully (**Figure 1.3**).

Figure 1.3 - Successful build of Glucosio



So now we figured we were in the clear. All we had to do was get it to show up in the Android Studio emulator of our choice. However we ran into an issue where we had not enabled VT-x/AMD-V nested virtualization on our virtual machines. So we ran the command: "VBoxManage modifyvm Nubuntu2 --nested-hw-virt on" which turned on the nested virtualization feature. At this point the build was building successfully, and everything was as it needed to be. However, when running the project the emulator remained a black screen with no indicator that anything was wrong or even happening. This was a difficult problem to diagnose, as we may have been a few simple steps away but had no leads to investigate for a solution. Indeed, our efforts were futile and no solution was found. Consequently, decided to move on to the next option Miradi, which quickly turned into a decision to move on to Moodle.

Section 3 - Moodle

Despite our reluctance to abandon yet another project, we forced ourselves to reevaluate our options. We took a look at our third choice, Miradi, but quickly came to the conclusion that it was not in our best interest to pursue. We felt that Miradi lacked sufficient documentation, and instead opted for Moodle precisely because it contained a significant amount of documentation.

Moodle attracted the interest of our group both because of its level of documentation, and because we felt as a group that we could work effectively with this project. The prospect of working with php, a language nobody on our team has used other than on occasion, also excited all three of us. Sure enough, we fairly quickly

worked through the entire build process and had Moodle up and running in about an hour. Even so, we did run into a number of issues chasing our success.

Our first issue arose while editing the mysql configuration file. We ran into an error stating: "The Job for mysql.service failed because the control process exited with error code." Turns out we were running a later version of Ubuntu where the three lines identifying the storage engine, the db files per table variable, and the file format were not necessary. Accordingly, we went back and removed these lines (**Figure 1.4**).

Figure 1.4 - Lines we removed from mysql configuration file

```
default_storage_engine = innodb
```

```
innodb_file_per_table = 1
```

```
innodb_file_format = Barracuda
```

Another issue arose when dealing with the configuration.php file. We ran the command “`chmod -R 777 /var/www/html/moodle`” immediately followed by “`chmod -R 0755 /var/www/html/moodle`.” The first command allows reading, writing, and executing privileges to all users, but the problem arises when running the 0755 command immediately after the 777 command, which removes writing access to the users and essentially undoes part of the prior command. In the documentation, the flow is as if these commands should be run right after each other (**Figure 1.5**), however it would be

more clear to specify that the 0755 command should be called later when necessary. If run too early though, it causes issues in setting up the configuration.php.

Figure 1.5 - Confusing commands in installation guide

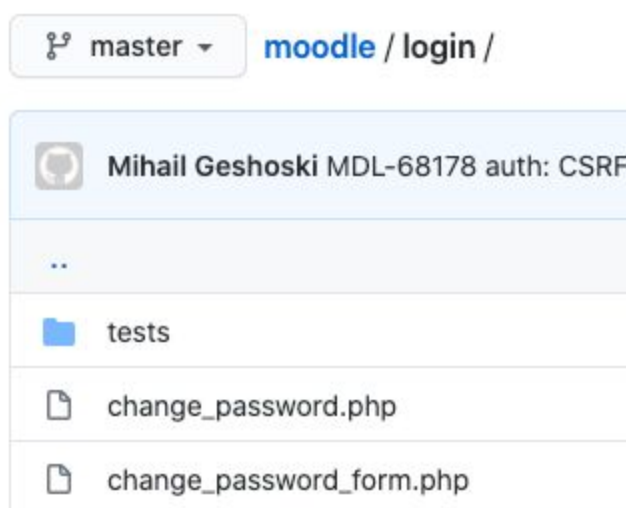
```
sudo chmod -R 777 /var/www/html/moodle
```

After you have ran the installer and you have moodle command.

```
sudo chmod -R 0755 /var/www/html/moodle
```

Now that we had Moodle set up properly and running on Localhost, it was time to examine the various folders of each component, and there are plenty from which to choose. For instance, in the login folder is a folder inside labeled “test” (**Figure 1.6**).

Figure 1.6 - Folders in Moodle



If we go a little further into the tests folder, we find a php file called lib_test.php with a class inside (**Figure 1.7**). It is becoming clear that we are going to need some other tools in order to run this unit test along with the others, but how do you unit test php test classes?

Figure 1.7 - lib_test.php in Moodle

```
class core_login_lib_testcase extends advanced_testcase {

    public function test_core_login_process_password_reset_one_time_without_username_protection() {
        global $CFG;

        $this->resetAfterTest();
        $CFG->protectusernames = 0;
        $user = $this->getDataGenerator()->create_user(array('auth' => 'manual'));

        $sink = $this->redirectEmails();

        list($status, $notice, $url) = core_login_process_password_reset($user->username, null);
        $this->assertSame('emailresetconfirmsent', $status);
        $emails = $sink->get_messages();
        $this->assertCount(1, $emails);
        $email = reset($emails);
        $this->assertSame($user->email, $email->to);
        $this->assertNotEmpty($email->header);
        $this->assertNotEmpty($email->body);
        $this->assertRegExp('/A password reset was requested for your account/', quoted_printable_decode($email->body));
        $sink->clear();
    }
}
```

In order to do this we needed two tools: Composer and PHPUnit. Composer is a dependency management tool for php while PHPUnit, as its name suggests, is a unit tester for php code. Getting Composer installed didn't give us much of an issue. However, when it came to getting the PHPUnit environment setup, there were a few extra things we had to figure out. We were confronted with our first issue when trying to run the Composer command to "require phpunit/phpunit" which would connect the phpunit dependency allowing us to set up the environment. The error message received was in regards to the version of phpunit that we were trying to get. When you run "php composer require phpunit/phpunit" the version that is brought in as a dependency is that

of the latest version, in our case version 9. The version of Moodle that we are running has unit tests that are only compatible with phpunit version 7, so after running the command “php composer require phpunit/phpunit ^7” we had success (**Figure 1.8**)!

Figure 1.8 - Success with Moodle

```
josh@josh-VirtualBox:/opt/moodle$ composer require --dev phpunit/phpunit ^7
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Nothing to install or update
Package container-interop/container-interop is abandoned, you should avoid using it. Use psr/container instead.
Package phpunit/dbunit is abandoned, you should avoid using it. No replacement was suggested.
Package phpunit/php-token-stream is abandoned, you should avoid using it. No replacement was suggested.
Writing lock file
Generating autoload files
12 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
```

Now it's time to run and everything should go perfectly as is always the case in running older software right? Wrong. We started to see dbdriver errors along with other errors that we weren't able to recognize at first, but after a while we started to notice a pattern (**Figure 1.9**). All of the errors had to do with variables within our configuration file. We were under the impression that these variables were set automatically given that we had successfully gotten moodle to run. However, even though we had the screen showing up properly, if we had tried to do something that involved accessing our mysql database, we would have run into some ugly issues. Thankfully we were able to catch this in the process of setting up the testing environment. So, sure enough, we entered the config.php file within our root moodle directory and noticed that certain variables had not been manually set (**Figure 1.10**). For example, “wwwroot” needs to be

set to “127.0.0.1/moodle” in our case. Another one of these examples was our dbdriver wasn’t set to “mysql” along with our username and password incorrect. Once we adjusted for these issues, we had some promising results (**Figure 1.11**).

