Project Name:	Moodle LMS
Product Name:	Moodle Testing Framework
Release Version:	Version 3.7.2

Final Paper

Created by: Alex Laughlin, Andrew Nesbett, and Chandler Long

Table of Contents

Introduction

The Goal……………………………………………….2

The Project ……………………………………………2

Chapters

Instructions cloning github……………………………3

Chapter One: Compiling Moodle……………………..4

Chapter Two: The Test Plan…………………………..6

Chapter Three: The Framework……………………...8

Chapter Four: The Framework Continued………..…12

Chapter Five: Fault Injection…………………………17

Chapter Six: Overall Experiences…………………....18

Conclusion

Self Evaluation………………………………………...19

Project Evaluation……………………………………..19

INTRODUCTION

The Goal:

The testing framework will run on Ubuntu (Linux/Unix). The testing framework will be invoked by a single script from within the top level folder using "./scripts/runAllTests.sh" and will access a folder of test case specifications, which will contain a single test case specification file for each test case. Each of these files will conform to a test case specification template. Each test case specification file thus contains the meta-data that the framework needs to setup and execute the test case and collect the results of the test case execution.

The Project:

Moodle is a free and open-source learning management system (LMS) written in PHP and distributed under the GNU General Public License. Moodle is used for blended learning, distance education, flipped classroom and other e-learning projects in schools, universities, workplaces and other sectors. Moodle was originally developed by Martin Dougiamas to help educators create online courses with a focus on interaction and collaborative construction of content, and it is in continual evolution. The first version of Moodle was released on 20 August 2002. Nowadays the Moodle Project is led and coordinated by Moodle HQ, an Australian company of 50 developers which is financially supported by a network of eighty-four Moodle Partner service companies worldwide. Moodle's development has also been assisted by the work of open-source programmers. Moodle as a learning platform can enhance existing learning environments. As an E-learning tool, Moodle has a wide range of standard and innovative features such as a calendar and a Gradebook. Moodle is a leading virtual learning environment and can be used in many types of environments such as education, training and development and in business settings.

Instructions

Cloning github repository

Getting Started

- To run the framework simple clone the project to your machine.

  ```
  git clone https://github.com/csci-362-01-2019/Home-School-Drop-Outs.git
  ```

- Navigate to the scripts directory located in the TestAutomation directory.

  ```
  cd ../pathToProject/TestAutomation/scripts
  ```

- Run the testing framework with the following command.

  ```
  bash ./runAllTests.sh
  ```

Chapter One:

Compiling Moodle

Project Setup:

Our first task was to get our project: Moodle, Up and running on localhost.

Thankfully moodle is very well documented, and we were able to find a step-by-step installation for moodle here.

Step 1: Install Ubuntu

Easy enough. We installed Ubuntu and we now have it running as a virtual machine. This allows us as a team to share the exact same distribution of Linux. Ultimately this will allow us to share files more easily when working on our moodle project.

Step 2: Install Apache/MySQL/PHP

Moodle is written using Apache, MySQL, and PHP. Apache is a web server software that will allow us to run moodle on our localhost. Or if we wanted to, we have the ability to run moodle on a server, however, for testing, we will just stick with the localhost. We need MySQL as a database language so that moodle can keep track of the information we send to the site like class information, users, etc. And finally, we need PHP installed. Moodle was written mostly using PHP.

Step 3: Install Additional Software

This part of the installation is installing mostly PHP related tools. We also install Git here if we haven't done so already. We use git in order to download the most recent version of the moodle project.

Step 4: Download Moodle

This is where we download moodle. We first start by finding the most recent update of the moodle project, which is the git branch: MOODLE_37_STABLE

Step 5: Copy local repository to /var/www/html/

This is the most critical part of the installation process. Basically what we did is copy the Moodle project and clone it on to our virtual machine. From there we give the machine the proper permissions to be able to write into the files once the server (or localhost) is running. Also, Since we setup a local repository in the previous step, Having your local repository outside of the webroot, we will be able to prepare and stage upgrades in a more efficient manner.

Step 6: Setup MySQL Server

In this step, we need to change some of the settings in our MySQL database. In the new version of Moodle we need to set the default storage engine to innodb and change the default file format to Barracuda, this is a new setting compared to previous versions. This caused us many problems and we ended up having to delete our entire Moodle project and start over since we had the wrong version of Moodle which did not use Barracuda or innodb. In this step, we also need to create the Moodle database and the Moodle MySQL User with the correct permissions. After setting up our user account with the correct permissions, our Moodle MySQL database was ready to go.

Step 7: Complete Setup

In this step, we were pretty much finished. All we needed to do was go to our localhost/moodle in a web browser and complete the initialization.

