

New Leaf Term Project

Final Report

Luke McGuire

Chris Taylor

Kasper Dugaw

College of Charleston


December 3, 2020

Table of Contents

Section	Page
Introduction	3
Chapter 1: Moodle	4
Introduction	4
Building Moodle and Configuring the Testing Environment	4
Chapter 2: The Test Plan	7
Introduction	7
The Test Plan	7
Test case specifications template:	10
Defining the First Five Test Cases	11
Chapter 3: Designing the Framework	12
Introduction	12
Architecture of the Framework	12
Usage	14
Running All Tests	14
Creating Drivers and Test Cases	14
The EvalMath evaluate Driver	15
Chapter 4: Developing Test Cases	16
Introduction	16
Completing the Testing Framework	16
Drivers	18
Chapter 5: Injecting Faults and Testing the Framework	20
Introduction	20
Designing Faults	20
Injecting Faults	22
Testing the Framework	23
Chapter 6: Conclusions and Overall Experiences	26
Team Self Evaluation	27
Assignment Evaluation	27

Introduction

In this report we will present our team term project, an automated testing framework built for Moodle, a popular open source learning management system. This report consists of six chapters that were written throughout the course of this project that detail the design and implementation process of our testing framework at various stages of development as well as the final product. At the end of this report we will include two sections evaluating the work our team completed throughout this project and an evaluation of the assignment itself. All of our team's work can be found on our GitHub repository which contains all of our framework's source code, a project wiki where we logged the timeline of our project, additional documentation for our framework, as well as a poster and presentation about our project.



NewLeaf Automated Testing Framework

Luke McGuire, Chris Taylor, Kasper Dugaw
College of Charleston Department of Computer Science

Introduction

In this project our team designed, implemented, and tested an automated PHP testing framework and several test cases for Moodle, an open source learning management system which we found lacked tests for external libraries included in their source code.

Conclusions

Over the course of this project our team learned quite a bit about the software engineering process, including project planning and management, designing and producing good tests, and how to effectively work as a team on a software project.

We were also pleased with how our testing framework itself ended up turning out. Although we had some setbacks, we were not only able to design and implement our vision, but expand upon it. After completing the basic functionality of our framework earlier than expected we were able to spend time improving or project by adding features including improved output generation, improved error handling, and scripts to automate the process of fault injection. Additionally, we were also glad we were able to produce something we felt was valuable by writing our tests for Moodle libraries that did not have pre-existing tests.

To conclude, if it's not already clear, we were very satisfied with the results of our project and our development as a team and as software engineers.

Acknowledgments


Thanks to the Moodle project and its contributors:
<https://moodle.org/>

Thanks to our instructor and faculty advisor:
Dr. Jim Bowring

Additional Information

Our GitHub repository, which contains the entirety of our project including source code and our final report can be found at the link below.

<https://github.com/csc1-362-02-2020-New-Leaf>



Framework Design

Our framework consists of three main components, the controller, test cases, and drivers.

The controller is responsible for parsing and executing test cases as well as generating the results into a report.

The test cases are a set of JSON files which specify individual tests and provide the controller with the data necessary to execute them including the driver that should be used to run the test, the input that should be given to the functionality being tested, and the expected output of the provided input.


Finally, the drivers are responsible for interfacing with the Moodle code and providing the controller with the ability to test a particular method or class and returning the results of a test back to the controller.

Testing and Results

To validate our testing framework we designed a system to inject faults into the Moodle source code and executed our framework before and after injecting the faults. The results of this procedure can be seen in Figures 1 and 2.

Implementing Tests

After implementing our framework we developed 30 tests for the Moodle project for 6 functions found in the project's libraries. The modular design of our framework provides a simple framework for creating new drivers and tests, and examples of these can be seen in Figures 3 and 4 respectively.



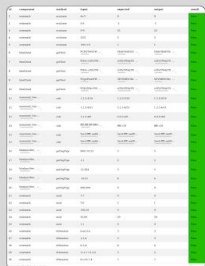


Figure 1: Test results prior to fault injection.

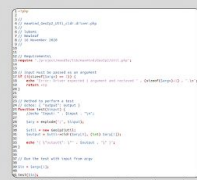


Figure 2: Test results post fault injection.

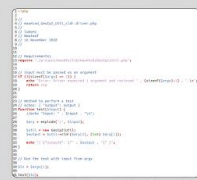


Figure 3: One of the drivers written for our framework.




Figure 4: One of the test cases written for our framework.