Figure 1.9 - dbdriver error

```
Debug info:
Error code: dbdriverproblem
Stack trace: * line 140 of /lib/dml/pgsql_native_moodle_da
             database.php: dml_exception thrown
             * line 231 of /lib/dml/moodle_read_slave_trait.php: call t
             o pgsql_native_moodle_database->raw_connect()
             * line 217 of /lib/dml/moodle_read_slave_trait.php: call t
             o pgsql_native_moodle_database->set_dbhwrite()
             * line 340 of /lib/dml/lib.php: call to pgsql_native_moodle
             _database->connect()
             * line 624 of /lib/setup.php: call to setup_DB()
             * line 220 of /lib/phpunit/bootstrap.php: call to require(
             )
             * line 83 of /admin/tool/phpunit/cli/util.php: call to req
```

Figure 1.10 - Variable we had to manually set

```
// http://docs.moodle.org/en/masquerading
$CFG->wwwroot = 'http://example.com/moodle';
```

Figure 1.11 - Promising results

```
-->tinymce_moodleemoticon
++ Success ++
-->tinymce_moodleimage
++ Success ++
-->tinymce_moodlemedia
++ Success ++
-->tinymce_moodlenolink
++ Success ++
-->tinymce_pdw
++ Success ++
-->tinymce_spellchecker
++ Success ++
-->tinymce_wrap
++ Success ++
-->logstore_database
++ Success ++
-->logstore_legacy
++ Success ++
-->logstore_standard
++ Success ++

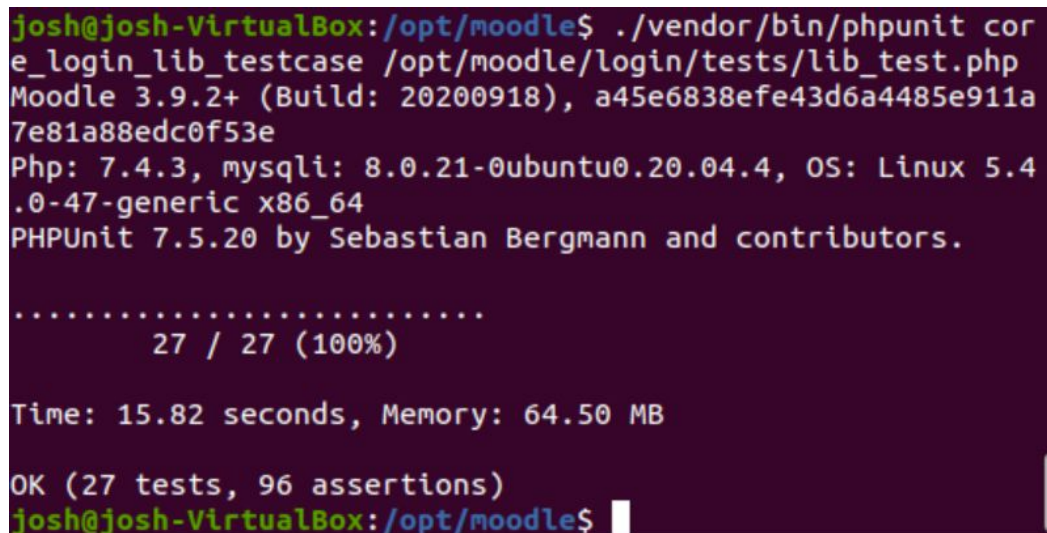
PHPUnit test environment setup complete.
josh@josh-VirtualBox: /opt/moodle$
```

As you can see in Figure 1.10 above, our PHPUnit testing environment has been properly set up. Now we can run some tests with the following command structure:

```
./path/to/phpunit    class to test  /path/to/test/file/to/test
```

So after running this command relevant to our machine, we got the output below (**Figure 1.12**). The output shows that 27 tests were run from this class and they were 100% successful with 96 assertions returning boolean expressions. We were thrilled to see a successful test! Along with the `lib_testcase`, we ran many other tests in many other folders for the different components of Moodle. The tests all had successful outcomes.

Figure 1.12 - Output from running some existing tests



```
josh@josh-VirtualBox:/opt/moodle$ ./vendor/bin/phpunit core_login_lib_testcase /opt/moodle/login/tests/lib_test.php
Moodle 3.9.2+ (Build: 20200918), a45e6838efe43d6a4485e911a7e81a88edc0f53e
Php: 7.4.3, mysqli: 8.0.21-0ubuntu0.20.04.4, OS: Linux 5.4.0-47-generic x86_64
PHPUnit 7.5.20 by Sebastian Bergmann and contributors.

.....
      27 / 27 (100%)

Time: 15.82 seconds, Memory: 64.50 MB

OK (27 tests, 96 assertions)
josh@josh-VirtualBox:/opt/moodle$
```

Reflection

In conclusion of Chapter 1, we clearly learned a lot as a team. One of the main takeaways from this initial stage was how difficult it was to build and run outdated software that is dependent on outdated/deprecated dependencies. Also configuring software to run on your machine can be particularly difficult depending on your familiarity with the particular software. For example, we really struggled in getting the pom.xml and settings.xml files to work properly in the Maven builds. This may be due to a lack of experience on our part or a lack of documentation on the part of the tool. In either case, we had some real success with Moodle because of the fabulous documentation of the system, which brings us to our final takeaway. The most important thing we learned through this phase is, documentation, documentation, documentation! It is so important to properly document your projects. It could be the difference between a good-doer successfully contributing to your project and the slow-creeping irrelevance of everything you've worked towards and accomplished.

Chapter 2 - Test Plan

Testing Process

- In order to test the desired method of the class we are interested in, we want to call a program at the command line that runs each test. The program will automatically run all of the tests one by one. The results of the tests will be compared to the expected results of the test cases, and then a test report with the results and other info will be displayed in a browser window.

Requirements Traceability

- TC01 | Creating a valid timestamp
- Timestamps are very important for every system, particularly for our endeavors in the Unix systems. In such systems, they are based on a running total of seconds that begins at the Unix epoch, which is advantageous as they can represent all timezones with a single measurement. As such, this facilitates users' ability to see a website (like Moodle) in real time as opposed to the timezone the server is located in.

Tested Items

- "Convert_to_timestamp" in the "type_base" class of the Moodle project
- /moodle/calendar/classes/type_base.php

Testing Schedule

- For the next week, we need to establish core functionality between script, driver, and php testing classes. Once established, we will spend this first week testing the Unix Timestamp method of the type_base class with its many test cases. The following weeks will be dedicated to applying the same techniques to other test

cases that we will discover. Methods that use any arithmetic to calculate grades, or calendar dates would be ideal. Successful connections to the database/browser could be another test. We are currently looking into this and several other possibilities.

