Unit Testing the Open-source project “Miradi”

Team 5

Ian Dudderar, Jacob Lipsey,
Jacob Nash, Jacob Roddam

Table of Contents

Commented [JN1]: If we change anything we need to make sure we update the page numbers here

Abstract

Write this after everything else.

Background

Linux, the UNIX-like OS named after Linus Torvalds who helped create it, is the most famous and most widely used open-source software ever. Part of this is due to the flexible nature of Linux, allowing it to be useful in thousands of different ways. This flexibility is fundamental to its open-source nature—here, open-source meaning literally that, that the *source* code is *openly* available to anyone and everyone. And since the source is open to all, anyone may change it in any way they see fit, for any reason. This has allowed Linux to become more than just one OS; it has been transformed into thousands of unique instances. Though by no means the first open-source software, the popularity and usefulness of Linux (and other softwares) have inspired many to turn their own endeavors into open-source projects, in order that their software might benefit from other's work and insights and that more people would find use in the programs than if they were proprietary. Among such software are both ODK Collect and Miradi, the two open-source projects that we used throughout the semester.

Commented [JN2]: Do you guys think we need these sections?

Chapter 1

At the beginning of the semester, we were planning on testing the software ODK (Open Data Kit). Eventually, we had to change projects due to lack of thorough documentation on ODK as well as not being able to find testable classes (at least, tests that could be automated) within ODK. We changed direction, and switched our project to Miradi, an open-source software designed to help worldwide conservation efforts. Miradi would prove to have its own issues, but we were able to build and run the program. Since we were all still learning how to use the Linux command line, we all felt satisfied once we had the software built and running. At this point in

the semester, we didn't have a good grasp yet of where we were headed with this project and didn't yet understand how we would be testing this software, but we weren't terribly concerned because we seemed to be on par with the other teams.

Chapter 2

For deliverable #2, we produced the following test plan:

Testing Process
Each method will be isolated in its class and built along with any dependencies necessary so that a main driver can be run from the command line, along with a file containing any necessary inputs. The expected output will be compared to the received output to determine success or failure.

Requirements Traceability
Many of the methods being used will have a dependency on other objects, classes, or methods in other directories existing in the Miradi repo. These will also be tested and well-documented to ensure that there are no loose-ends to minimize potential reasons for a failed test.

Planned Schedule

10-13: Revised testing plan presented, 5 test cases specified
10-15: All teams have success building classes from command line
10-20: Final decisions of classes/methods to test
10-27: Implemented at least one test using automated testing script
11-5: Deliverable #3 --> Automated testing framework designed and operable; also, at least 15 test cases designed (and hopefully tested)
11-12: At least 20 test cases designed (and hopefully tested); also, begin "testing" testing framework
11-17: Deliverable #4 --> Testing framework revised and finalized, 25 test cases designed and tested
11-20: Re-analysis of testing framework and test cases, inspection for possible faults
11-24: Deliverable #5 --> Fault injection testing & report
11-27: Final report accumulation, and presentation designed
12-1: Team comfortable with project presentation; have practiced more than once
12-3: Final Report --> Project presentations

Procedures
Each method will be traced and recorded, ensuring that any necessary inherited classes or abstract methods are appropriately documented. From this information, we will study what is being performed so that we can understand the purpose of the function as well as predict the

overall outcome. Test cases will be developed and written down, along with their expected outputs. After testing, pass/fail will be listed as well as any reasons why the test case was successful or a failure.

Hardware and Software Requirements
In order to enact our system of testing we will be using a Linux-based VM terminal as well as an installed JDK and JRE. This can be performed by entering '$ sudo dnf install java-latest-openjdk' on the command line.

Constraints
Our current and predicted future constraints are as follows but not limited to:

-Lack of prior knowledge and experience in github forking/cloning, as well as Linux
-Omission of an easy-to-use build file provided by Miradi
-Poor communication and group understanding placed by virtual constraints due to COVID
-Time management issues throughout the semester colliding with other courses
-Having to retrace certain steps as we encounter dead-ends in our progress

We were mostly able to hold to this plan, with a few exceptions in the schedule due to unplanned occurrences such as changing work schedules and difficulty finding a time for all team members to “meet” virtually. Also, the original 5 test cases that were specified with this deliverable were discarded in favor of others, so they have been omitted here; test cases actually implemented are given in the next section and in the project submission.