Chapter Two:

The Test Plan

Test Plan:

Test Execution

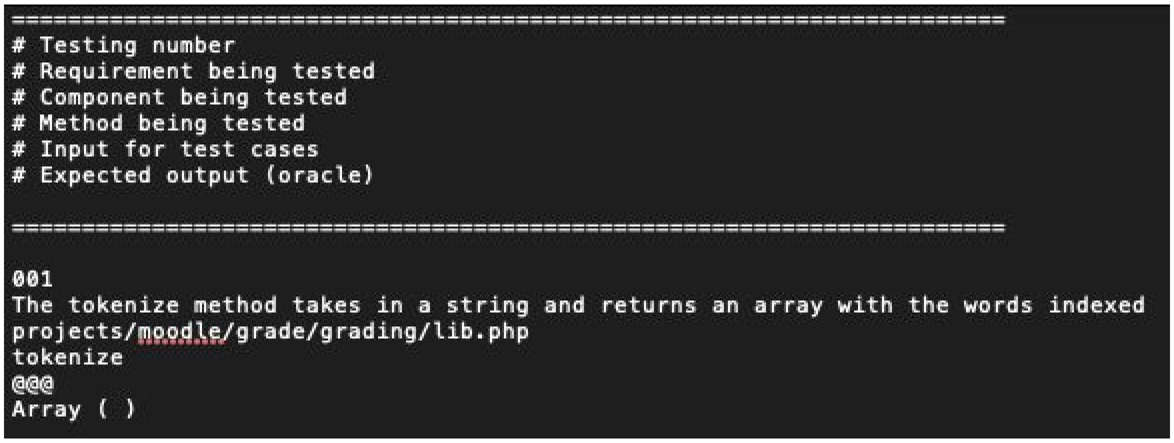
Test cases will be initialized and run with the bash script runAllTests.sh within the test automation folder.

../TestAutomation/scripts/runAllTests.sh

Running Tests

Tests are ran by the script by reading in all test cases from the testCase folder. Each test case is required to follow a specific format in which our script can parse through the .txt file and find the required information.

The following is an example of how the test cases should be formatted:



```
======================================================================
# Testing number
# Requirement being tested
# Component being tested
# Method being tested
# Input for test cases
# Expected output (oracle)

======================================================================

001
The tokenize method takes in a string and returns an array with the words indexed
projects/moodle/grade/grading/lib.php
tokenize
@@@
Array ( )
```

Tested Items

The items to be tested is the Tokenize method which is found within the Moodle project in the grade class.

Automated Testing Framework

1. Call a bash that loops through all our test cases files
2. The necessary data is aggregated and assigned to variables
3. The expected output is then written to the oracle
4. The bash script navigates to the driver and executes the method
6. The output of the method is written to a text file
7. The expected output and actual output (oracle) are compared by either passing or failing.

8. The information parsed from the .txt file and the results of the method are then formatted into a JSON file for later use in the web browser

Constraints

Constraints that affect the testing process: staff shortages, time allocation, deadlines

Staff shortages - due to our small team, testing all available methods in moodle would be impractical.

Time allocation - Due to the scope of the Moodle application, the test cases that the team will examine are limited.

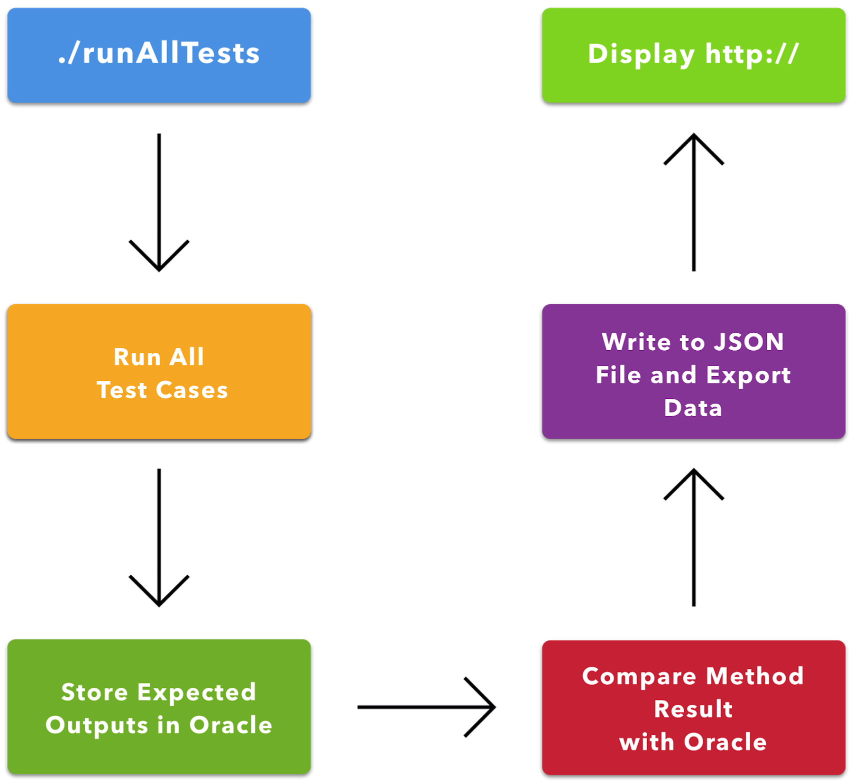
Deadlines - Since the team is only working on the project for a semester, limitations are imposed by the time allotted

Chapter Three:

