Automated Testing Framework

By Ryan Barrett, Jimmy Chu, Tyler Malone

Team CtrlAltElite

CSCI362 Fall 2019

Table of Contents

Introduction

For our project, we decided on the Spatiotemporal Epidemiological Modeler (STEM) open-source project because we found it to be very interesting and of high importance to test. The spread of diseases and public health are two very important subject matters and areas of inquiry. The application is created using Maven and is hosted on SVN repositories. The purpose of the application is to allow users to create scenarios that model the spread of disease in order to better visualize and analyze these models. The goal of this project is to create an automated testing framework that will include 25 test cases to validate the correctness of 5 of the methods used in STEM.

Chapter 1

We had some difficulties retrieving the source code. The links provided to browse the repository online failed to work and we were only able to retrieve the code using the eclipse import functionality using the url. The Eclipse STEM website has meticulous instructions on how to run the application; however, they are dated and no longer completely reliable. For instance, the directions wanted us to install two different plugins, one of which was not available to download in Eclipse. We found a similarly named plugin and installed that one instead, hoping that it would work. After very carefully following the directions, the application failed to run on Linux and gave a great deal of error messages. The program would begin to load, but very quickly shut itself down. The application did run on Windows with no error after completion of the directions.

In order to run STEM following their directions, the user has to run a *.product file that we believe to be unique to Eclipse. We were unable to build or run the application using Maven on the command line or in Eclipse and when attempting to do so were only greeted with error messages. One recurring error message told us that we were missing a dependency that we have been unable to resolve. In order to possibly resolve some of these issues, we emailed the entire STEM development team asking for help, specifically with running STEM on Linux and with running the included test cases. We never heard back from them.

Chapter 2

We found a class in our open-source project called MathOps.java that contains the 5 methods we are going to use to test our automated framework. The methods are testable with very definite outputs. The methods are such that there are plenty of possible test cases and, because the methods are mathematically based, the correct results are indisputable.

We have developed five test cases for the gcd (greatest common divisor) method in the MathOps.java file. Because of the broad range of possible inputs, we came up with five partitions and created a test case to reflect each one. This was a great mental exercise in which we had to think creatively to cover as many bases as we could. We also developed a test case template that defines the structure of each test case.

We also developed a hypothetical test plan that's similar to what STEM developers might have created for themselves. In it, we describe several things like the testing process, schedule, and constraints. This test plan could serve as the basis for a real life scenario. This was a unique experience as we had to think about the challenges and expectations that the actual developers might face while developing STEM.

Chapter 3

Our team is designing and building an automated testing framework by writing a script using Bash that will be called from the Linux terminal. The framework is going to test 5 methods from the open-source project STEM (Spatiotemporal Epidemiological Modeler) written in Java. The 5 methods we are testing are: gcd, lcm, argMin, argMax, and approxEqual. They all come from the MathOps.java library in STEM. In total, we have developed 5 test cases for each method for a total of 25 test cases.

The first method we created test cases for was gcd (greatest common divisor). This method takes in 2 integers as arguments and calculates the greatest common divisor from the 2 integers. The inputs for the 5 test cases we have developed for the gcd method are: (30, 12), (20, 20), (0, 50), (2147483647, 5), and (-10, 17). These test cases were chosen by coming up with partitions for gcd and then creating a test case from each partition in order to test gcd as completely as possible. We then followed a similar process to come up with test cases for each of the other 4 methods we're testing.

