Unit Testing the Open-source project “Miradi” using an Automated Testing Framework

CSCI 362 - 01

Jacob Lipsey, Jacob Nash, Jacob Roddam, and Ian Dudderar

Abstract:

Automated testing frameworks are valuable resources to a software engineer, allowing them to test the whole, or part, of any project by executing a single file. Previously, the Miradi project didn’t have a testing framework, by using a previously existing project without a framework, making a framework will make this project easier to test and inject faults. Projects without a framework are much harder to test, therefore it is harder to confirm that the correct output is generated. Adding a testing framework will decrease the chances of incorrect outputs, and once they are verified to work correctly, injecting faults in the project files may produce incorrect output.

Introduction:

Having no prior experience creating an automated testing framework on a virtual machine, this project was a new challenge for all of us. By doing so, we will increase the usability of the program, as well as making it much easier to test the program; and inject faults that may cause the program to fail. Any project can benefit from an automated testing framework.

Fig. 1

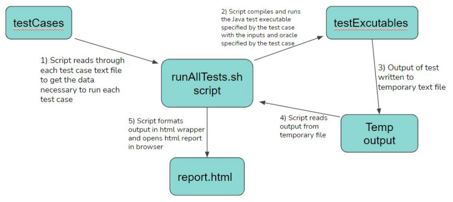
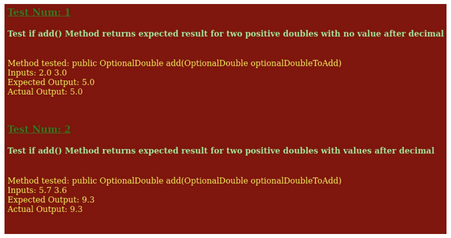

testCases
testExcutables
runAllTests.sh script
Temp output
report.html
1) Script reads through each test case text file to get the data necessary to run each test case
2) Script compiles and runs the Java test executable specified by the test case with the inputs and oracle specified by the test case
3) Output of test written to temporary text file
4) Script reads output from temporary file
5) Script formats output in html wrapper and opens html report in browser

Fig. 2



Test Num: 1

Test if add() Method returns expected result for two positive doubles with no value after decimal

Method tested: public OptionalDouble add(OptionalDouble optionalDoubleToAdd)
Inputs: 2.0 3.0
Expected Output: 5.0
Actual Output: 5.0

Test Num: 2

Test if add() Method returns expected result for two positive doubles with values after decimal

Method tested: public OptionalDouble add(OptionalDouble optionalDoubleToAdd)
Inputs: 5.7 3.6
Expected Output: 9.3
Actual Output: 9.3

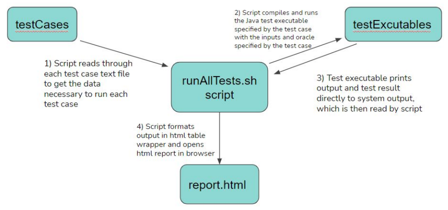
Fig. 3

Test #	Class	Method	Requirement	Inputs	Oracle
1	OptionalDouble	OptionalDouble add(OptionalDouble optionalDoubleToAdd)	add() Method returns correct sum of two positive doubles with no value after decimal	2.0, 3.0	5.0
2	OptionalDouble	OptionalDouble add(OptionalDouble optionalDoubleToAdd)	add() Method returns correct sum for two positive doubles with values after decimal	5.7, 3.6	9.3
3	OptionalDouble	OptionalDouble add(OptionalDouble optionalDoubleToAdd)	add() Method returns correct sum for two very large positive doubles	99999.9999, 123456.789	223456.78899
4	OptionalDouble	OptionalDouble add(OptionalDouble optionalDoubleToAdd)	add() Method returns correct sum for two very large negative doubles	-99999.99999, -123456.789	-223456.78899
5	OptionalDouble	OptionalDouble add(OptionalDouble optionalDoubleToAdd)	add() Method returns correct sum for a positive double and a small double	7.0, -5.6	1.4

Methods:

Since the virtual machine is an Ubuntu, we had to use shell script for our execution file for testing automation. The project was mostly in Java, so we had to download and install Java onto our machines in order to run any of the Miradi project. Once this was done, we were able to start working on the shell script to run the project. In order to do this we had to choose the files and methods we wanted to test. We chose the OptionalDouble class, testing methods: add,subtract, divideBy, and multiply; and IgnoreCaseStringComparator class, testing method: compare. Some of our initial test cases are shown in Fig. 3. In order to determine our tests’ output we create a html document, filling it as the script executes each test case on the respective java file, and open it at the end of the script. The initial html, shown in Fig. 2, showed all test outputs on a singular page, listing down. We changed this to have a html page for each test case with the same number respectively and the page that is pulled up has links to each one of these pages. The framework also was initially set up differently by having another folder labeled Temp output that was a middleman between the testExecutables folder and testing automation takes up less space. This is shown in the differences between Fig. 1, the initial framework, and Fig. 2, the current framework.