Test Recording Procedures

- All of the actual test results will be compared to the expected test results contained in various appropriately labeled json files.
- The results of the comparisons described above will be displayed in a browser. The information will be properly and clearly labeled in a uniform manner with each test case getting its own portion of the report.

Hardware and Software Requirements

- Linux OS
- A computer

Constraints

- We have a limited schedule to figure out the relation of different pieces to our framework.

Chapter 3 - Designing & Building our Testing Framework

To say that deliverable 3 encompassed the lion's share of the work for our project would be an understatement, and this is something the Another-One-Bytes-the-Dust team recognized immediately. We quickly allocated responsibilities and laid out a schedule according to our test plan to hammer out designing and building the initial script as well as our first driver. Despite some hurdles in the implementation process, sticking to our schedule greatly aided us in working diligently and efficiently to achieve not just results but outright success. Over the course of this chapter, we will discuss designing and building our initial driver and script before moving on to some of the hurdles we faced and some of the innovative solutions we found, then finally we will give a brief reflection on our experiences through this phase of the project.

Section 1 - Building the Driver

Considering the script partially relies on output from the drivers to function properly, we decided to tackle the initial driver before the script. Josh and Clae set to work designing and implementing the driver while Alex chipped away at the script. Designing our driver was fairly simple. We imported the proper files, decoded our json test case files into variables we could work with (i.e. variables that contain all of the attributes of the test cases such as ID number, method tested, testing inputs, etc.), and set to work creating the array the driver will return as a long string. Next, we set the attributes for each test case in the return array according to the attributes in the corresponding spot in the array containing our data on the test cases. At this point, we

actually call our test method `parse_charset` (seen in **Figure 3.1**), which converts text from uppercase to lowercase and formats some strings according to various rules. We then compare the actual output from the test method with the expected output from our array of test cases and pass the result of each comparison to our return array for each test case. At this point, we return the array of test case data (now with the actual output and a pass/fail variable) from our driver to our script.

Figure 3.1 - Test method `parsecharset`

```

130     public static function parse_charset($charset) {
131         $charset = strtolower($charset);
132
133         // shortcuts so that we do not have to load typo3 on every page
134
135         if ($charset === 'utf8' or $charset === 'utf-8') {
136             return 'utf-8';
137         }
138
139         if (preg_match('/^(cp|win|windows)-?(12[0-9]{2})$/', $charset, $matches)) {
140             return 'windows-'. $matches[2];
141         }
142
143         if (preg_match('/^iso-8859-[0-9]+$/', $charset, $matches)) {
144
145             return $charset;
146         }
147
148         if ($charset === 'euc-jp') {
149             return 'euc-jp';
150         }
151         if ($charset === 'iso-2022-jp') {
152             return 'iso-2022-jp';
153         }
154         if ($charset === 'shift-jis' or $charset === 'shift_jis') {
155             return 'shift_jis';
156         }
157         if ($charset === 'gb2312') {
158             return 'gb2312';
159         }
160         if ($charset === 'gb18030') {
161             return 'gb18030';
162         }
163
164         // fallback to typo3
165         return self::typo3()->parse_charset($charset);
166     }

```

One important step we took to build our driver properly was constructing our own class, which can be seen in **Figure 3.2**. This class is just a simple object called `driverObject` that contains all of the information of a test case (including test case number, driver name, testing inputs, etc.). We also included getter and setter methods for each of the variables included in the object. In our code for the driver, we use a number of these objects in an array, with one object per test case, and we set each variable for each object according to the info from our json files, the output from our test method, and whether that output matches the expected output. Then we build a long string from all of the data in our array of objects and return that string for the script to work with.

Figure 3.2 - driverObject class we wrote

```
class driverObject {  
    var $tcID;  
    var $requirement;  
    var $driver;  
    var $class;  
    var $method;  
    var $testingInput;  
    var $expectedOutput;  
    var $success;  
  
    function settcID($id){  
        $this->tcID = $id;  
    }  
    function gettcID(){  
        echo $this->tcID."$$$";  
    }  
    function setRequirement($id){
```

While this seemed simple, initially our driver was much more linear and inefficient. In this previous design the construction of the arrays and process of comparing outputs and returning the long string were very similar. The primary difference is that we didn't have a loop implemented in the older design, so essentially we had a block of code devoted to each test case. Maintaining this code would be needlessly arduous and inefficient, so we felt the need to redo it to a small extent. The result of this redesign is the current driver, now with a loop that will loop through each test case and do the work, rather than 5 individual blocks of code that each run a very similar test. At the end of the day, we're happy with our code which can be seen in **Figure 3.3**.

Figure 3.3 - Our initial driver

```

1 <?php
2 //error_reporting(0);
3 //TEST STRING TO LOWER & REGULAR EXPRESSIONS
4
5 define('CLI_SCRIPT', true);
6 require("driverObject.php");
7 require_once("../project/moodle1/config.php");
8 require("../project/moodle1/user/lib.php");
9 require("../project/moodle1/lib/classes/text.php");
10
11 // Create Object of our Selected "Core Text" Class
12 $text = new core_text;
13
14 // Decode Json Objects from Test Case Folder
15 $tc01 = json_decode(file_get_contents("../testCases/TC01.json"));
16 $tc02 = json_decode(file_get_contents("../testCases/TC02.json"));
17 $tc03 = json_decode(file_get_contents("../testCases/TC03.json"));
18 $tc04 = json_decode(file_get_contents("../testCases/TC04.json"));
19 $tc05 = json_decode(file_get_contents("../testCases/TC05.json"));
20 $obj = new driverObject;
21
22 // Fill Array with Empty Objects for Test Case Info
23 $finalArr = array($obj, $obj, $obj, $obj, $obj);
24
25 // Fill Array of Test Case Decode Json
26 $testCaseArr = array($tc01, $tc02, $tc03, $tc04, $tc05);
27

```

```

for ($i = 0; $i < count($finalArr); $i++){
    $input = $testCaseArr[$i]->testingInputs;
    $expectedOut = $testCaseArr[$i]->expectedOutput;

    //Establish Object
    $finalArr[$i]->settcID($testCaseArr[$i]->testId);
    $finalArr[$i]->setRequirement($testCaseArr[$i]->requirement);
    $finalArr[$i]->setDriver($testCaseArr[$i]->driver);
    $finalArr[$i]->setClass($t
    <p>Requirements: %s</p>$testCaseArr[$i]->classTested);
    $finalArr[$i]->setMethod($testCaseArr[$i]->methodTested);
    $finalArr[$i]->setTestingInput($testCaseArr[$i]->testingInputs);
    $finalArr[$i]->setExpectedOutput($testCaseArr[$i]->expectedOutput);

    $parsed_charset = $text->parse_charset($input);

    if ($parsed_charset == $expectedOut){
        $finalArr[$i]->setSuccess("PASS");
    } else{
        $finalArr[$i]->setSuccess("FAIL");
    }

    $finalArr[$i]->gettcID();
    $finalArr[$i]->getRequirement();
    $finalArr[$i]->getDriver();
    $finalArr[$i]->getClass();
    $finalArr[$i]->getMethod();
    $finalArr[$i]->getTestingInput();
    echo $parsed_charset."$$$";
    $finalArr[$i]->getExpectedOutput();
    $finalArr[$i]->getSuccess();
}