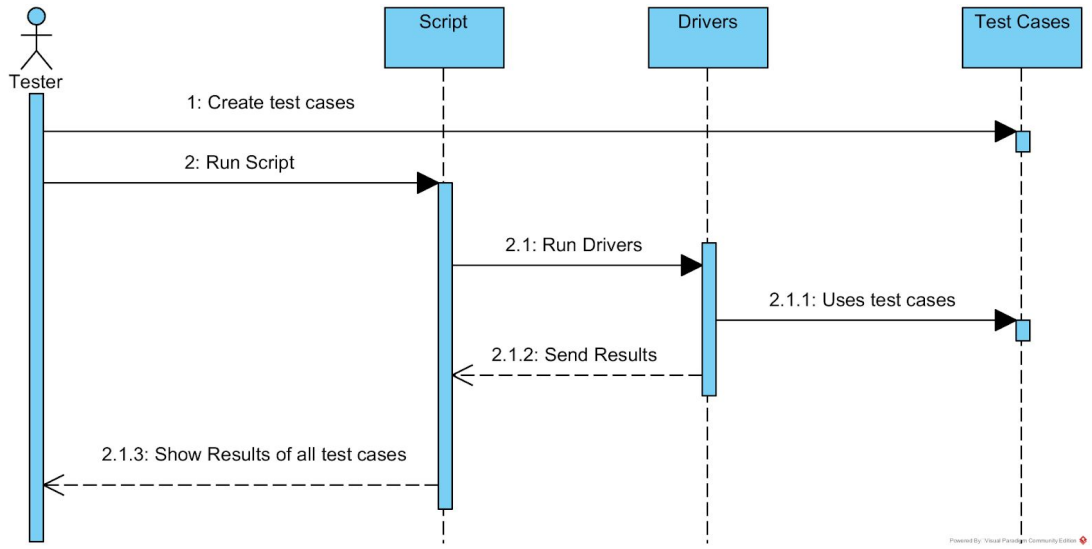
We have also created a driver for each of the 5 methods we are testing as part of our automated testing framework. Our script references each test case file, determines the appropriate driver to use based on the driver specified in that file, then runs that driver with the arguments also specified in the file. As the script runs, it writes the results of each test case into the HTML file organized in a table. The file then automatically opens in a web browser.

One problem we ran into was trying to create test cases for the sumLogs method. It's unclear exactly what the sumLogs method does so we had to switch from testing sumLogs to testing approxEqual. The only way we can tell what it's doing is by tracing the algorithm. There isn't any way to come up with test cases other than by looking at the algorithm to see how the method works. From our research, we could not find any similar established mathematical process that we could use to verify the correctness of sumLogs. For this reason, we have switched to using approxEqual.

Framework How-To

Step 1: Create test case(s) conforming to the test case template (refer to the testCaseTemplate.txt file in the docs directory).
Step 2: Execute runAllTests.sh on Linux terminal.
Step 3: Enjoy!