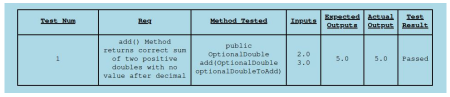
Fig. 4


testCases
testExcutables
runAllTests.sh script
report.html
1) Script reads through each test case text file to get the data necessary to run each test case
2) Script compiles and runs the Java test executable specified by the test case with the inputs and oracle specified by the test case
3) Test executable prints output and test result directly to system output, which is then read by script
4) Script formats output in html table wrapper and opens html report in browser

Results:

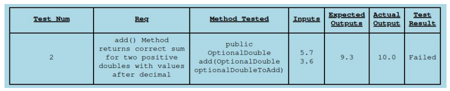
The results of implementing our testing framework were that we were able to verify that the current methods worked correctly for all of our testing cases, and only failed when we injected faulty code into the program. The result of the final html is shown in Fig. 7. Upon a click of any of the test case hyperlinks it opens a page similar to Fig. 5 with the respective number and data. Injecting the faults causes some test cases to fail resulting in the the Test Result to change to Failed, and the Expected and Actual Output will not match, shown in Fig. 6.

Fig. 5



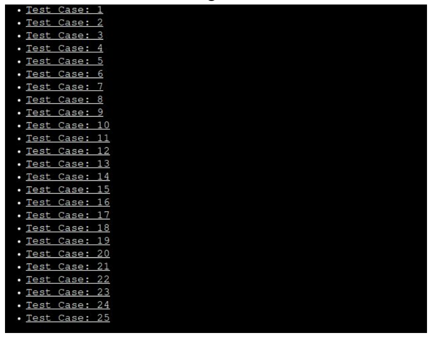
Test Num	Req	Method Tested	Inputs	Expected Outputs	Actual Output	Test Result
1	add() Method returns correct sum of two positive doubles with no value after decimal	public OptionalDouble add(OptionalDouble optionalDoubleToAdd)	2.0 3.0	5.0	5.0	Passed

Fig. 6



Test Num	Req	Method Tested	Inputs	Expected Outputs	Actual Output	Test Result
2	add() Method returns correct sum for two positive doubles with values after decimal	public OptionalDouble add(OptionalDouble optionalDoubleToAdd)	5.7 3.6	9.3	10.0	Failed

Fig. 7



- Test Case: 1
- Test Case: 2
- Test Case: 3
- Test Case: 4
- Test Case: 5
- Test Case: 6
- Test Case: 7
- Test Case: 8
- Test Case: 9
- Test Case: 10
- Test Case: 11
- Test Case: 12
- Test Case: 13
- Test Case: 14
- Test Case: 15
- Test Case: 16
- Test Case: 17
- Test Case: 18
- Test Case: 19
- Test Case: 20
- Test Case: 21
- Test Case: 22
- Test Case: 23
- Test Case: 24
- Test Case: 25

Fig. 8

Class	Method	Fault Description	Test #s Caused to Fail
OptionalDouble.java	OptionalDouble add(OptionalDouble optionalDoubleToAdd)	Rounds sum to nearest integer before returning	2, 3, 4, 5
OptionalDouble.java	OptionalDouble subtract(OptionalDouble optionalDoubleToSubtract)	Returns absolute value of difference	7
OptionalDouble.java	OptionalDouble divideBy(OptionalDouble optionalDoubleToDivideBy)	Divisor and dividend are switched	22, 23, 24, 25
OptionalDouble.java	OptionalDouble multiply(OptionalDouble optionalDoubleToMultiply)	Multiplies the product by -1	16, 17, 19, 20
IgnoreCaseStringComparator.java	int compare(Object object1, Object object2)	Doesn’t ignore case for comparison	12

Conclusion:

Working throughout the course of the project the testing framework wasn’t fully automated but once it was, it was clear why automated testing is used. Once the framework is set up, testing the project took only a single execution of a single file. In conclusion, this project has helped us realize the power of an automated testing framework as well as the capabilities of shell scripts. Testing frameworks are, and will continue to be a tool for software engineers.

References:

Cooper M. (2014). *Advanced Bash Scripting Guide: an in-depth exploration of the art of shell scripting*. Independent publisher.

Garrels, M. (2010). *Bash guide for beginners*. Palo Alto, CA: Fultus Books.

Getodk. (2020, December 02). Getodk/collect. Retrieved September, 2020, from https://github.com/getodk/collect

Gite, V. G. (2002). Linux Shell Scripting Tutorial v1.05r3 A Beginner's handbook. Retrieved September, 2020, from http://www.freeos.com/guides/lsst/

The Linux Documentation Project. (2020, August 08). Retrieved September, 2020, from https://tldp.org/

Sitkatech. (2019, July 17). Sitkatech/miradi-client. Retrieved October, 2020, from https://github.com/sitkatech/miradi-client