The Framework

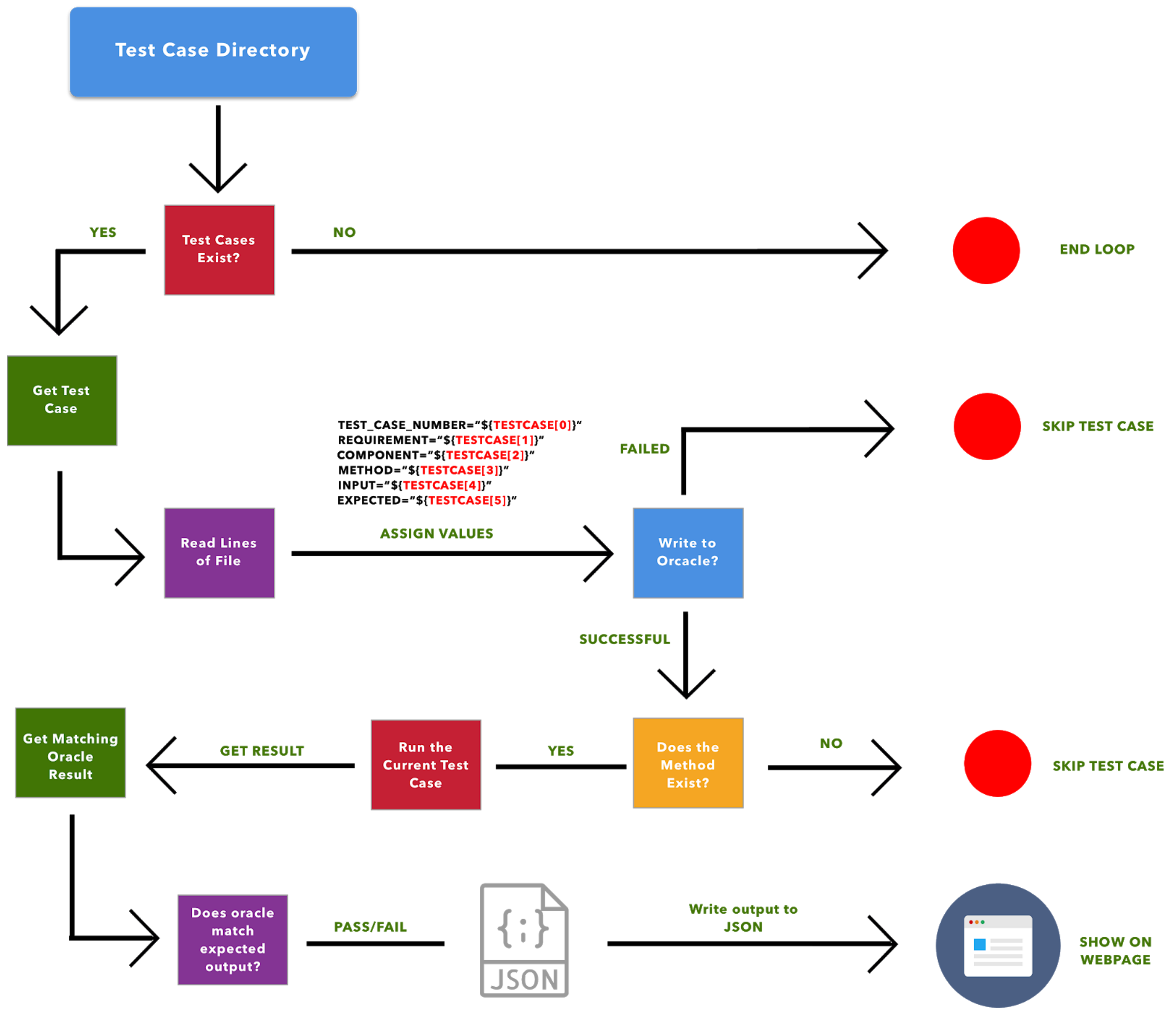
Test Recording Procedures:

In order to confirm that out tests have completed successfully, the tests will be writing to .txt files with their results. This will provide useful insight into what type of test cases provide valuable feedback. This will also allow us to confirm that Moodle software is working as intended.


./runAllTests
Run All Test Cases
Store Expected Outputs in Oracle
Compare Method Result with Oracle
Write to JSON File and Export Data
Display http://

Framework Broken Down

At the framework's highest level, it can be broken down into six steps. Start the script, Run all test cases, store expected outputs, compare to the output, write to JSON, display web page. At any step during the process, if what the script is looking for is not there, skip the test case. First, the runAllTests.sh bash script is ran. Once the script begins running, it looks for test cases in the test case folder. Once the file is found, it is parsed and broken up into an array of values, this is why the format of the test case is important. The expected output (specified in the test case) is written into an oracle file to test for correctness later on. Once all values are parsed and the script determines that it is a valid test case, it then determines whether or not the method to test is valid. If it is a valid test, the script then runs the method with the input from the test case and checks the output with the oracle to test whether or not the output matches the oracle. The values from the array are then stored in a JSON formatted file and then the script restarts until it iterates through the entire test case folder. Once completed, the JSON file is parsed and sent into an HTML document to output all results into a table.