Chapter 1: Moodle

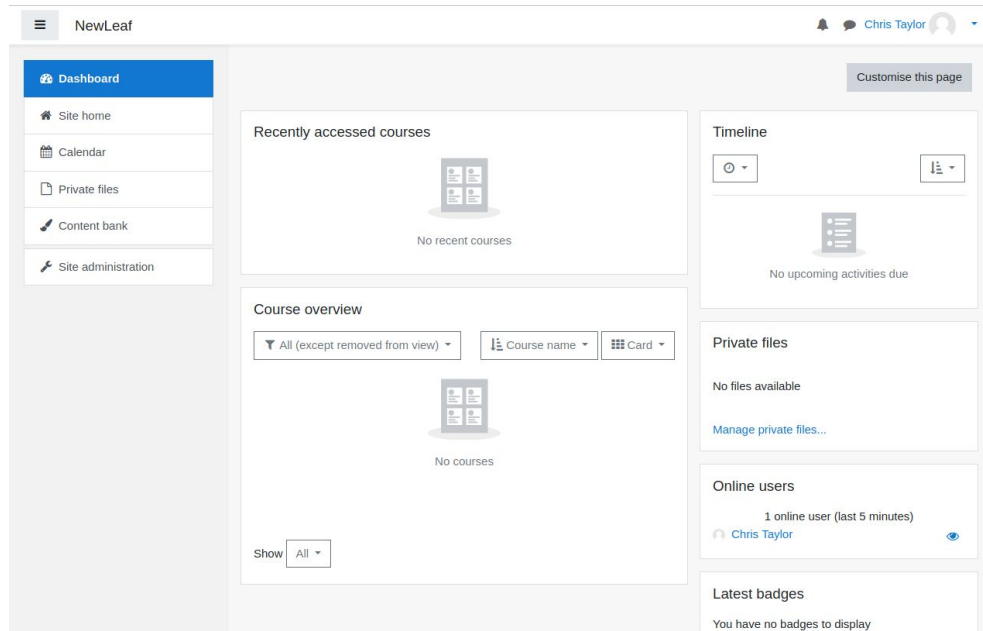
Introduction

In this chapter we will describe the project we have chosen to build our testing framework for, Moodle. Moodle is a popular and free learning management system (LMS) designed to provide educators with a “single robust, secure and integrated system” to create personalized learning environments. It is an active project written primarily in PHP. We chose Moodle because its mission of providing a free and robust LMS resonated with us, especially as online learning has become increasingly important in the last few months due to COVID-19, as well as because of its popularity and its extensive documentation for both users and contributors.



Building Moodle and Configuring the Testing Environment

This section will describe the methodology we used to build Moodle and initialize the testing environment; more detailed descriptions of this process can be found on our team project wiki. First, we built Moodle by cloning the Moodle repository and following Moodle’s installation guide, which involved the installation and configuration of PHP, MySQL, and an Apache web server. After completing the installation process, we succeeded in getting Moodle up and running on our virtual machine.



The Moodle web UI running on our VM.

Next, we initialized Moodle's testing environment by installing PHPUnit through the PHP Dependency Manager Composer and configuring it to work with Moodle's pre-existing unit tests. After this process was complete, we were able to run all tests with the command:

```
vendor/bin/phpunit
```

```
Moodle 3.9.2 (Build: 20200914), ccd4ef8ddd03d98b84e3231866b8b1e024dab1db
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.
```

.....	59 / 15908 (0%)
.....ES.....	118 / 15908 (0%)
.....F.....	177 / 15908 (1%)
.....SSSSSSSS.....	236 / 15908 (1%)
.....	295 / 15908 (1%)
.....	354 / 15908 (2%)
.....F.....	413 / 15908 (2%)
.....	472 / 15908 (2%)
.....	531 / 15908 (3%)
.....	590 / 15908 (3%)
.....	649 / 15908 (4%)
.....	708 / 15908 (4%)
.....	767 / 15908 (4%)
.....	826 / 15908 (5%)
.....	885 / 15908 (5%)
.....	944 / 15908 (5%)
.....	1003 / 15908 (6%)
.....	1062 / 15908 (6%)
.....	1121 / 15908 (7%)
.....	1180 / 15908 (7%)

A sample of the results produced by PHPUnit.

This output demonstrates that the test environment is working properly and gives us some interesting insight into the Moodle testing environment, for example the results show that Moodle has nearly 16000 PHP unit tests. At this point we began to consider what we could do with our project that would actually prove valuable given the extensive pre-existing testing setup, and we noticed something interesting. Moodle includes a series of third party modules used for math, IP based geolocation, HTML parsing, and several other things in its source code, and none of these modules contained pre-existing tests. With this discovery we realized that we could write our tests for these modules and manage to provide the value we were looking for in our project.