```

Section 2 - Building the Script

While the driver was being worked on by Josh and Clae, Alex put her efforts into the script. Despite the fact that we'd spent some of our spare time working on the script previously, we knew it would be difficult to fully implement as it's the piece that unites all of the parts of our automated testing framework. And sure enough, it was one of the more challenging components of our project up to this point, but we'll discuss some of those challenges in the next section. Alex made considerable progress nonetheless, and thankfully she received some help as soon as the driver was running perfectly.

After the necessary import statements, our python script starts by running our driver as a php subprocess and getting a long string-like object (seen in **Figure 3.4**) from our driver that contains all of our data for each of our 5 test cases. Our script then creates and formats the html file that will display a test report in a browser (which can be seen in **Figure 3.5**). It's at this point that our script converts the string-like object to a string, splits it by test case, then splits each test case by each attribute. This enables us to write each attribute for each test case to the formatted html file so we can then close and display it in a browser. While seemingly a lot of code, most of the heavy lifting comes from a small portion which can be seen in the loop in **Figure 3.6**.

Figure 3.4 - String the driver is returning

```
CasesExecutables$ php TC01_05Driver.php
TC01$$$Method parses and returns charset so all characters are lower-case$$$TC01
_05Driver.php$$$core_text$$$parse_charset$$$UTF-8$$$utf-8$$$utf-8$$$PASS***TC02$
$$$Method parses and returns charset so all characters are lower-case$$$TC01_05Dr
iver.php$$$core_text$$$parse_charset$$$ISO-2022-JP$$$iso-2022-jp$$$iso-2022-jp$$$
PASS***TC03$$$Method parses and returns charset so all characters are lower-cas
e$$$TC01_05Driver.php$$$core_text$$$parse_charset$$$utf-10$$$utf-10$$$utf-10$$$P
ASS***TC04$$$Method parses and returns charset so all characters are lower-case$
$$$TC01_05Driver.php$$$core_text$$$parse_charset$$$utf-8$$$utf-8$$$utf-8$$$PASS***
TC05$$$Method parses and returns charset so all characters are lower-case$$$TC01
_05Driver.php$$$core_text$$$parse_charset$$$shift-jis$$$shift_jis$$$shift_jis$$$
PASS***josh@josh-VirtualBox:~/Desktop/SE/Another-One-Bytes-the-Dust/TestAutomati
```

Figure 3.5 - Our initial test report

AOBTD TESTING FRAMEWORK

Test ID: TC01
Requirements: Method parses and returns charset so all characters are lower-case
Driver: TC01_05Driver.php
Class Tested: core_text
Method Tested: parse_charset
Testing Input: UTF-8
Actual Output: utf-8
Expected Output: utf-8
Success or Fail: PASS

Test ID: TC02
Requirements: Method parses and returns charset so all characters are lower-case
Driver: TC01_05Driver.php
Class Tested: core_text
Method Tested: parse_charset
Testing Input: IsO-2022-jP
Actual Output: iso-2022-jp
Expected Output: iso-2022-jp
Success or Fail: PASS

Figure 3.6 - Our initial script

```

1 #! /usr/bin/python3.8
2
3 import sys
4 import subprocess
5 import webbrowser
6 import time
7
8 result = subprocess.run(
9     ['php', '../testCasesExecutables/TC01_05Driver.php'],
10     stdout=subprocess.PIPE,
11     check=True
12 )
13
14 htmlOpening = '''
15 <!DOCTYPE html>
16 <head>
17 <style>
18 .header{
19     text-align: center;
20     font-size: 100px;
21     font-family: Arial, Helvetica, sans-serif;
22 }
23
24 .container{
25     width: 90%;
26     margin-left: auto;
27     margin-right: auto;
28     height: 380px;
29     background-color: #f2f2f2;
30 }
31 </style>
32 <meta charset="utf-8">
33 <title>Test Report</title> </head>
34 <body>
35 <h1 class="header">AOBTD TESTING FRAMEWORK</h1>
36 f = open("testReport.html", "w")
37 f.write(htmlOpening)
38
39 x = str(result.stdout)
40 y = x.split('***')
41 #print(y)
42 outstring = ""
43 for case in y:
44     attr = case.split('$$$')
45     if (len(attr) == 9):
46         whole = innerText % (attr[0].replace("b'", "'"), attr[1], attr[2], attr[3], attr[4], attr[5], attr[6], attr[7], attr[8])
47         f.write(whole)
48
49 f.write(htmlClosing)
50 f.close()
51 webbrowser.open("testReport.html")

```

Section 3 - Hurdles Faced

As with every part of the project so far, we faced a number of hurdles we had to overcome. Aside from the typical struggles with technology/computers and simple errors like syntax errors, we faced two real challenges during this phase. The first involved a config file that we didn't realize had to be tailored to each individual user. The other problem was actually related to this issue, though is a different issue altogether involving the same file.

Our initial issue was relatively easy to solve. Having configured the script in a way that it should at least run, two of us were receiving a rather obscure error message about a file named config.php while the third group member's code was running perfectly. After some investigating, we realized that all three of us had our config.php files set with the Moodle database details for the group member whose code was working. After correcting these discrepancies by filling the config.php file with our own Moodle database details, our code ran without any issue, yet this wasn't the end of the hurdles we faced.

The other significant issue came with the resolution of our previous dilemma. Having personalized our config.php files, we realized that pushing our updates to our github repo and pulling them would write over each other's config.php files, thus necessitating us to perform the same fix from the previous issue each time we pull from our repo. To solve this, it was decided that we should include our config.php file in a .gitignore folder so that github will ignore the config.php file when we push to and pull from our repo. These were the biggest issues we faced over the course of this deliverable, and thankfully we seem to have a handle on them.

Section 4 - Reflection

All in all this deliverable comprised a huge portion of the work required to complete this project, and it feels good to have delivered what we feel is a strong performance, at least before any constructive criticism. Having worked through our issues with our initial driver and having created a functional version of our script, we now have a roadmap on how to implement the other drivers and simply need to adjust

the script slightly so it can handle multiple drivers. At this point, we are feeling not just comfortable with where we are, but confident that we can quickly complete the next deliverable with little difficulty.

Chapter 4 - Full Testing Suite

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. Consequently, the code pictured in the previous deliverables is all outdated and now vastly different from our current code. 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 obstacles we overcame before giving a brief reflection on our experiences developing this deliverable.