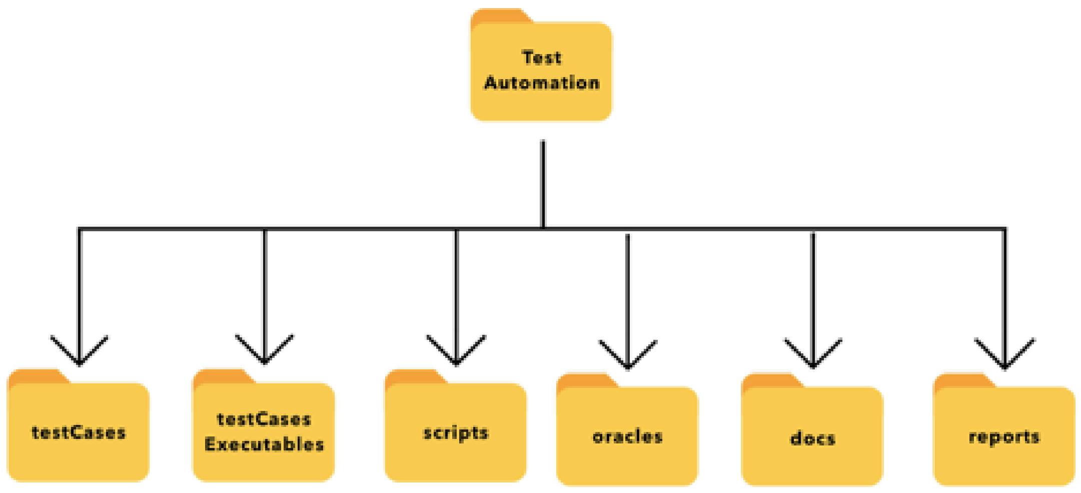


```
TEST_CASE_NUMBER="${TESTCASE[0]}"
REQUIREMENT="${TESTCASE[1]}"
COMPONENT="${TESTCASE[2]}"
METHOD="${TESTCASE[3]}"
INPUT="${TESTCASE[4]}"
EXPECTED="${TESTCASE[5]}"
```

FILE STRUCTURE:

Framework Directories:

Directories inside of the testing framework are structures in such a way that allows the framework to reach into certain directory and extract information. The framework as also capable of storing files so that can be referenced later.



/testCases:

The “testCases” directory houses all the tests cases that can be run in the framework

/testCasesExecutables:

The “testCasesExecutables” directory houses all the methods that are extracted from the Moodle directory. Files inside of “testCasesExecutables” are initialized when there are new methods that need to be tested. These methods are then instantiated in the driver.

/scripts:

The “scripts” directory houses all the scripts needed to properly run the testing framework. This is where ./runAllTests.sh is stored which runs all test cases in the framework. Driver.php is also located in this directory.

Driver.php initializes all the classes that are need to properly run Moodle methods and calls upon them.

/oracles:

The “oracles” directory houses all the expected results that are obtained from the testCase.txt files.

/docs:

The “docs” directory houses the README.rm file.

/reports:

The “reports” directory houses the generated HTML file which is produced by the testing framework.

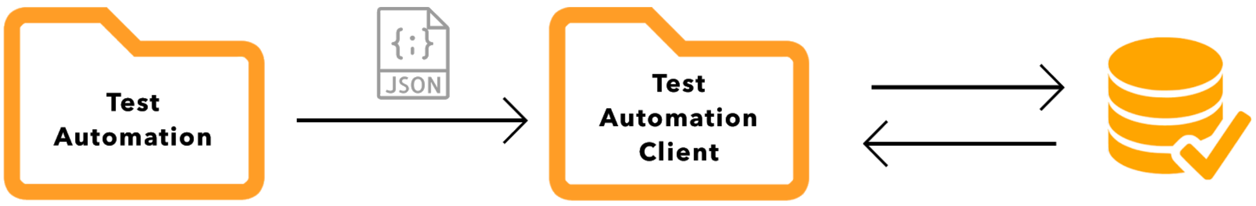
Chapter Four:

The Framework Implementation

FILE STRUCTURE:

Framework Directories:

At the root of our testing framework directly a second folder has been added called TestAutomationClient. TestAutomationClient houses our client web application which displays the results generated from our TestAutomation directory. Once stored in the tst automation directory the test cases are then housed again in a MySQL database in order to reduce the load of the testing framework if we were to test one hundred or even thousands of cases.

