Canvas Testing Framework

Team *Don't Test Me* - Kenneth Dela Cruz, Sam Word, Kyle Glick
College of Charleston Department of Computer Science

Introduction

Objective:
Our objective in this project is to design and build an automated testing framework for a chosen Humanitarian Free Open Source Software (HFOSS) project.

Reasoning:
We chose to test Canvas LMS because it provides extensive documentation, is actively maintained, and is written for Ruby on Rails, a well-structured web application framework.

Test Requirements:
We designed 25 unit tests for 5 methods across 3 models. Each one tests a specific scenario within the method. We used Canvas's existing test suite to help determine our test cases.

Test Report Don't Test Me

Test Case	Model and Method	Requirements	Inputs	Oracle	Method Return	Result
01	Course#score_to_grade(score)	Converts a score of 0 to a grade when the grading scale is [["Pass",50],["Fail",0]]	0 [["Pass",0.5], ["Fail",0.0]]	Fail	Fail	Pass
02	Course#score_to_grade(score)	Converts a score of 25 to a grade when the grading scale is [["Pass",50],["Fail",0]]	25 [["Pass",0.5], ["Fail",0.0]]	Fail	Fail	Pass
03	Course#score_to_grade(score)	Converts a score of 49.999 to a grade when the grading scale is [["Pass",50], ["Fail",0]]	49.999 [["Pass",0.5], ["Fail",0.0]]	Fail	Fail	Pass
04	Course#score_to_grade(score)	Converts a score of 50 to a grade when the grading scale is [["Pass",50],["Fail",0]]	50 [["Pass",0.5], ["Fail",0.0]]	Pass	Fail	Fail
05	Course#score_to_grade(score)	Converts a score of 100 to a grade when the grading scale is [["Pass",50],["Fail",0]]	100 [["Pass",0.5], ["Fail",0.0]]	Pass	Fail	Fail

Figure 1: *An example test report generated by our testing framework, including 2 fault injection failures*

Framework Design

Our test script utilizes the *Rails Runner* component to execute Ruby test case executables inside of a native Ruby on Rails environment. This allows anyone to place the *TestAutomation* directory inside any Rails application and run tests for that application with minimal overhead requirements.

Team Repository:
github.com/
CSCI-362-02-2017/
Don-t-Test-Me

Framework Architecture

Our framework lives in *TestAutomation/*, which must be placed inside a Ruby on Rails project.

Tests are run with *./scripts/runAllTests.sh*. The script uses the data provided by test cases in *testCases/* to run tests located in *testCasesExecutables/*.

The script generates a report in *reports/* and opens it in a browser. Other helper code can be placed and included in *project/*.

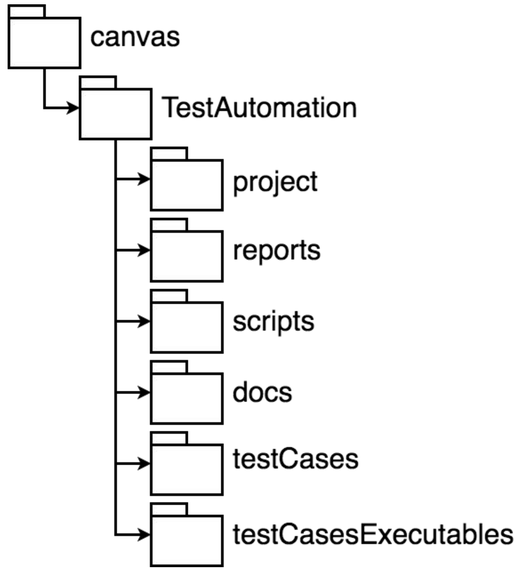

Figure 2: *The basic architecture of our test framework*

Test Cases

Model Methods tested:
Five unique test cases were created for each of the following methods:

- ➢ Course#score_to_grade
- ➢ LatePolicy#missing_points_deducted
- ➢ LatePolicy#points_for_missing
- ➢ Assignment#score_to_grade
- ➢ LatePolicy#points_deducted

Injected Faults

Objective:
Simulate common user error when typing. Initially, all of the test cases passed, so we injected faults to show that our test cases may catch some common errors when coding.

Example faults:

- ➢ Changing "If" to "unless"
- ➢ Changing "pass_fail" to "pass fail"
- ➢ Swapping "-" for a "+"

Observations:
Despite using test cases from Canvas's existing test suite, test case 06 (testing LatePolicy#missing_points_deducted) passed after a fault was injected to make it fail. We changed "pass_fail" to "pass fail" in the method, expecting the method to return the points possible multiplied by the missing submission deduction percentage instead of just the points possible. However, test case 06 originally set points_possible to 80 and missing_submission_deduction to 100%. In both cases, the method returned 80, making the test pass.