Section 1 - The Refactored 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 of 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 simplified things and rendered 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 drivers' 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 - Refactored parse_charset driver

```

1  <?php
2  //DRIVER FOR PARSECHARSET METHOD
3  define('CLI_SCRIPT', true);
4  require_once("../project/moodle1/config.php");
5  require("../project/moodle1/lib/classes/text.php");
6
7  if (!$argv[1]) {
8      throw new Exception('Inputs are invalid...');
9  }
10 try {
11     $text = new core_text;
12     $output = $text->parse_charset($argv[1]);
13     print_r($output);
14 } catch (Exception $e) {
15     print_r('Caught exception: '.$e->getMessage()."\n");
16 }
17 ?>

```

Figure 4.2 - parse_charset method signature in Moodle

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

Figure 4.3 - break_up_long_words driver

```

1  <?php
2  //error_reporting(0);
3  //TEST STRING TO LOWER & REGULAR EXPRESSIONS
4
5  define('CLI_SCRIPT', true);
6  require_once("../project/moodle1/config.php");
7  require("../project/moodle1/user/lib.php");
8
9  if (!$argv[1]) {
10     throw new Exception('Inputs are invalid...');
11 }
12 try {
13     $splitStrings = explode(" ", $argv[1]);
14     $output = break_up_long_words($splitStrings[0], intval($splitStrings[1]), $splitStrings[2]);
15     print_r($output);
16 } catch (Exception $e) {
17     print_r('Caught exception: '.$e->getMessage()."\n");
18 }
19
20 ?>
```

Figure 4.4 - break_up_long_words method signature in Moodle

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

Section 2 - The Refactored Script

After realizing our error and rebuilding our driver, we knew we had to put all of the functionality we stripped out of the driver back into our script to make a functional automated 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 by contrast is actually very simple (seen in **Figure 4.5**). It's nothing more than some json reading, variable creation, method calls, and 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 to sort the test cases by driver, 1 for test cases by test id). 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 refactored 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])

```

Section 3 - The New 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 without adding additional code. 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 and test cases as the user could write without any adjustments. 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 would be a cheap move to use that class more than we had to. 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 anyway. This was likely for the best as the methods we found work very well for our purposes.

Two other small issues we had were trying to figure out how to call our script from the correct place and having it run, and needing to redesign our test report. The first issue 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 our runAllTests script. Regarding our test report, we felt that it was far too vertical, so we redesigned it to be more horizontal which can be seen in **Figure 4.7**. Having remedied these final errors, we feel that we have adequately completed deliverable 4 and we are confident in our ability to accomplish everything for deliverable 5.

Figure 4.7 - Our Redesigned Test Report

AOBTD TESTING FRAMEWORK

Test ID	Requirements	Driver	Class Tested	Method Tested	Testing Input	Actual Output	Expected Output	Pass/Fail
NV01	Normalize a string to be a series of numbers	normalizeDriver.php	environment_lib	normalize_version	1.135...	135	135	Pass
NV02	Normalize a string to be a series of numbers	normalizeDriver.php	environment_lib	normalize_version	1.2.3.4	1.2.3.4	1.2.3.4	Pass
NV03	Normalize a string to be a series of numbers	normalizeDriver.php	environment_lib	normalize_version	1.2.3	1.2.3	1.2.3	Pass
NV04	Normalize a string to be a series of numbers	normalizeDriver.php	environment_lib	normalize_version	hello00world...	000	000	Pass
NV05	Normalize a string to be a series of numbers	normalizeDriver.php	environment_lib	normalize_version	hello_2	2	2	Pass
PC01	Method parses and returns charact so all characters are lower-case	parseCharactDriver.php	core_text	parse_charact	UTF-8	ut-8	ut-8	Pass
PC02	Method parses and returns charact so all characters are lower-case	parseCharactDriver.php	core_text	parse_charact	iso-2022-jp	iso-2022-jp	iso-2022-jp	Pass
PC03	Method parses and returns charact so all characters are lower-case	parseCharactDriver.php	core_text	parse_charact	iso-100	iso-100	iso-100	Pass
PC04	Method parses and returns charact so all characters are lower-case	parseCharactDriver.php	core_text	parse_charact	utf-8	utf-8	utf-8	Pass
PC05	Method parses and returns charact so all characters are lower-case	parseCharactDriver.php	core_text	parse_charact	shift_jis	shift_jis	shift_jis	Pass
RD01	Rounds a given value with precision 0	roundDriver.php	evalmath	round	12	1	1	Pass
RD02	Replaces characters in the string with HTML entity escapes	roundDriver.php	evalmath	round	5.234	5	5	Pass
RD03	Rounds a given value with precision 0	roundDriver.php	evalmath	round	-45.0	-45	-45	Pass
RD04	Rounds a given value with precision 0	roundDriver.php	evalmath	round	743.01	743	743	Pass
RD05	Rounds a given value with precision 0	roundDriver.php	evalmath	round	001	0	0	Pass
LW01	Method breaks up a string into multiple parts with a symbol separator	breakUpLpLomWordsDriver.php	weblib	break_up_long_words	arnoldmcdock@gmail.com*"5"@"	arnold@mcdock@igmail@l.com	arnold@mcdock@igmail@l.com	Pass
LW02	Method breaks up a string into multiple parts with a symbol separator	breakUpLpLomWordsDriver.php	weblib	break_up_long_words	helloworldhelloworld*10^555	helloworldSShelloworld	helloworldSShelloworld	Pass
LW03	Method breaks up a string into multiple parts with a symbol separator	breakUpLpLomWordsDriver.php	weblib	break_up_long_words	anotherbytesducs*4%	anotherbytesducs4%	anotherbytesducs4%	Pass
LW04	Method breaks up a string into multiple parts with a symbol separator	breakUpLpLomWordsDriver.php	weblib	break_up_long_words	simplestring*100%	simplestring	simplestring	Pass
LW05	Method breaks up a string into multiple parts with a symbol separator	breakUpLpLomWordsDriver.php	weblib	break_up_long_words	ABC@DEF@GHIJK*"5"@"	ABC@DEF@G@HIJK@L	ABC@DEF@G@HIJK@L	Pass
IT01	Removes TRUNTEXT label from string	breakTestDriver.php	weblib	breaktest_strip	hellorun@TRUNTEXT000000world	helloworld	helloworld	Pass
IT02	Removes TRUNTEXT label from string	breakTestDriver.php	weblib	breaktest_strip	00000TRUNTEXT000000000000TRUNTEXT000000	hey	hey	Pass
IT03	Removes TRUNTEXT label from string	breakTestDriver.php	weblib	breaktest_strip	000000TRUNTEXT000000000000	000000TRUNTEXT000000	000000TRUNTEXT000000	Pass
IT04	Removes TRUNTEXT label from string	breakTestDriver.php	weblib	breaktest_strip	00000TRUNTEXT000000	00000TRUNTEXT000000	00000TRUNTEXT000000	Pass
IT05	Removes TRUNTEXT label from string	breakTestDriver.php	weblib	breaktest_strip	0000000TRUNTEXT000000	0000000TRUNTEXT000000	0000000TRUNTEXT000000	Pass

Section 4 - Reflection

This deliverable was particularly difficult for the team. We thought that most of the work would be done after deliverable 3, and 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 methods we might be able to use. 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.

Chapter 5 - Error Insertion and Analysis

After the fiascos that were deliverables 3 and 4, the Another-One-Bytes-the-Dust team was looking forward to a nice laid back completion of deliverable 5. Though we did have a fairly nice time injecting errors into the methods we're testing and documenting our work, it was not without surprises. Despite the fun, we managed to learn a fair bit from our efforts. Below, we break down the changes we made to the Moodle code to inject our errors and the results of running our testing suite with the altered code, before giving a brief reflection of our work on this deliverable.

Section 1 - normalize_version

Our first error injection went essentially exactly as intended. This method takes a string of letters and numbers, and normalizes it to look like a software version number (ex: 5.2.4.3). Part of the method is responsible for removing any duplicate dots present, so the return would be something like 1.3.4.6 instead of something such as 1..3.4....6 which is clearly incorrect. One of our test cases has a test input with multiple dots between numbers, so we commented out this line of code (**Figure 5.1**). Sure enough, the resulting output for that test case had duplicate dots between the otherwise correct digits, and our testing suite marked that test as having failed since the expected and actual outputs now differed. All of our other test cases were unaffected as no other test input had unnecessary dots. All of this was completely expected.

Figure 5.1 - normalize_version w/ injection (line 252)