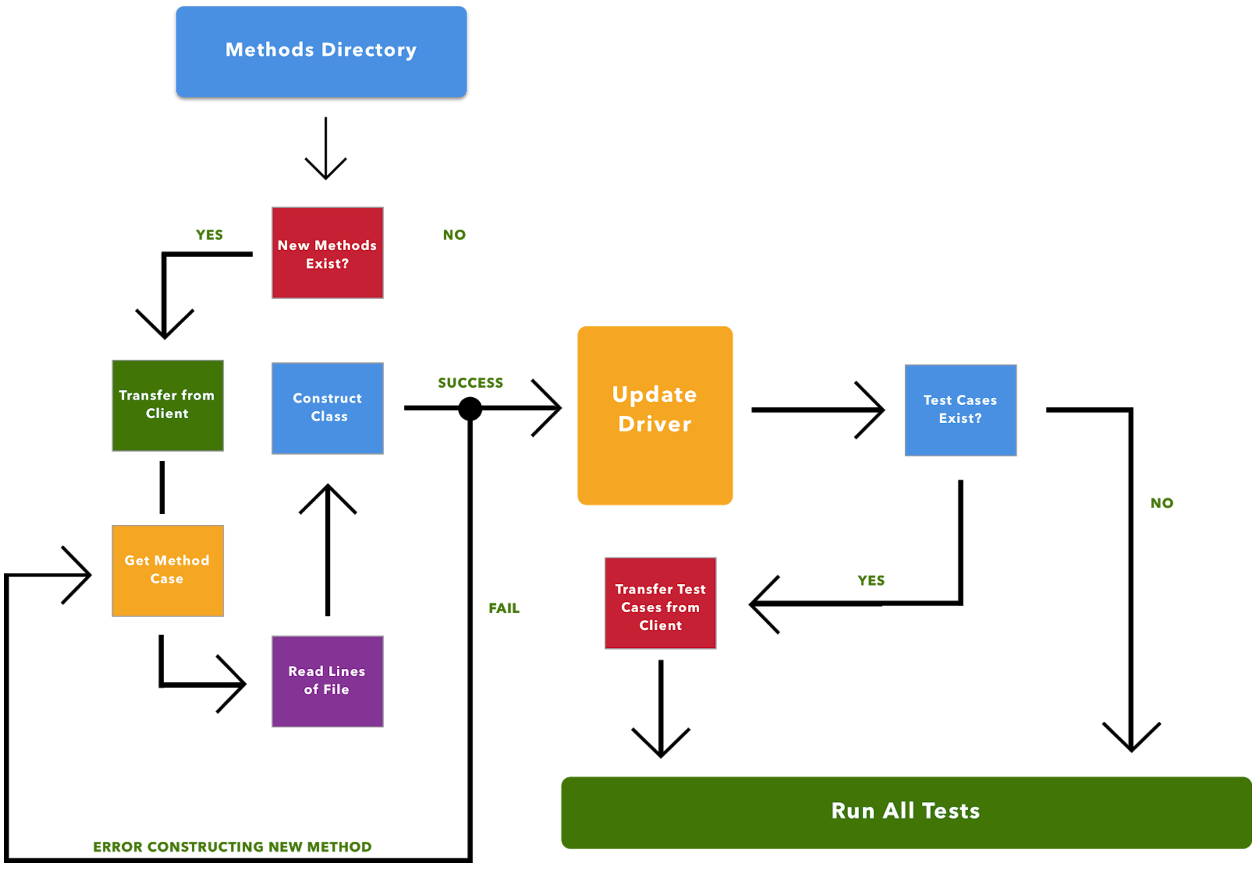


ARCHITECTURE

Overview:

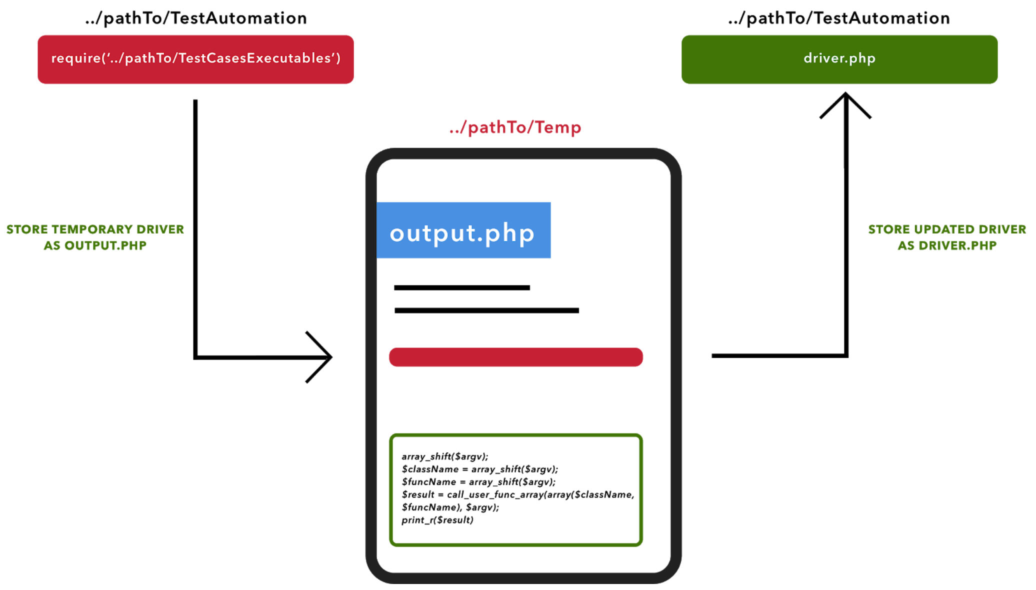
The architecture of the automated testing framework has changed since that of deliverable 3. In deliverable 3 we expressed that were were in need of a way to instantiate our classes in order to properly test the majority of classes available in the massive project that is Moodle. To provide users with a more flexible and interactive experience, the framework has been modified not only to accept new test cases, but also new moodle methods. The architecture which runs all the test cases remains the same. What does change is the process before running all the tests. Before running all of our tests, our bash scripts checks the client “executables” directory to see if any new methods or test cases have been added by a user. If new methods were added, we cycle through them and attempt to construct a new method. To construct a new method, we first look through the component (given to us by the method case) to find the name of the method we wish to extract. From there, we grab the entire line where the method is location and all lines following up until the functions end.