Framework Architecture



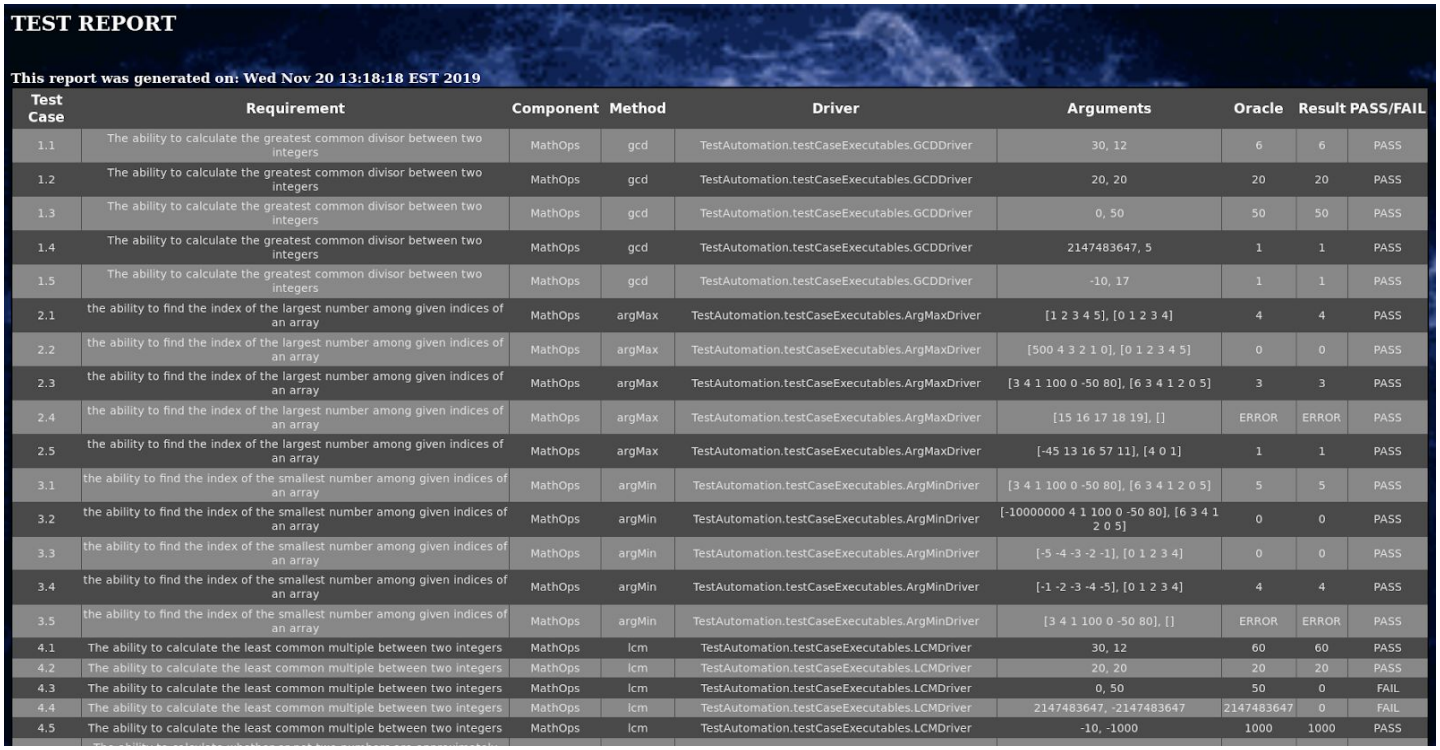
Sample Test Cases

```
1
2
3  1.1
4  The ability to calculate the greatest common divisor between two integers
5  MathOps
6  gcd
7  TestAutomation.testCaseExecutables.GCDDriver
8  30
9  12
10 6
11
```

```
1
2
3  4.3
4  The ability to calculate the least common multiple between two integers
5  MathOps
6  lcm
7  TestAutomation.testCaseExecutables.LCMDriver
8  0
9  50
10 50
11
```

```
1
2
3  2.3
4  the ability to find the index of the largest number among given indices of an array
5  MathOps
6  argMax
7  TestAutomation.testCaseExecutables.ArgMaxDriver
8  [3 4 1 100 0 -50 80]
9  [6 3 4 1 2 0 5]
10 3
11
```

