Joey Baldwin, Eric Hofesmann, Marge Marshall

CSCI 362 Software Engineering

November 15, 2016

Chapter 4: Completed Automated Testing Framework

Experiences:

For this deliverable, our primary goal was being able to test methods inside of a class. We've managed to stay fairly on schedule, meaning that a functional testing framework was already in place. Much of the work was in adding functionality to the existing framework and specifying new test cases. While what we have currently meets the criteria, we had hoped to be able to expand it in a few other ways that we weren't able to finish, such as testing for a thrown exception, and specifying test cases for setter methods.

Testing Framework Changes:

For our testing framework, we've added the capability to test methods within a class. This was done with a driver for each method that is tested, which is only called when a class is specified by the test cases. As a result of this change, we also slightly altered the format of our test cases to include a class parameter, which is passed on the same line as the module name, separated by a space. We have also added the ability to specify the precision of the testalmostequals parameter. This can be modified in the testCase file by adding a comma and an integer next to testalmostequals.

Test Cases:

The template for our test cases is as follows, with a description below:

<test case id>
<description of function's purpose>
<module name> <class name if applicable>
<function name>
<input>
<test type of expected output>
<expected output>
The test case id is designated with two digits, and starts with the first test case at 00.
The description is a plain English explanation of what the function does.
The module name is the module or file in which the function is found.
The function name is the name of the function being tested.
The input is the current value or collection of parameters being passed to the function.
The test type dictates what type of assertion will be used to determine if the actual output passes or not. This is given in the form of the name of the test functions, such as testexception or testequals. The expected output is what the function should return, given the input parameters.

A sample of current test cases:

00
calculate the frequency value of a given midi pitch
music
noteToFreq
69
testalmostequals, 2
440.00

05
Create range of floats
music
frange
1.0, 5.0, 0.5
testequals
[1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5]

10
calculate the color gradient given two colors and a step
gui
colorGradient
[0,0,0], [255,255,255], 2
testequals
[[0,0,0],[127,127,127]]

17
calculate the midi pitch of a given frequency value
music
freqToNote
451.0
testequals
69, 1751

20
Get if the timer repeats or not
timer Timer2
getRepeat
True
testequals
True

24
Get if the delay time interval (in milliseconds)
timer Timer2
getDelay
10000000000
testequals
10000000000