Chapter 2: The Test Plan

Introduction

In this chapter of our term project, we will discuss our test plan and the methodology we will use to determine and implement test cases, as well as to develop the testing framework that will automate these tests.

What We Are Testing

As discussed in Chapter 1, we will be writing our tests for some of the external libraries which Moodle has incorporated into its source code. For our first five test cases, we focused upon a central math library called EvalMath which evaluates string mathematical expressions, for example to provide automatic math question grading within the LMS. We will be testing the evaluate method from this library for both edge cases and normal use cases, all of which will be detailed in the following sections. Additionally, for the remainder of our test cases we will identify other testable methods and functionality included in the libraries we have been focusing upon.

The Test Plan

The Testing Process

We will be testing several classes that are contained within external modules which have been included in Moodle's core library. To do this, we will create PHP drivers that will accept inputs and write the results to stdout for each method we intend to test. Our main script will control all of the drivers by reading in the JSON test cases and using their information to run the specified

driver and inputs. The results will be appended to the end of an HTML report that will then be displayed in the default browser.

Requirements Traceability

(Test cases 1-5): evalMath_evaluate.driver.php

Requires a mathematical string expression and returns the result.

(Test cases 6-10): html2text_getText.driver.php

Requires a string of HTML as input and returns the text found inside of HTML tags. Here we convert the input and output to base64 to prevent the HTML being tested from interacting with the HTML of the generated reports.

(Test cases 11-15): maxmind_GeoIp2_Util_cidr.driver.php

Requires a string internet protocol address (IP) and network mask as input and returns the IP in Classless Inter Domain Routing (CIDR) notation.

(Test cases 16-20): htmlpurifier_UnitConverter_getSigFigs.driver.php

Requires an integer or float value and returns the number of significant digits in the input.

(Test cases 21-25): evalMath_mod.driver.php

Requires two integer inputs separated by a comma; the first is the modular dividend and the second is the modulus (i.e. an input of 7,2 will compute $7\%2$). The output is the computed solution to the modulus as an integer.

(Test cases 26-30): evalMath_ifthenelse.driver.php

Requires three string arguments and evaluates the first argument, the conditional, as PHP to see

if the result is true or false. If it is true, the returned value is the second argument and if it is false, the returned value is the third argument.

Test recording procedures

Tests will be accessed one-by-one and results will be appended to an HTML file. The HTML file will then be automatically opened in the default web browser to display the results.

Machine and software requirements

Operating system: Tested on Ubuntu versions 20.04 and 20.10, any Debian based Linux should work without modification, other platforms will require a modified setup process.

Software required: PHP, Python, and the Moodle source code.

Constraints

Limited team size and limited time to complete different stages of the project.

System tests

The system tests will check that the tests run properly and provide the expected results in a clean system, implying no reliance on particular file paths or components from individual machines.

Test case specifications template:

Fields:

1. Test number or ID
2. Driver
3. Requirement being tested
4. Component being tested
5. Method being tested
6. Test input(s)
7. Expected outcome(s)

JSON File:

```
{  
  "id": integer,  
  "driver": "driver"  
  "requirement": "Requirement",  
  "component": "Component Name",  
  "method": "Method Name",  
  "input": "input",  
  "output": "output"  
}
```

Defining the First Five Test Cases

Test cases 1 through 5 have been defined and can be found in our TestCases GitHub directory.

As discussed, these first 5 test cases focus upon testing the functionality of the evalMath function in Moodle's math library. For example, in one of the test cases (shown below) we check the functionality of the addition subroutine by defining "4 + 5" as input, with the expected output being 9. We designed our tests to cover a variety of scenarios including uses of various operators and numeral types.

```
1 {  
2     "id": 1,  
3     "driver": "evalmath_evaluate.driver.php",  
4     "requirement": "basic addition functionality",  
5     "component": "evalmath",  
6     "method": "evaluate",  
7     "input": "4+5",  
8     "expected": 9  
9 }
```

Test case 1 definition file.

Chapter 3: Designing the Framework

Introduction

This chapter will discuss in detail the structure and implementation of our testing framework, the framework's use cases and workflow, and an initial subset of the eventual 25 test cases we will construct for this project. These discussions will include details on our thought process and design decisions, as well as considerations of where this project will develop from here and what our next steps will be.