Screenshot of Generated Testing Report



TEST REPORT

This report was generated on: Wed Nov 20 13:18:18 EST 2019

Test Case	Requirement	Component	Method	Driver	Arguments	Oracle	Result	PASS/FAIL
1.1	The ability to calculate the greatest common divisor between two integers	MathOps	gcd	TestAutomation.testCaseExecutables.GCDDriver	30, 12	6	6	PASS
1.2	The ability to calculate the greatest common divisor between two integers	MathOps	gcd	TestAutomation.testCaseExecutables.GCDDriver	20, 20	20	20	PASS
1.3	The ability to calculate the greatest common divisor between two integers	MathOps	gcd	TestAutomation.testCaseExecutables.GCDDriver	0, 50	50	50	PASS
1.4	The ability to calculate the greatest common divisor between two integers	MathOps	gcd	TestAutomation.testCaseExecutables.GCDDriver	2147483647, 5	1	1	PASS
1.5	The ability to calculate the greatest common divisor between two integers	MathOps	gcd	TestAutomation.testCaseExecutables.GCDDriver	-10, 17	1	1	PASS
2.1	the ability to find the index of the largest number among given indices of an array	MathOps	argMax	TestAutomation.testCaseExecutables.ArgMaxDriver	[1 2 3 4 5], [0 1 2 3 4]	4	4	PASS
2.2	the ability to find the index of the largest number among given indices of an array	MathOps	argMax	TestAutomation.testCaseExecutables.ArgMaxDriver	[500 4 3 2 1 0], [0 1 2 3 4 5]	0	0	PASS
2.3	the ability to find the index of the largest number among given indices of an array	MathOps	argMax	TestAutomation.testCaseExecutables.ArgMaxDriver	[3 4 1 100 0 -50 80], [6 3 4 1 2 0 5]	3	3	PASS
2.4	the ability to find the index of the largest number among given indices of an array	MathOps	argMax	TestAutomation.testCaseExecutables.ArgMaxDriver	[15 16 17 18 19], []	ERROR	ERROR	PASS
2.5	the ability to find the index of the largest number among given indices of an array	MathOps	argMax	TestAutomation.testCaseExecutables.ArgMaxDriver	[-45 13 16 57 11], [4 0 1]	1	1	PASS
3.1	the ability to find the index of the smallest number among given indices of an array	MathOps	argMin	TestAutomation.testCaseExecutables.ArgMinDriver	[3 4 1 100 0 -50 80], [6 3 4 1 2 0 5]	5	5	PASS
3.2	the ability to find the index of the smallest number among given indices of an array	MathOps	argMin	TestAutomation.testCaseExecutables.ArgMinDriver	[-10000000 4 1 100 0 -50 80], [6 3 4 1 2 0 5]	0	0	PASS
3.3	the ability to find the index of the smallest number among given indices of an array	MathOps	argMin	TestAutomation.testCaseExecutables.ArgMinDriver	[-5 -4 -3 -2 -1], [0 1 2 3 4]	0	0	PASS
3.4	the ability to find the index of the smallest number among given indices of an array	MathOps	argMin	TestAutomation.testCaseExecutables.ArgMinDriver	[-1 -2 -3 -4 -5], [0 1 2 3 4]	4	4	PASS
3.5	the ability to find the index of the smallest number among given indices of an array	MathOps	argMin	TestAutomation.testCaseExecutables.ArgMinDriver	[3 4 1 100 0 -50 80], []	ERROR	ERROR	PASS
4.1	The ability to calculate the least common multiple between two integers	MathOps	lcm	TestAutomation.testCaseExecutables.LCMDriver	30, 12	60	60	PASS
4.2	The ability to calculate the least common multiple between two integers	MathOps	lcm	TestAutomation.testCaseExecutables.LCMDriver	20, 20	20	20	PASS
4.3	The ability to calculate the least common multiple between two integers	MathOps	lcm	TestAutomation.testCaseExecutables.LCMDriver	0, 50	50	0	FAIL
4.4	The ability to calculate the least common multiple between two integers	MathOps	lcm	TestAutomation.testCaseExecutables.LCMDriver	2147483647, -2147483647	2147483647	0	FAIL
4.5	The ability to calculate the least common multiple between two integers	MathOps	lcm	TestAutomation.testCaseExecutables.LCMDriver	-10, -1000	1000	1000	PASS

Chapter 4

At this point, we have successfully come up with all 25 of our test cases that our framework will use to test our 5 methods and all 25 test cases and their results are displayed in the report. Also, we have made our report much more aesthetically pleasing. For the background, we used a galaxy gif that isn't too distracting. We encoded this gif using a 64 bit encoder and placed the encoded string into our CSS file. We also used a tableview CSS generator tool we found online to make our tableview stylish. Overall, the report now looks much more professional and is more pleasing to look at. In order to allow users to move around the report html file while keeping the CSS style, we don't refer to the CSS file or the gif file in the html, but rather read the contents of the CSS file into the report html file on creation.

