New Leaf Term Project Report

Chapter 3: Testing Framework

Luke McGuire, Chris Taylor, Kasper Dugaw
College of Charleston
Fall 2020

Introduction

This section will discuss in detail the structure and implementation of our testing framework, the framework's use cases and workflow, as well as 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 generated reports 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 providing the controller with the ability to test a particular method or class and 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 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. The controller is written in python, this is a bit of an odd choice since the remainder of our project and Moodle itself is written in PHP, but as the script does not require the ability to directly interact with PHP code, the choice of language did not really matter.

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 and instantiate an object from the class that we are then able to call methods on. We pass information to the drivers through our script (explained above) and then use that information to pass in specific inputs to the method that the driver controls. We create a different driver for each method to avoid complicated conditionals and allow other developers to easily add more for other classes they may want to test.

Test Cases

The test case specification is defined in chapter 2.

How to Use

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 the process of 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: ./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. Drivers are required to accept two parameters are command line arguments, the input for the test and the expected output. They also must echo a string in the format {result, status} to stdout to be captured by the controller.

Example Test Cases

At this point in time our team has completed 1/5 test drivers and 5/25 test cases. The driver that we have implemented will be discussed in the following subsections. We are currently finishing up another driver that will add to the ⅕ test drivers.

EvalMath evaluate

In the evaluate method inside of the EvalMath class, we are testing different mathematical operations that are inputted as a string. The method takes in a mathematical expression in the form of a string and parses it and calculates and returns the answer the method comes up with. We then check to see if the returned value is the same as the expected value and either pass or fail the test.