Architecture of the Framework

The structure of our testing framework is divided into three distinct parts: the controller, the drivers, and the test cases themselves. The controller, `runAllTests.py`, is responsible for parsing and executing test cases while aggregating the results and presenting them in a generated report in a web browser. The test cases are a set of JSON files which specify individual tests and provide the controller with the data necessary to execute them, including the driver that should be used to run the test, the input that should be given to the functionality being tested, and the expected output of the provided input. Finally, the drivers are responsible for interfacing with the Moodle code and allowing the controller to test a particular method or class, as well as returning the results of a test back to the controller. The following three subsections will provide specific implementation details about these components of our testing framework.

Controller

The controller is defined in `runAllTests.py`. As stated above, this script is responsible for executing all of the test cases and aggregating the results. The controller is written in Python; this may seem to be an odd choice, since much of the remainder of our project and Moodle itself are

written in PHP, but as the responsibility of interfacing with Moodle lies with the drivers we were able to use a language we preferred for the controller. The script accomplishes three main tasks: loading the test cases, executing the tests, and displaying the results in a web browser after embedding the results in HTML.

<p>Test Case: 1 evalmath->evaluate</p> <p>Input: 4+5 Expected Output: 9 Result: 9 -> Pass</p>
<p>Test Case: 2 evalmath->evaluate</p> <p>Input: 5-6 Expected Output: -1 Result: -1 -> Pass</p>
<p>Test Case: 3 evalmath->evaluate</p> <p>Input: 5*5 Expected Output: 25 Result: 25 -> Pass</p>
<p>Test Case: 4 evalmath->evaluate</p> <p>Input: 25/5 Expected Output: 5 Result: 5 -> Pass</p>
<p>Test Case: 5 evalmath->evaluate</p> <p>Input: 100>10 Expected Output: 1 Result: 1 -> Pass</p>
<p>Test Case: 6 html2text->getText</p> <p>Input: PCFET0NUWVBFIGh0bWw+PGh0bWw+PGjvZHK+PGgxPIRoZSBjZWFlaW5nPC9oMT48cD5BIHhcmFncmFwaC48L3A+PC9ub2R5PjwvaHR0bD4= Expected Output: VEHFIehFQURJtKcKcEgcGFyYWdyYXBoLgo= Result: VEHFIehFOURJtKcKcEgcGFyYWdyYXBoLgo= -> Pass</p>

The first implementation of our output generation.

Drivers

Since Moodle is primarily PHP, we decided to build our drivers using PHP so they can easily interface with the class we are attempting to test. The drivers ‘require’ the PHP file being tested then implement a test by instantiating an object or executing a method with input provided to the driver by the controller. The controller uses the drivers to pass in the test inputs to the class/method that the driver controls and then the driver returns the results to the controller.

Usage

Due to the modularity of our testing framework, adding test cases and drivers to the framework is a relatively straightforward process. The controller, `runAllTests.py`, automatically finds all test case definitions in the `testCases` directory and will run them so long as the driver specified in the test case is present in the `testCaseExecutables` directory. This makes adding new test cases to the framework as simple as dropping the JSON definitions into the `testCases` directory.

Running All Tests

To execute all of the tests defined in the `testCases` directory the controller can be executed with the following command: `./scripts/runAllTests.py`

Creating Drivers and Test Cases

Test cases can be created easily following the template defined in Chapter 2 of this project.

Drivers, on the other hand, are a bit more complex as they are responsible for directly interacting with the Moodle source code, as well as providing an interface for the controller to pass and receive data from the functionality being tested. Fortunately, drivers are also pretty flexible within our framework, and there are only two major requirements to keep in mind when implementing one:

1. Each driver must accept input to pass to the method being tested within Moodle as an argument.
2. Each driver must return the output generated by the Moodle source code to stdout as a JSON string in the format “{ "output": output }”.

Otherwise, just keep the test case specifications in mind, load in the necessary modules from Moodle, and implement the desired test.

The EvalMath evaluate Driver

The evaluate driver is the first driver we implemented for our project. It was written for the evaluate method inside of the EvalMath class. The method takes in a mathematical expression as a string, parses it, then calculates and returns the solution to the expression. The driver takes a string as input via an argument, instantiates the EvalMath class, executes the method with the provided input, and returns the results to stdout. The controller is then able to check to see if the returned value is the same as the expected value and either pass or fail the test.

```

1 <?php
2
3 //
4 // evalmath_evaluate.driver.php
5 //
6 // lukem1, chris-m-taylor
7 // Newleaf
8 // 19 October 2020
9 //
10
11
12 // Requirements
13 // Note: Not part of moodle_internal, so no need to load moodle config
14 require "../project/moodle/lib/evalmath/evalmath.class.php";
15
16
17 // Input must be passed as an argument
18 if (!(sizeof($argv) == 2)) {
19     echo "Error: Driver expected 1 argument and recieved " . (sizeof($argv)-1) . ".\n";
20     return -1;
21 }
22
23
24 // Method to perform a test
25 // Echos: { "output": output}
26 function test($input) {
27     //echo "Input: " . $input . "\n";
28
29     $math = new EvalMath;
30     $output = $math->evaluate($input);
31
32     echo "{ \"output\": " . $output . " }";
33 }
34
35
36 // Run the test with command line input
37
38 $in = $argv[1];
39
40 test($in);

```

The evalmath_evaluate driver.

