Brielan Beamon: beamonbh@g.cofc.edu
Logan Smith: smithla1@g.cofc.edu
Scott White: whites2@g.cofc.edu

Dr. Bowring
CSCI 362
September 27, 2016

Automated Testing Framework

Introduction

Our project is centered around Sugar Labs, a free and open source software that focuses on allowing people around the world to learn more about the world around them for free. Sugar Labs is a spin off of One Laptop Per Child, constituting a simple Linux-based operating system. Due to the complexity of emulating and then testing an entire operating system, we have opted to test individual components of Sugar Labs — which are referred to as activities.

Testing Framework

Architecture

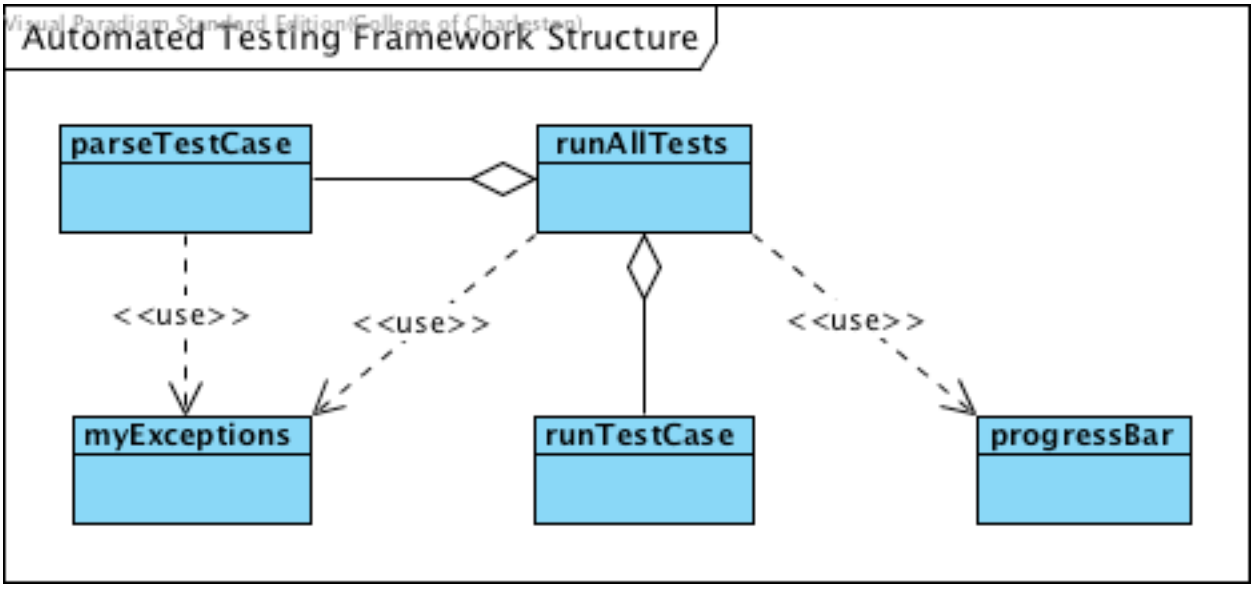
Our testing framework is composed of five python files that interact with the source code and a set of test cases. These five files are:

runAllTests.py
parseTestCase.py
runTestCase.py
myExceptions.py
progressBar.py

The first file, runAllTests.py, comprises the main logic of our testing framework. It's goal is to run every test case located in the testCases directory. runAllTests.py will then use parseTestCase.py to parse each test case (filtering out test cases that do not follow the naming convention) and collect all pertinent information (namely the test case ID, a path to a testable file, the method in that file to test, test inputs, and expected test outputs). With this information, runAllTests.py then uses runTestCase.py to run each test case. This will output the result of the tested method with the supplied input. runAllTests.py then compares this result to the expected output and determines if the test passed or failed from that determination.

The remaining two files, myExceptions.py and progressBar.py, add some utility to the automated testing framework. myExceptions.py defines a custom exception that is used as a means of ensuring durability. In the case where a testCase follows the naming convention, but does not follow the pre-defined template for its contents, a custom exception (ImproperTestCaseSpecification) is raised. This allows for errors due to improper structuring of

the test case (such as missing elements or superfluous elements) to be avoided. By raising this exception, the caller – runAllTests.py – knows to skip this test case. This prevents a poorly written test case from crashing the automated testing framework, and instead will allow the testing to continue.
progressBar.py simply allows for runAllTests.py to show the user through the command line interface roughly how much of the testing is complete at any given moment. It does this by calculating how many tests have been run versus the total number of test cases that will be run. This percentage is displayed next to a progress bar that adjusts itself to the width of the console. After finishing every test case, the progress bar will display "Complete" to signify completion.



How-To-Documentation
This automated testing framework requires no additional dependencies (other than python2.7). It can be invoked by running the following command from the home directory of the project:

./TestAutomation/scripts/runAllTests.py

This will start the automated testing framework, which will run all test cases that follow the formatting specifications. After each test case is run, a test report will be opened in a web browser that displays the results of each test case in the following format:

Testing Results					
Test Case ID	Tested Method	Input	Expected Output	Actual Output	Pass/Fail
ID number	*method name*	*arguments*	*expected output*	*actual output*	*Pass or Fail*

An example of this when populated by actual results may look something like this:

Testing Results

Test Case ID	Tested Method	Input	Expected Output	Actual Outcome	Pass/Fail
1	pow(a, b)	2, 10	1024	1024	Pass
2	add(a, b)	14.0, 7.5	21.5	21.5	Pass
3	mul(a, b)	15, -5	-75	-75	Pass
4	div(a, b)	120, 8	15	15	Pass
5	sqrt(a)	16	4.0	4.0	Pass

Test Cases

So far, we have a set of five test cases that test a few of the methods in functions.py, which is the backbone of the calculator activity in Sugar Labs. These test cases attempt to verify that various functions (pow, add, mul, div, sqrt) function properly on basic use cases.

The test cases are as follows:

```
Test Case ID: 1
```

```
Requirement being tested:

    Correct and accurate exponentiation
```

```
Component being tested:

    Sugar labs calculator - functions.py

    Path to file: TestAutomation/project/sugar-calculate/
functions.py
```

```
Method being tested: pow(a, b)
```

```
Test input(s) including command-line arguments: 2, 10
```

```
Expected outcome(s): 1024
```

Test Case ID: 2

Requirement being tested:

Accurate summation

Component being tested:

Sugar labs calculator - functions.py

Path to file: TestAutomation/project/sugar-calculate/functions.py

Method being tested: add(a, b)

Test input(s) including command-line arguments: 14.0, 7.5

Expected outcome(s): 21.5

Test Case ID: 3

Requirement being tested:

Accurate multiplication

Component being tested:

Sugar labs calculator - functions.py

Path to file: TestAutomation/project/sugar-calculate/functions.py

Method being tested: mul(a, b)

Test input(s) including command-line arguments: 15, -5

Expected outcome(s): -75

Test Case ID: 4

Requirement being tested:

Accurate division

Component being tested:

Sugar labs calculator - functions.py

Path to file: TestAutomation/project/sugar-calculate/functions.py

Method being tested: div(a, b)

Test input(s) including command-line arguments: 120, 8

Expected outcome(s): 15

Test Case ID: 5

Requirement being tested:

Accurate calculation of square roots

Component being tested:

Sugar labs calculator - functions.py

Path to file: TestAutomation/project/sugar-calculate/functions.py

Method being tested: sqrt(a)

Test input(s) including command-line arguments: 16

Expected outcome(s): 4.0