Chapter 3

In deliverable 3, we included the following how-to build steps, a diagram of the testing framework, and the first 5 test cases.

How-To:
(Step 1) Create an Ubuntu 20.04 Virtual Machine or use a working Ubuntu 20.04 Virtual Machine.

(Step 2a) Verify that java 8 is on your device and not a different version. If it isn't java 8, and you don't have java 8 installed, then type the following into the terminal, and after the last command select java 8. "sudo apt-get update" "sudo apt-get install openjdk-8-jre" "sudo update-alternatives --config java"

(Step 2b) If you do have java 8 installed, but it is not the default, then type the following into the terminal and select java 8. "sudo update-alternatives --config java"

(Step 3) Clone the Team-5 repository to the Virtual Machine.

(Step 4) In the terminal, move to directory /Team-5/TestAutomation/ and run the runAllTests.sh file with the following command, your default browser should appear with the test results of all tests. "./scripts/runAllTests.sh"

Framework Directory Structure

/TestAutomation
- /project
  - /src
    - DoubleUtilities.class
    - DoubleUtilities.java
    - EnhancedJsonArray.class
    - EnhancedJsonArray.java
    - EnhancedJsonObject.class
    - EnhancedJsonObject.java
    - FloatingPointFormatter.class
    - FloatingPointFormatter.java
    - IgnoreCaseStringComparator.class
    - IgnoreCaseStringComparator.java
    - IntVector.class
    - IntVector.java
    - OptionalDouble.class
    - OptionalDouble.java
    - StringList.class
    - StringList.java
    - StringUtilites.class
    - StringUtilites.java
  - /bin/org
    - /json
      - JSONArray.class
      - JSONArray.java
      - JSONObject$Null.class
      - JSONObject.class
      - JSONObject.java
      - JSONTokener.class
      - JSONTokener.java
    - /miradi/utils
- /scripts
  - runAllTests.sh
- /testCases
  - testCase01.txt
  - testCase02.txt
  - testCase03.txt

Commented [JN3]: I'm pretty sure he doesn't want the compiled class files in here.

Commented [JN4]: Do we need these? Do we use them?

testCase04.txt
testCase05.txt
testCase06.txt
testCase07.txt
testCase08.txt
testCase09.txt
testCase10.txt
testCase11.txt
testCase12.txt
testCase13.txt
testCase14.txt
testCase15.txt
testCase16.txt
testCase17.txt
testCase18.txt
testCase19.txt
testCase20.txt
testCase21.txt
testCase22.txt
testCase23.txt
testCase24.txt
testCase25.txt
/testCaseExecutables
DoubleUtilities.java
FloatingPointFormatter.java
OptionalDouble.java
TestAdd.java
TestDivideBy.java
TestMultiply.java
TestSubtract.java
/org/miradi/utils
FloatingPointFormattor.class
OptionalDouble.class
TestAdd.class
TestSubtract.class
/oracles
/docs
README.txt
TestRemove.txt
Test_Case_template.txt
Testadd.txt
/reports
report.html
report1.html
report2.html
report3.html

Commented [JN5]: Should we take these out? Like up above, I think we ought to

Commented [JN6]: Currently in GitHub our reports are in here? Also, what are TestRemove and TestAdd?

report4.html
report5.html
report6.html
report7.html
report8.html
report9.html
report10.html
report11.html
report12.html
report13.html
report14.html
report15.html
report16.html
report17.html
report18.html
report19.html
report20.html
report21.html
report22.html
report23.html
report24.html
report25.html

Commented [JN7]: Why do we have 26 different reports? Shouldn't they be cleaned out each time before it runs?