Chapter 4: Developing Test Cases

Introduction

During this chapter of our project, we completed our testing framework, added functionality and styling to our generated reports, and completed our 25 test cases.

Completing the Testing Framework

Since the last chapter, we have almost entirely rewritten our `runAllTests` script and have made significant improvements. For one, our controller now performs quite a bit of error handling to handle cases including malformed test case definitions, missing drivers, drivers with errors or improper return values, and exceptions within the code being tested. Additionally, we have redesigned the controller such that only one test case is loaded at a time and that once the output is determined, it is written directly to the output file. We have also made substantial changes to output generation, which now includes a table which supports a sort option as well as scrollable components for items that are too large to fit within their table cells. The pass and fail colors are more clearly defined, and we also have more fields describing the tests. A time stamp also appears in the upper left corner of the report to show when the tests were run. The images below show the progress we have made between chapters 3 and 4 of our project.

Old:

<div>Test Case: 1 evalmath->evaluate</div> <div>Input: 4+5</div> <div>Expected Output: 9</div> <div>Result: 9 -> Pass</div>
<div>Test Case: 2 evalmath->evaluate</div> <div>Input: 5-6</div> <div>Expected Output: -1</div> <div>Result: -1 -> Pass</div>
<div>Test Case: 3 evalmath->evaluate</div> <div>Input: 5*5</div> <div>Expected Output: 25</div> <div>Result: 25 -> Pass</div>
<div>Test Case: 4 evalmath->evaluate</div> <div>Input: 25/5</div> <div>Expected Output: 5</div> <div>Result: 5 -> Pass</div>
<div>Test Case: 5 evalmath->evaluate</div> <div>Input: 100>10</div> <div>Expected Output: 1</div> <div>Result: 1 -> Pass</div>
<div>Test Case: 6 html2text->getText</div> <div>Input: PCFET0NUWVBFtGbdWw+PGb0Ww+PGjvZk+PGxpPmhZSRZWPk+WnPCNmT4k-cD5B0BhmPmcnPwaC4B.3A+PC9Bd2R3PyeaU8RbD4=</div> <div>Expected Output: VEhFIEhFQUjTckKGEg-GFyYWdyYXB=.ge=</div> <div>Result: VEhFIEhFQUjTckKGEgGFyYWdyYXB.ge=-> Pass</div>

New:

NewLeaf Testing Framework Results

2020-11-17 11:41:00.388002

id	component	method	input	expected	output	result
-4	evalmath	evaluate	4+5	-4	9	Fail
1	evalmath	evaluate	4+5	9	9	Pass
2	evalmath	evaluate	5-6	-1	-1	Pass
3	evalmath	evaluate	5*5	25	25	Pass
4	evalmath	evaluate	25/5	5	5	Pass
5	evalmath	evaluate	100>10	1	1	Pass
6	html2text	getText	PCFET0NUW ...	VEhFIEhFQU ...	VEhFIEhFQU ...	Pass
7	html2text	getText	PHA+cGFyYW ...	cGFyYWdyYXB...	cGFyYWdyYXB...	Pass
8	html2text	getText	PHA+cGFyYW ...	cGFyYWdyYXB...	cGFyYWdyYXB...	Pass
9	html2text	getText	PGgxPmhlyW ...	SEVBRElORzE...	SEVBRElORzE...	Pass
10	html2text	getText	PGjvZHK+PH ...	cGFyYWdyYXB...	cGFyYWdyYXB...	Pass
11	maxmind_Geol...	cidr	1.2.3.4/16	1.2.0.0/16	1.2.0.0/16	Pass
12	maxmind_Geol...	cidr	1.2.3.4/31	1.2.3.4/31	1.2.3.4/31	Pass

Test Cases

At the time of writing this chapter our team has completed 5/5 test drivers and 25/25 test cases.

The drivers that we have implemented will be discussed in the following subsections.