If the method is constructed correctly then we need to update the driver to account for the new method. The framework has a flexible driver than can be expanded upon to account for any new methods the testing framework gets.



The Flexible Driver:

We are able to add new methods into our testing framework due to the implementation of a expandable/flexible driver. What this means is that if a new method is added then the driver will be updated to account for the new method that it received.


../pathTo/TestAutomation
require('../pathTo/TestCasesExecutables')
../pathTo/TestAutomation
driver.php
../pathTo/Temp
STORE TEMPORARY DRIVER AS OUTPUT.PHP
output.php
STORE UPDATED DRIVER AS DRIVER.PHP

```
array_shift($argv);
$className = array_shift($argv);
$funcName = array_shift($argv);
$result = call_user_func_array(array($className,
$funcName), $argv);
print_r($result)
```

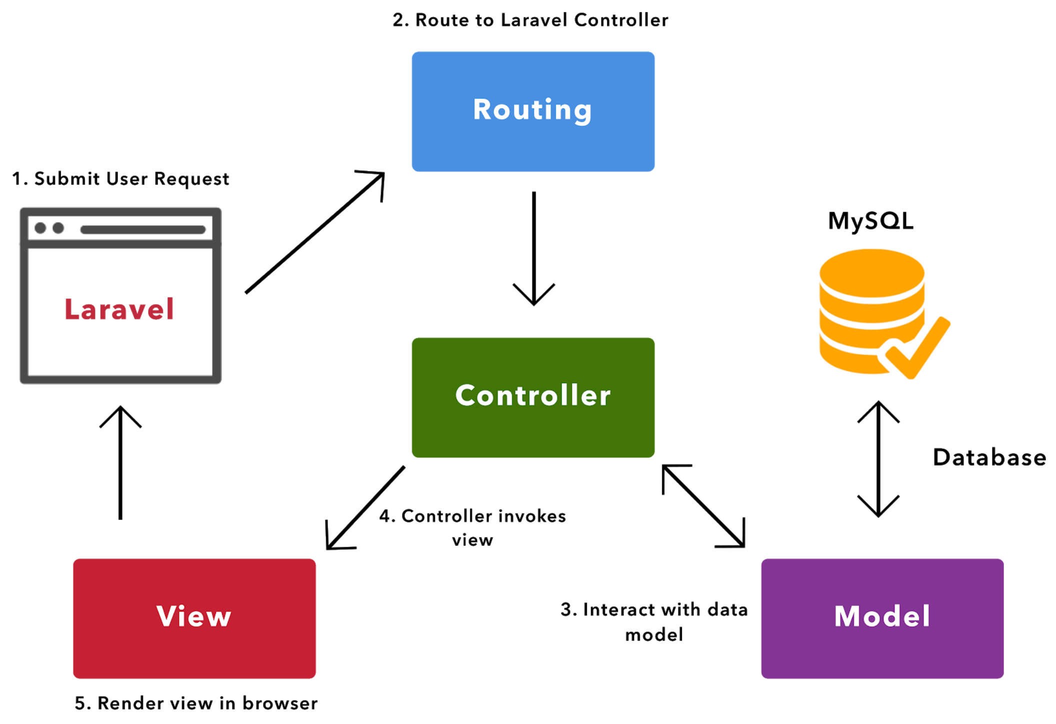
The process of adding new methods to the driver starts with the creation of the require statement. This require statement is the same name of the method we are testing, it is added using a “sed” command directly after the last require statement in the file. The function that follows is a simple php method that calls executes the method. The input accepts an array of arguments which makes this flexible enough to account for a functions with any given amount of parameters. Once the require statement is added, the updated version of the “driver.php” file is stored in the “TestCasesExecutables” directory. From there we have a new updated driver that has accounted for any new methods that have been added.