We also added additional information to our report. We have added columns for the drivers and components. The report now also includes the date and time at which the report was generated. In addition, we corrected our project file structure. This caused us an unexpected complication. We had to change the way in which we compile our java source files, which was the biggest problem we encountered. We now pipe the output of a recursive search function that finds all java files including those in subdirectories. We also of course had to modify the file paths in our script and modify the packaging of our java files. We also had to change each test case to reflect the new path of its relevant driver. After doing all of that, our script now works with our new file structure.

Chapter 5

We injected 5 faults into the source code we were testing. Before injecting these faults, 24 of our 25 test cases passed. After injecting these faults, only 8 of our 25 test cases passed. This demonstrates that our test cases tested inputs from multiple partitions and cover many possibilities. If so many of our tests hadn't failed after injecting faults, it would have shown that our test data was of poor quality. We attempted to inject faults that were bad enough but didn't completely annihilate the functionality of the code. This was a fun, but frustrating experience because we had to look for changes to make that would only cause some of our test cases to fail rather than all of them.

The faults we injected into our source code files are as follows:

1. Modified the lcm method from
    a. return Math.abs(arg0 * arg1) / MathOps.gcd(arg0, arg1); to
    return Math.abs(arg0 * arg1) / MathOps.gcd(arg0, arg0);
    b. We expected this to cause all test cases with arguments that aren't equal to fail.
2. Changed the gcd loop while condition from rest > 0 to rest >= 0.
    a. We expected this to cause an infinite loop and cause all test cases to fail. This is precisely what happened. Since lcm uses this method, we expected it to also cause all test cases for lcm to fail, which is what happened.
3. In argMax, changed entries[element] > entries[argmax] to entries[element] < ….
    a. We expected this to cause all test cases to fail except for (0, 50)
4. In argMin, changed entries[element] < entries[argmax] to entries[element] > …
    a. We expected this to cause all test cases to fail except for (0, 50)
5. Changed dbl1 - dbl2 to dbl1 + dbl2 in approxEqual
    a. We expected this to cause only (1, 1) to fail

Chapter 6

Overall, this was a challenging, yet rewarding experience. We learned about some of the capabilities of a scripting language in Bash. We learned how to create and use driver methods in our framework, how to navigate through our files and perform commands using the Linux terminal, and we got to practice our visual design skills with our testing report using HTML and CSS. In addition, we practiced our teamwork skills and learned to work as a cohesive whole.

Aside from Ryan, we were wholly unfamiliar with using git at the beginning of this project and were a bit intimidated by it. After a few rounds of pushing and pulling from GitHub and a lot of help from Ryan, we have become much more comfortable with git and have started using it for projects outside of class as well.

Tyler was also unfamiliar with what a driver was at the beginning of this project. He has really only heard of printer drivers, and the thought of creating something like that from scratch was even more intimidating than git. However, both Ryan and Jimmy assured him that the drivers we would be creating wouldn’t be too complex.

Chapter 7

Overall, our team worked well together throughout the progression of this project. We completed the objectives of every deliverable on time and went above and beyond in our visual quality of the report. Our test cases covered a very wide array of scenarios and our testing framework will be very easily utilized by “customers”. We didn’t encounter any significant problems throughout the course of the assignment, and any problems that we did encounter got resolved fairly quickly. Our testing framework works as it’s supposed to and is simple to use. In this group’s opinion, we nailed it.

Chapter 8

In evaluating the project assignments, we believe that the scheduling of deliverables spaced out the workload nicely and we were exposed to a wide array of activities and technologies throughout the semester. One suggestion we have for improving the project in the future would be to provide a demonstration of a previous group's framework in the beginning of the semester so that we have an idea of what the end result will look like because coming in, everything about the project was unfamiliar to us and seemed a bit overwhelming at first.