```

232  /**
233   * This function will normalize any version to just a serie of numbers
234   * separated by dots. Everything else will be removed.
235   *
236   * @param string $version the original version
237   * @return string the normalized version
238   */
239  function normalize_version($version) {
240
241    /// 1.9 Beta 2 should be read 1.9 on enviromental checks, not 1.9.2
242    /// we can discard everything after the first space
243    $version = trim($version);
244    $versionarr = explode(" ", $version);
245    if (!empty($versionarr)) {
246        $version = $versionarr[0];
247    }
248    /// Replace everything but numbers and dots by dots
249    $version = preg_replace('/[^\.\d]/', '.', $version);
250    /// Combine multiple dots in one
251    /// [THIS IS AN INJECTION WHERE WE REMOVE THE LINE THAT REMOVES MULTIPLE DOTS]
252    //$version = preg_replace('/(\.{2,})/', '.', $version);
253    /// Trim possible leading and trailing dots
254    $version = trim($version, '.');
255
256    return $version;
257  }

```

Section 2 - round:

Our second error injection was also pretty by the book in terms of what we expected. In our round method (which actually functions more like a method for truncating than rounding) we added lines that would throw an error if the testing input was less than zero (**Figure 5.2**). This affected only one of our test cases since only one test case had a test input that was less than zero. As a result, that test now failed since

the expected output was no longer a number, but an error. All of our other tests still passed without issue. None of this came as any real surprise to the team.

Figure 5.2 - round w/ injection (line 603)

```

597     static function round($val, $precision = 0) {
598         // return round($val, $precision);
599         /// [THIS IS AN INJECTION WHERE IF VAL IS LESS THAN 0, ERROR MESSAGE RETURNED]
600         if ($val > 0){
601             return round($val, $precision);
602         } else {
603             return "You have entered a number less than 0..";
604         }
605     }

```

Section 3 - trusttext_strip:

This error injection was another one that was not particularly surprising. This method accepts a string with a marker that needs to be removed and returns the string without the marker. Initially, the marker the method removes is "#####TRUSTTEXT#####" but we altered the function to instead search for both that and "#####TRUSTEXT#####" (we removed one of the middle T's. Seen in **Figure 5.3**). Unsurprisingly, this resulted in a failure for one of our test cases in which part of our input included "#####TRUSTTEXT#####". The expected output for this test case is that the text will not be replaced, so the output will include "#####TRUSTTEXT#####" but since we injected an error to address this input, this input was caught and replaced with an empty space so the test failed. Again, nothing exactly unexpected in our experiment with this error.

Figure 5.3 - trusttext_strip w/ injection (line 1634)

```

1620  /**
1621   * Legacy function, used for cleaning of old forum and glossary text only.
1622   *
1623   * @param string $text text that may contain legacy TRUSTTEXT marker
1624   * @return string text without legacy TRUSTTEXT marker
1625   */
1626  function trusttext_strip($text) {
1627      if (!is_string($text)) {
1628          // This avoids the potential for an endless loop below.
1629          throw new coding_exception('trusttext_strip parameter must be a string');
1630      }
1631      while (true) { // Removing nested TRUSTTEXT.
1632          $orig = $text;
1633          /// [THIS IS AN INJECTION WILL NOW HANDLE TRUSTTEXT AS VALID TRUSTTEXT]
1634          $text = str_replace('#####TRUSTTEXT#####', '#####TRUSTTEXT#####', $text);
1635          $text = str_replace('#####TRUSTTEXT#####', '', $text);
1636          if (strcmp($orig, $text) === 0) {
1637              return $text;
1638          }
1639      }
1640  }

```

Section 4 - break_up_long_words:

Our fourth error injection took place within this method. As the name suggests, this function accepts a string, a maximum length, and the character we want to use to cut up any long words. It returns the same string but any word within the string over the maximum length specified gets chopped up by the cut character at the index position corresponding to max length. In other words, any word over the max length gets the cut character inserted between the index of the word that is the maximum allotted by the max length and the char that causes the word to go over the max length.

For our error injection here, we added a new if statement that exits the loop which actually does the work of cutting the long words in the string if the string contains the character 'A' (**Figure 5.4**). The result is that any long words after the 'A' will not be cut should they exceed the max length. As we have a single test case input that starts with the character 'A' we expected this test case to fail terribly while the rest of our test cases passed. This was indeed the case. The affected test case failed as the actual and expected outputs differed, and none of the other test cases were affected by this change. However, things got very interesting with this method when we made our final injection.

Figure 5.4 - break_up_long_words w/ injection (lines 1010-1012)