Drivers

evalMath_evaluate.driver.php

In the evaluate method inside of the evalMath class, we are testing different mathematical operations that are input as a string. The method takes in a mathematical expression in the form of a string, parses it, calculates, and returns the answer the method comes up with.

evalMath_mod.driver.php

The modulo evalMath driver evaluates the modulo of the two numbers it is given. For example, if the user passes in 7,7 the output should be 0 since $7 \% 7$ is 0. Modulo is calculated by getting the remainder of dividing the first input by the second. If the second number is larger than the first, the remainder will always be the first number since it is not divisible by the second number. It is impossible to take any number $\% 0$, so if attempted the method returns an undefined value.

html2text_getText.driver.php

This driver performs tests on a method which parses text from HTML. For instance, the input “<p>An HTML paragraph</p>” would return the output “An HTML paragraph”. In order to pass the input and output from the driver in a safer, more compressed manner, the driver takes its input and returns output as base64 strings. Our test cases include HTML with multiple tags, nested tags, and combinations of both.

htmlpurifier_UnitConverter_getSigFigs.driver.php

This driver performs tests on a method that computes the number of significant figures in a string containing a decimal number. The method is part of a class which provides unit conversion implementations. The driver accepts a string as input, for example “01.23”, and provides the integer count of significant digits, in this case 3, as output. Our test cases cover various scenarios including cases with and without leading zeros and cases with all zeros.

maxmind_GeoIp2_Util_cidr.driver.php

This driver performs tests on a method that converts internet protocol addresses (IP) and their prefixes into Classless Inter-Domain Routing (CIDR) notation. This method is part of a class which provides IP address geolocation services. The driver accepts strings in the format “IP Address/Prefix” and passes the input to the method which returns a string in a similar format. For example, using the IP 192.168.86.42 with the prefix 16 would produce the output “192.168.0.0/16”. The method also accepts IPv6 addresses. Our test cases for this method examine a variety of scenarios including IPv4 and IPv6 addresses and both maximal and minimal prefixes.

Chapter 5: Injecting Faults and Testing the Framework

Introduction

In this chapter of our report, we will discuss the design and implementation of 5 faults into the Moodle source code, describe the development of a script that will automate the fault injection process, and analyze the results of the testing framework we have developed over the course of this project when these faults have been injected into the Moodle source code.

Designing Faults

When designing the faults to test our framework, we decided to inject one fault into each of the methods we had written test cases for. The faults are defined in the `./project/faults` directory where modified lines are indicated by comments in the format `“// FAULT:”` and the file that the fault definition should replace is specified on line 1 with a comment in the format `“#Destination: ./path/from/moodle”`. The details for the 5 faults we have defined will be provided below.

fault1.php

`./moodle/lib/html2text/Html2Text.php`

Line 144 changed so that the class improperly handles HTML with `
` elements.

`“/(<(br)[^>]*>[]*/i,” to “/(<(brk)[^>]*>[]*/i,”`

This fault causes the test case with the following id to fail: 8

fault2.php

`./moodle/lib/maxmind/GeoIp2/Util.php`

Line 23 changed so that the loop executes 1 time less then necessary.

“for (\$i = 0; \$i < \strlen(\$ipBytes) && \$curPrefix > 0; \$i++) {“

to

“for (\$i = 0; \$i < \strlen(\$ipBytes)-1 && \$curPrefix > 0; \$i++) {“

This fault causes the test case with the following id to fail: 12

fault3.php

./moodle/lib/htmlpurifier/HTMLPurifier/UnitConverter.php

Line 194 changed so that negative numbers are improperly handled.

“\$n = ltrim(\$n, '0+-');” to “\$n = ltrim(\$n, '0+');”

This fault causes the test case with the following id to fail: 19

fault4-5.php

./moodle/lib/evalmath/evalmath.class.php

Line 436 changed so that expressions with subtraction throw an error.

“} elseif (in_array(\$token, array('+', '-', '*', '/', '^', '>', '<', '==', '<=', '>='), true)) {“

to

“} elseif (in_array(\$token, array('+', '+', '*', '/', '^', '>', '<', '==', '<=', '>='), true)) {“

This fault causes the test case with the following id to fail: 2

Line 586 changed so that mod is calculated incorrectly.

“return \$op1 % \$op2;” to “return \$op2 % \$op1;”

This fault causes the test cases with the following ids to fail: 22, 23, 24

Injecting Faults

To automate the injection of faults into the Moodle source code, two routines (or actions) were added to the moodleMod script: inject and reset. The inject routine finds all fault definitions in `../project/faults/` and copies them to their destinations in the Moodle source code, which are specified in a comment in the first line of each fault definition file. The reset routine cleans the local Moodle repository, resetting it to the original state. The routines are executed by running the moodleMod script from the TestAutomation directory as follows:

```
./scripts/moodleMod.sh inject
```

```
./scripts/moodleMod.sh reset
```

In addition to managing faults, the moodleMod script also implements two other actions to manage the cloned Moodle repository: clone and delete. The clone action will clone the appropriate version of Moodle from the official Moodle repository into `../project/moodle` and the delete action will delete the cloned repository.