Test Cases

The format for test cases is:

Test Number
Requirement being tested
Method being tested
Input1 Input2 ...
Expected Output
Test Execuatble name (without ".java" i.e. TestAdd)

Here are the first five test cases:

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
TestAdd

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
TestAdd

3
add() Method returns correct sum for two very large positive doubles
public OptionalDouble add(OptionalDouble optionalDoubleToAdd)
99999.99999 123456.789
223456.78899
TestAdd

4
add() Method returns correct sum for two very large negative doubles
public OptionalDouble add(OptionalDouble optionalDoubleToAdd)
-99999.99999 -123456.789
-223456.78899
TestAdd

5
add() Method returns correct sum for a positive double and a small double
public OptionalDouble add(OptionalDouble optionalDoubleToAdd)
7.0 -5.6
1.4
TestAdd

Chapter 4

For deliverable 4, we finished adding in all 25 of our test cases, and we changed the html format to look nicer, we made it into a chart showing test case ID, requirement being tested, inputs, expected output, and whether the test passed or failed. Previously, we didn't have all of this information displayed and easily located, so we added in what we were missing as well as making the format look nicer. The specific 25 test cases have been omitted here for brevity; they can be found in /Team-5/TestAutomation/testCases/ if you would like to view them all.

Chapter 5

For deliverable 5, we added in faults to our testing framework to see how they were handled.

Instructions for running code with and without faults:
-Each fault will be located in the specified class and method within the "testExecutables" folder

-Original and faulty code segments are specified by comments
-By default, the original code is commented out and the faulty code runs
-To switch back to the original code simply uncomment out the original code and comment out the faulty code
-Upon running scripts/runAllTests.sh again, the program will run with the original code

Fault 1:

Class: OptionalDouble
Method: add(OptionalDouble OptionalDoubleToAdd)

Change Made: Each value involved in the addition is rounded using the method "round" from the java.math library. Doing so allows each double to be changed to its nearest whole number, and then parsed to an integer.

Effects:
Test ID #2
Expected Result: 9.3
New Result: 10
Failed
Test ID #3
Expected Result: 223456.7889
New Result: 223457
Failed
Test ID #4
Expected Result: -223456.78899
New Result: -223457
Failed
Test ID #5
Expected Result: 1.4
New Result: 1
Failed
Note: Test #1 unaffected by change

Fault 2:

Class: OptionalDouble
Method: subtract(OptionalDouble OptionalDoubleToSubtract)

Change Made: Previously the code would find the difference between two numbers. After the inserted changes, the code now returns the absolute value of the difference of the two numbers, resulting in the inability to return a negative number.

Effects:
Test ID #7
Expected Result: -8.1
New Result: 8.1
Failed

Note: Tests #6, #8, #9, #10 unaffected by change

Fault 3:

Class: OptionalDouble
Method: divideBy(OptionalDouble optionalDoubleToDivideBy)

Change Made: Previously the code would return the dividend divided by the divisor. The changes to the code swap these two values, so that the dividend becomes the divisor and vice versa.

Effects:
Test ID #22
Expected Result: 0.0368767629
New Result: 27.11345
Failed
Test ID #23
Expected Result: 222.1059113300
New Result: 0.00450
Failed
Test ID #24
Expected Result: -997.1319311663
New Result: 0.001002876
Failed
Test ID #25
Expected Result: 2.2829181495
New Result: 0.438035853
Failed
Note: Test #21 unaffected by change.

Fault 4:

Class: OptionalDouble
Method: multiply(OptionalDouble optionalDoubleToMultiply)

Change Made: The calculated product in the original code is now multiplied by -1 before returning the value.

Effects:
Test ID #16
Expected Result: 3189.34
New Result: -3189.34
Failed
Test ID #17
Expected Result: -105.148449
New Result: 105.148449
Failed
Test ID #19

Expected Result: 0.00000154737
New Result: -0.00000154737
Failed
Test ID #20
Expected Result: -11.115179348
New Result: 11.115179348
Failed
Note: Test #18 unaffected by change.

Fault 5:
Class: IgnoreCaseStringComparator
Method: compare(Object object1, Object object2)

Change Made: Inserted code so that the method no longer ignores case during the comparisons.

Effects:
Test ID #12
Expected Result: 0 (false)
New Result: 1 (true)
Failed
Note: Test #11, #13, #14, #15 unaffected by change.

Chapter 6

Overall team experiences and what we learned from this project.

Commented [JN8]: I wanted all your input before I wrote this

Appendix A

Team 5 self-evaluation

something

Commented [JN9]: I also wanted y'alls input here too

Comments on course and suggestions for improvements

something

Commented [JN10]: Your input here too