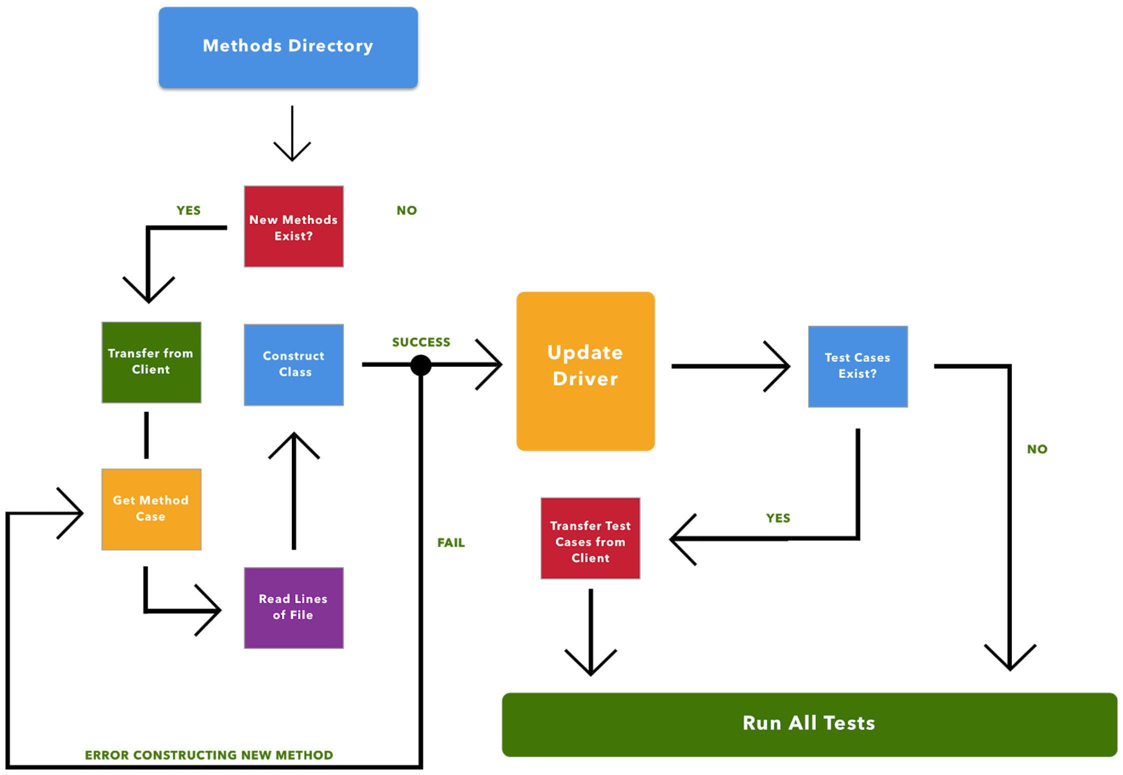
Laravel:

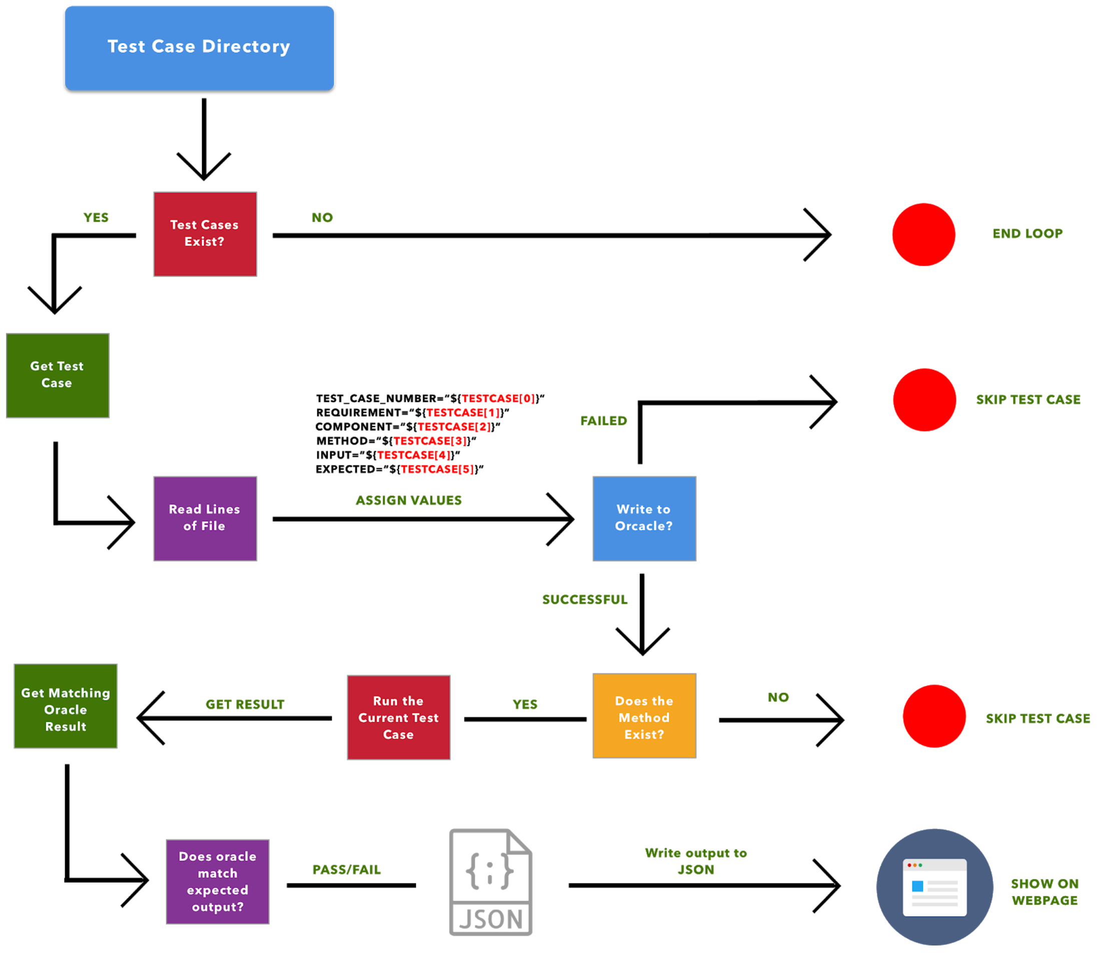
Unlike in the previous chapter, we are no longer just opening the json file in a basic HTML, PHP, and CSS stack. Instead, in parallel with moodle we will be using a PHP Web Application Framework called Laravel. Laravel is a free, open-source PHP web framework, created by Taylor Otwell. The goal of Laravel is to aid the development of web applications following the model view controller (MVC) architectural pattern. Some of the features of Laravel are a modular packaging system with a dedicated dependency manager, and different ways for accessing relational databases. Utilizing the power behind this framework also gives us the opportunity to implement a more robust client side to the already robust testing framework. User authentication, and the ability to create test cases and methods directly from the client side are two of the main features implemented through the Laravel framework.