Testing the Framework

To test our framework we performed the following procedure:

1. Clone the New-Leaf team repository
2. Move into `../New-Leaf/TestAutomation/`
3. Clone Moodle by executing: `./scripts/moodleMod.sh clone`
4. Execute: `./scripts/runAllTests.py`
5. Record results without faults
6. Inject faults by executing: `./scripts/moodleMod.sh inject`
7. Execute: `./scripts/runAllTests.py`
8. Record results with faults
9. Remove faults by executing: `./scripts/moodleMod.sh reset`

Initial Results (without fault injection):

id	component	method	input	expected	output	result
1	evalmath	evaluate	4+5	9	9	Pass
2	evalmath	evaluate	5-6	-1	-1	Pass
3	evalmath	evaluate	5*5	25	25	Pass
4	evalmath	evaluate	25/5	5	5	Pass
5	evalmath	evaluate	100>10	1	1	Pass
6	html2text	getText	PCFET0NUW ...	VEhFIEhFQU ...	VEhFIEhFQU ...	Pass
7	html2text	getText	PHA+cGFyYW...	cGFyYWdyYX ...	cGFyYWdyYX ...	Pass
8	html2text	getText	PHA+cGFyYW...	cGFyYWdyYX ...	cGFyYWdyYX ...	Pass
9	html2text	getText	PGgxPmhlYW ...	SEVBREIORz ...	SEVBREIORz ...	Pass
10	html2text	getText	PGJvZHk+PH ...	cGFyYWdyYX ...	cGFyYWdyYX ...	Pass
11	maxmind_Geo ...	cidr	1.2.3.4/16	1.2.0.0/16	1.2.0.0/16	Pass
12	maxmind_Geo ...	cidr	1.2.3.4/31	1.2.3.4/31	1.2.3.4/31	Pass
13	maxmind_Geo ...	cidr	1.2.3.4/0	0.0.0.0/0	0.0.0.0/0	Pass
14	maxmind_Geo ...	cidr	ffff:ffff:ffff:f...	ffff::/16	ffff::/16	Pass
15	maxmind_Geo ...	cidr	2acd:ffff::aabb...	2acd:ffff::aa00...	2acd:ffff::aa00...	Pass
15	maxmind_Geo ...	cidr	2acd:ffff::aabb...	2acd:ffff::aa00...	2acd:ffff::aa00...	Pass
16	htmlpurifier_ ...	getSigFigs	0001.0123	5	5	Pass
17	htmlpurifier_ ...	getSigFigs	1.1	2	2	Pass
18	htmlpurifier_ ...	getSigFigs	12.304	5	5	Pass
19	htmlpurifier_ ...	getSigFigs	-10.01	4	4	Pass
20	htmlpurifier_ ...	getSigFigs	000.000	0	0	Pass
21	evalmath	mod	7,7	0	0	Pass
22	evalmath	mod	7,6	1	1	Pass
23	evalmath	mod	100,50	0	0	Pass
24	evalmath	mod	50,60	50	50	Pass
25	evalmath	mod	1,1	0	0	Pass
26	evalmath	ifthenelse	true,3,4	3	3	Pass
27	evalmath	ifthenelse	1,3,4	3	3	Pass
28	evalmath	ifthenelse	0,3,4	4	4	Pass
29	evalmath	ifthenelse	1+1==2,3,4	3	3	Pass
30	evalmath	ifthenelse	0==0,7,4	7	7	Pass

Summary: 30 / 30 tests passed.