```

986  /**
987   * Given some normal text this function will break up any
988   * long words to a given size by inserting the given character
989   *
990   * It's multibyte savvy and doesn't change anything inside html tags.
991   *
992   * @param string $string the string to be modified
993   * @param int $maxsize maximum length of the string to be returned
994   * @param string $cutchar the string used to represent word breaks
995   * @return string
996   */
997  function break_up_long_words($string, $maxsize=20, $cutchar=' ') {
998
999      // First of all, save all the tags inside the text to skip them.
1000      $tags = array();
1001      filter_save_tags($string, $tags);
1002
1003      // Process the string adding the cut when necessary.
1004      $output = '';
1005      $length = core_text::strlen($string);
1006      $wordlength = 0;
1007
1008      for ($i=0; $i<$length; $i++) {
1009          $char = core_text::substr($string, $i, 1);
1010          if ($char == 'A'){
1011              continue;
1012          }
1013          if ($char == ' ' or $char == "\t" or $char == "\n" or $char == "\r" or $char == "<" or $char == ">") {
1014              $wordlength = 0;
1015          } else {
1016              $wordlength++;
1017              if ($wordlength > $maxsize) {
1018                  $output .= $cutchar;
1019                  $wordlength = 0;
1020              }
1021          }
1022          $output .= $char;
1023      }

```


Section 5 - parsecharset:

This method is where things got very interesting very quickly. This method for the most part takes a set of chars and replaces any uppercase chars with their lowercase counterparts. Importantly, this method has a small number of special cases that take an additional step, possibly replacing an underscore with a dash or something along similar lines. Since they seemed fairly important, we decided to mess with one of those special cases. One of our test inputs was “UTF-8” which is specifically handled and supposed to return as “utf-8” but our error injection made the program return “utf-7” instead (**Figure 5.5**). We expected this to result in having a single test case now fail just as how it’d happened in our other injections. We were shocked with the actual results.

While we did get a failure for the test case we intended to fail, we also got a surprise. One of our test cases for `break_up_long_words` also failed. We then confirmed that the error we injected in `break_up_long_words` was commented out, the code was restored to its original state, and that we’d made no other changes to cause this error. The result was consistent until we rectified the error we injected into `parsecharset`, after which the test case for `break_up_long_words` (that wasn’t supposed to change in the first place) was now back to how it should be. All of this can be seen in the snapshot of our testing report with the injected errors in **Figure 5.6**. This was something that greatly surprised us, and something we will discuss in the next section.

Figure 5.5 - parsecharset w/ injection (lines 136-138)