The Model View Controller Architecture:

The Model View Controller (usually known as MVC) is a software design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements. The model represents the data, and does nothing else. The model doesn't depend on the controller or the view. The view displays the model data, and sends user actions (e.g. button clicks) to the controller.


2. Route to Laravel Controller
Routing
1. Submit User Request
Laravel
MySQL
Controller
Database
4. Controller invokes view
View
3. Interact with data model
Model
5. Render view in browser





```
TEST_CASE_NUMBER="${TESTCASE[0]}"
REQUIREMENT="${TESTCASE[1]}"
COMPONENT="${TESTCASE[2]}"
METHOD="${TESTCASE[3]}"
INPUT="${TESTCASE[4]}"
EXPECTED="${TESTCASE[5]}"
```

Chapter Five:

Fault Injection

Below is a list of faults that would cause the script to fail:

No test cases

Since our script uses the test case for specify the method to test, if the test case directory is empty the script will not run and exit the loop since there is no test cases to run and there is no method to be determined to test.

No methods

In the test case, if there is no method specified in the test case, then the script will be unable to determine what method to test. This could cause a problem if there is only one test case. If there is only one test case in the folder and no method is specified in the test case then this will cause a fault in the script since there is no method to test.

Wrong test case format

There are multiple ways that this could cause a fault. The test case is required to follow a very specific format. If this format is broken or it does not follow the format correctly then this could cause a fault. One possibility is that the oracle can be formatted incorrectly in which it does not match to the output that results from the tokenize method. This can cause the oracle to fail even though you gave it the correct values.

Wrong method format

If the method is not constructed properly then this can cause a fault in the script in which it will fail.

If the test case input starts with

Before test cases are stored, they must undergo a few checks that make sure the values that are being obtained are the ones needed to execute the method. In order to properly label these values, notes are included at the top of every test case which start with “#”. When the text file is being processes, lines that start with the character “#” are ignored. A major fault in our testing framework is exposed here, if a test case input values happens to be the character “#”, the line will be ignored resulting in a faulty test case.

Chapter Six:

Experiences and Lessons Learned

Experiences:

Working with Moodle initially proved frustrating due to its size and complexity. Installing the software and it's necessary dependencies took longer than anticipated due to unforeseen errors. Once installed and properly set up, we needed to find testable methods. Many of the methods rely on objects instantiated during normal use such as grade, student, class, etc. Luckily the method we found, tokenize, does not depend on any built objects found within the running Moodle project. Finding this method took up a lot of the time in our project, due to the fact that almost every method or function depends on other objects already created inside Moodle. Once this method was found, it was easy for us to start creating our bash script and create the necessary drivers to finish the automated test.

What we learned:

- Became familiar with Linux OS
- Simple Bash scripting
- Setting up a local server
- HTML formatting
- Automated testing
- The PHP language
- MySQL commands
- Apache web hosting

Self Evaluation

Our team worked well together and our communication was effective. Due to our conflicting schedules, we were not able to meet as often as we initially desired but managed to complete large portions of the project remotely. Github was a great help in managing our project and working independently when necessary. At times we seemed to be running behind the schedule we forecasted, but we managed to complete the last few phases faster than we had planned. Initially only one of our team members had ever used bash and no one in the group had ever used PHP, so this took a toll on trying to create the bash script and the necessary drivers. Luckily, we were all able to be brought up to speed quickly to match each others knowledge of the bash scripting language. We all had to learn PHP on our own however, since none of us knew how the language worked, but as we learned, PHP is a very loose language and it was rather simple to understand how it runs. Once we were able to overcome this obstacle, the project began to come together smoothly with not too many bumps. All things considered, our team worked together very well and we were able to set tasks and get them completed.

Project Evaluation

Our team enjoyed this project more than we had anticipated largely due to the autonomy granted by the assignment. It was interesting researching open source software and their respective communities. Seeing “what’s under the hood” on such a large project provided a great deal of insight on how to structure a large, complex system. Designing tests to ensure full coverage of a method also taught us a great deal about what makes a good test case. To improve this project, consider adding some time for instruction on how to choose an open source software that will be viable and interesting for this project.