Final Results (with fault injection):

id	component	method	input	expected	output	result
1	evalmath	evaluate	4+5	9	9	Pass
2	evalmath	evaluate	5-6	-1	PHP Fatal error: Uncaught Error:	Fail
3	evalmath	evaluate	5*5	25	25	Pass
4	evalmath	evaluate	25/5	5	5	Pass
5	evalmath	evaluate	100>10	1	1	Pass
6	html2text	getText	PCFET0NUW ...	VEhFIEhFQU ...	VEhFIEhFQU ...	Pass
7	html2text	getText	PHA+cGFyYW...	cGFyYWdyYX ...	cGFyYWdyYX ...	Pass
8	html2text	getText	PHA+cGFyYW...	cGFyYWdyYX ...	cGFyYWdyYX ...	Fail
9	html2text	getText	PGgxPmhlYW ...	SEVBREIORz ...	SEVBREIORz ...	Pass
10	html2text	getText	PGJvZHk+PH ...	cGFyYWdyYX ...	cGFyYWdyYX ...	Pass
11	maxmind_Geo ...	cidr	1.2.3.4/16	1.2.0.0/16	1.2.0.0/16	Pass
12	maxmind_Geo ...	cidr	1.2.3.4/31	1.2.3.4/31	1.2.3.0/31	Fail
13	maxmind_Geo ...	cidr	1.2.3.4/0	0.0.0.0/0	0.0.0.0/0	Pass
14	maxmind_Geo ...	cidr	ffff:ffff:ffff:f...	ffff::/16	ffff::/16	Pass
15	maxmind_Geo ...	cidr	2acd:ffff:aabb...	2acd:ffff:aa00...	2acd:ffff:aa00...	Pass
16	htmlpurifier_ ...	getSigFigs	0001.0123	5	5	Pass
17	htmlpurifier_ ...	getSigFigs	1.1	2	2	Pass
18	htmlpurifier_ ...	getSigFigs	12.304	5	5	Pass
19	htmlpurifier_ ...	getSigFigs	-10.01	4	5	Fail
20	htmlpurifier_ ...	getSigFigs	000.000	0	0	Pass
21	evalmath	mod	7,7	0	0	Pass
22	evalmath	mod	7,6	1	6	Fail
23	evalmath	mod	100,50	0	50	Fail
24	evalmath	mod	50,60	50	10	Fail
25	evalmath	mod	1,1	0	0	Pass
26	evalmath	ifthenelse	true,3,4	3	3	Pass
27	evalmath	ifthenelse	1,3,4	3	3	Pass
28	evalmath	ifthenelse	0,3,4	4	4	Pass
29	evalmath	ifthenelse	1+1==2,3,4	3	3	Pass
30	evalmath	ifthenelse	0==0,7,4	7	7	Pass

Summary: 23 / 30 tests passed.

As expected, all tests passed prior to fault injection, and after fault injection the expected test cases failed.

Chapter 6: Conclusions and Overall Experiences

Over the course of this project our team learned quite a bit about the software engineering process, including project planning and management, designing and producing good tests, and how to effectively work as a team on a software project.

We were also pleased with how our testing framework itself ended up turning out. Although we had some setbacks, we were not only able to design and implement our vision, but expand upon it. After completing the basic functionality of our framework earlier than expected we were able to spend time improving our project by adding features including improved output generation, improved error handling, and scripts to automate the process of fault injection. Additionally, we were also glad we were able to produce something we felt was valuable by writing our tests for Moodle libraries that did not have pre-existing tests.

To conclude, if it's not already clear, we were very satisfied with the results of our project and our development as a team and as software engineers.

Team Self Evaluation

The start of the semester was difficult for all team members getting used to working in a complete online environment and also learning to work with different types of people. During the first week we were able to delegate work evenly between team members, which allowed us to finish projects in advance so that we were able to spend more time on fixing minor things to improve the overall quality of our project. Throughout the semester we were constantly communicating and talking about current things we were working on as well as future assignments that we would eventually have to complete. Github was a great tool during this semester since it allowed us to easily see each other's contributions and share our work. Our team was also pleased with the results of our development of the automated testing framework we developed over the course of this semester, from our initial scripts and first attempts to our final product there was a clear and significant improvement that occurred as we continued to improve our skills. In conclusion, our team was very pleased with our advances in learning and project management as well as our final product and we all feel that we learned a lot about the software engineering process.

Assignment Evaluation

Throughout the course of this assignment our team felt that we developed our skills and achieved our project's goals. The assignment was very challenging in the first couple of weeks as we learned what would be expected of this assignment and what we would be working on throughout the semester. It took several meetings to grasp the specific way we were going to complete the project. After finally having our eureka moment we were able to begin working at a more consistent rate and start producing results. Overall the assignment was challenging and

made us struggle without having someone to save us immediately. Learning the power of research was key to succeeding in this project. If we were to do this project again we would start by fully understanding the requirements before writing any code in order to avoid wasting time with unnecessary misunderstandings. We enjoyed working in an new situation and it gave us a better understanding of life outside of college. We feel that we are often a bit sheltered from the realities of software engineering in our coursework and working on this project allowed us to develop a better understanding of something we weren't used to and has encouraged us to feel more confident working on larger projects in the future.