```

121     /**
122     * Standardise charset name
123     *
124     * Please note it does not mean the returned charset is actually supported.
125     *
126     * @static
127     * @param string $charset raw charset name
128     * @return string normalised lowercase charset name
129     */
130     public static function parse_charset($charset) {
131         $charset = strtolower($charset);
132
133         // shortcuts so that we do not have to load typo3 on every page
134
135         if ($charset === 'utf8' or $charset === 'utf-8') {
136             //return 'utf-8';
137             /// [THIS IS AN INJECTION WHERE SET THE RETURN IF UTF8 to UTF7]
138             return 'utf-7';
139         }
140
141         if (preg_match('/^(cp|win|windows)-?(12[0-9]{2})$/i', $charset, $matches)) {
142             return 'windows-'. $matches[2];
143         }
144
145         if (preg_match('/^iso-8859-[0-9]+$/i', $charset, $matches)) {
146
147             return $charset;
148         }
149
150         if ($charset === 'euc-jp') {
151             return 'euc-jp';
152         }
153         if ($charset === 'iso-2022-jp') {
154             return 'iso-2022-jp';
155         }
156         if ($charset === 'shift-jis' or $charset === 'shift_jis') {
157             return 'shift_jis';
158         }
159         if ($charset === 'gb2312') {
160             return 'gb2312';
161         }
162         if ($charset === 'gb18030') {
163             return 'gb18030';
164         }

```


hole calls it), so altering a small part of the parsecharset method might ultimately affect the outcome for other methods we're using. This is something that will require a significant amount of investigation.

All in all, the group is feeling pretty good about the project. Our script, drivers, test cases, etc are all where we want them to be, which is in line with the guidance provided by Professor Bowring. At this point, we simply need to make a poster and work on our final presentation for our entire project. We are certainly looking forward to showing off our efforts.

Chapter 6 - Overall Experiences and Lessons Learned

Overall Reflection

As a whole, this project was not what any of us expected. We all thought that working on an open source project and developing a program such as the one we wrote would be different than from how it actually was. Of course while we each had our own unique expectations, we were all still surprised by how even the simple things such as working in a team turned out to be different. Nonetheless it was still a very challenging and constructive endeavor, and it's safe to say that we all learned something.

I think one of the biggest misconceptions we had about working with an open source project is that we figured it would be, well, better. Moodle is obviously a large project with years of development behind it, something that in and of itself is impressive. And for the most part, Moodle not only functioned well, but appeared to be fairly polished in terms of documentation and project architecture. However, we were still surprised at how some small things within Moodle seemed wrong or misleading, such as how our round test method actually just truncated the input. Perhaps a more glaring mistake is that some methods lacked any significant documentation. Considering the size of the project, we also found this to be surprising.

One of the other surprising factors for us was the need for efficiency. We covered the refactor process fairly in depth previously in this report, but one small piece of that refactor process involved reworking some code that worked terribly inefficiently. While we actually never ran into this issue, we were aware that another project group with a similar strategy ran into an issue that caused their program to crash. Clearly this scared us enough to go and quickly solve the issue so that our program won't crash as a result

of our being inefficient. In hindsight, this makes perfect sense, but it was a tad surprising to us that such a seemingly innocuous design choice could wreak havoc like that down the road.

One of the final surprises for us was learning how to work as a team. We pretty quickly divvied up the roles and duties we would each have throughout the course of this project, but we weren't always good at sticking to them. Ultimately we managed to push out a product that we can all be proud of, but there were moments when the roles shifted because things needed to get done. None of this is to say that we clashed as a team or anything like that, in fact we remained largely disagreement free and never degenerated into any sort of argument. Still, the changing team dynamics and responsibilities on the fly and as we encountered problems was a surprise to each of us.

Final Conclusion

In conclusion, we think we did a fairly good job building this project. None of us had any prior experience constructing this sort of project, so none of us knew what to expect. While we had some surprises, the fact that we learned quite a bit was not one of the surprises. Working with an open source project in such a way really let us get our virtual hands dirty, as if we were in the trenches in battle. Working with such a massive project was really an invaluable experience, even if it was a baptism by fire sort of deal. At the end of the day, the experiences we gleaned from this project will likely serve as a fantastic foundation on which to build our future software engineering endeavours.

Chapter 7 - Evaluations

Self evaluation (team)

All in all, we think our team did pretty well. As best as we can tell, we completed the project according to the specifications provided. We did so with little drama (at least in our opinion) and always conducted ourselves in a civil and professional manner. We also think our project is clean and well documented throughout. The methods we chose to work with for our testing suite were also more complex than just common getters and setters, which made for a more interesting project. We also found some fascinating results from our error injections and did our best to explore and document them. None of this is to say that we did everything perfectly, as we're sure there are some areas in which we could improve. In the end, we still think we did a pretty good job.

Project Assignments and Suggestions

We all agree that it's a great project. As Professor Bowring has maintained this whole time, there's really no better way to learn this stuff than to just jump in and do it. We don't remember if this came up in our class, but we were wondering if Professor Bowring would've been open to a group working with an open source project that was not on the wikipedia page he provided us. We also think it would be beneficial to spend some time in class going over the specifications for the project right before the students are supposed to build their script and initial driver. This might have helped eliminate some of the confusion we had after trying to interpret the document with the specifications, as well as possibly prevented us from designing/building our initial driver and script in an incorrect fashion and having to go back and completely redo them.

Presentation Material

Google Slides Presentation:

<https://docs.google.com/presentation/d/1-uqWyUDmOo5nPsCWwX0GzYsCPSy9nzs9T1YgonWHJF4/edit?usp=sharing>

Poster:

Why moodle?

Moodle attracted the interest of our group both because of its level of documentation, and because we felt as a group that we could work effectively with this project. The prospect of working with php, a language nobody on our team has used other than on occasion, also excited all three of us. Sure enough, we fairly quickly worked through the entire build process and had Moodle up and running in about an hour.



Test Plan

Testing Process

In order to test the desired method of the class, we are interested in we want to call a program at the command line that runs each test. The program will automatically run all of the tests one by one. The results of the tests will be compared to the expected results of the test cases, and then a test report with the results and other info will be displayed in a browser window.

Requirements Traceability

"TOI" (Creating a valid timestamp)

Timestamps are very important for every system, particularly for our end users in the Unix systems. In such systems, they are based on a running total of seconds that begins at the Unix epoch, which is advantageous as they can represent all timestamps with a single measurement. As such, this facilitates user ability to use a website like Moodle in real time as opposed to the times when the server is located in.

Tested Items

"Convert, to timestamp" in the type_base class of the Moodle project/moodle/calendar/case/type_base.php

Testing Schedule

For the next week, we need to establish core functionality between script, driver, and php testing classes. Once established, we will spend this first week testing the Unix Timestamp method of the type_base class with its many test cases. The following weeks will be dedicated to applying the same techniques to other test cases that we will discover. Methods that use any arithmetic to calculate grades or calendar dates would be ideal. Successful connections to the database browser could be another test. We are currently looking into this and several other possibilities.

Test Recording Procedures

All of the actual test results will be compared to the expected test results contained in various appropriately labeled json files.

The results of the comparisons described above will be displayed in a browser. The information will be properly and clearly labeled in a uniform manner with each test case getting its own portion of the report.

Hardware and Software Requirements

Linux OS

Computer

Conditions

We have a limited schedule to figure out the relation of different pieces to our framework.

Building our first Driver & Script

Initial Driver

Initial Script

Output



Hurdles Faced

As with every part of the project so far, we faced a number of hurdles we had to overcome. Aside from the typical struggles with technology, computers, and simple errors like syntax errors, we faced two real challenges during this phase. The first involved a config file that we didn't realize had to be tailored to each individual user. The other problem was actually related to this issue, though it is a different issue altogether involving the same file.

Our initial issue was relatively easy to solve. Having configured the script in a way that it should at least run, two of us were receiving a rather obscure error message about a file named config.php while the third group member's code was running perfectly. After some investigating, we realized that all three of us had our config.php files set with the Moodle database details for the group member whose code was working. After correcting these discrepancies by filling the config.php file with our own Moodle database details, our code ran without any issue, yet this wasn't the end of the hurdles we faced.

The other significant issue came with the resolution of our previous dilemma. Having personalized our config.php files, we realized that pushing our updates to our github repo and pulling them would write over each other's config.php files, thus necessitating us to perform the same fix from the previous issue each time we pull from our repos. To solve this, it was decided that we should include our config.php file in a .gitignore folder so that githubs will ignore the config.php file when we push to and pull from our repos. These were the biggest issues we faced over the course of this deliberation, and thankfully we seem to have a handle on them.

Full Testing Suite

Refactored parse_charset driver

break_up_long_words driver

parse_charset method signature in Moodle

break_up_long_words method signature in Moodle

function break_up_long_words(\$string, \$maxlength, \$scatchem="")

The Refactored Script

Error Insertion and Analysis

normalize_version w/ injection (line 252)

round w/ injection (line 603)

trusttext_strip w/ injection (line 1634)

break_up_long_words w/ injection (lines 1010-1012)

parsecharset w/ injection (lines 136-138)

Overall Experience

As a whole, this project was not what any of us expected. We all thought that working on an open source project and developing a program such as the one we wrote would be different than from how it actually was. Of course while we each had our own unique expectations, we were all still surprised by how even the simple things such as working in a team turned out to be different. Nonetheless it was still a very challenging and constructive endeavor, and it's safe to say that we all learned something.

I think one of the biggest misconceptions we had about working with an open source project is that we figured it would be, well, better. Moodle is obviously a large project with years of development behind it, something that in and of itself is impressive. And for the most part, Moodle not only functioned well, but appeared to be fairly polished in terms of documentation and project architecture. However, we were still surprised at how some small things within Moodle seemed wrong or misleading, such as how our round test method actually just truncated the input. Perhaps a more glaring mistake is that some methods lacked any significant documentation. Considering the size of the project, we also found this to be surprising.

One of the other surprising factors for us was the need for efficiency. We covered the refactor process fairly in depth previously in this report, but one small piece of that refactor process involved reworking some code that worked terribly inefficiently. While we actually never ran into this issue, we were aware that another project group with a similar strategy ran into an issue that caused their program to crash. Clearly this scared us enough to go and quickly solve the issue so that our program won't crash as a result of our being inefficient. In hindsight, this makes perfect sense, but it was a tad surprising to us that such a seemingly innocuous design choice could wreak havoc like that down the road.

One of the final surprises for us was learning how to work as a team. We pretty quickly divided up the roles and duties we would each have throughout the course of this project, but we weren't always good at sticking to them. Ultimately we managed to push out a product that we can all be proud of, but there were moments when the roles shifted because things needed to get done. None of this is to say that we clashed as a team or anything like that, in fact we remained largely disagreement free and never degenerated into any sort of argument. Still, the changing team dynamics and responsibilities on the fly and as we encountered problems was a surprise to each of us.

AOBTD TESTING FRAMEWORK

TEST CASE	TEST CASE ID	TEST CASE DESCRIPTION	TEST CASE STATUS	TEST CASE PRIORITY	TEST CASE SEVERITY
Test Case 1	TC001	Test Case 1 Description	Pass	High	Critical
Test Case 2	TC002	Test Case 2 Description	Fail	Medium	Major
Test Case 3	TC003	Test Case 3 Description	Pass	Low	Minor
Test Case 4	TC004	Test Case 4 Description	Pass	High	Critical
Test Case 5	TC005	Test Case 5 Description	Fail	Medium	Major
Test Case 6	TC006	Test Case 6 Description	Pass	Low	Minor
Test Case 7	TC007	Test Case 7 Description	Pass	High	Critical
Test Case 8	TC008	Test Case 8 Description	Fail	Medium	Major
Test Case 9	TC009	Test Case 9 Description	Pass	Low	Minor
Test Case 10	TC010	Test Case 10 Description	Pass	High	Critical