

STEM Automated Testing Framework  
Final Report  
CSCI 362.01 Fall 2020

Noah Drake, Chloe Harris, Tristan Lawler, Ryan Ratliffe

# Table of Contents

<b>Chapter 1</b>	<b>2</b>
Evaluating Martus	2
Exploring STEM (The Spatiotemporal Epidemiological Modeler)	2
<b>Chapter 2</b>	<b>4</b>
Directory Structure	4
Testing Framework Description	5
Test Cases	6
Resource Constraints	8
Dependencies and System Requirements	8
<b>Chapter 3</b>	<b>9</b>
<b>Chapter 4</b>	<b>10</b>
<b>Chapter 5</b>	<b>11</b>
Faults injected	11
org.eclipse.stem.graphgenerators.impl.PajekNetGraphGeneratorImplOld	11
org.eclipse.stem.analysis.automaticexperiment.NelderMeadAlgorithm	11
org.eclipse.stem.analysis.impl.ReferenceScenarioDataMapImpl	11
org.eclipse.stem.core.math.BinomialDistributionUtil	11
org.eclipse.stem.core.model.impl.STEMTimeImpl	11
<b>Chapter 6</b>	<b>12</b>

# Chapter 1

## Evaluating Martus

At the start of our project we had the goal of downloading, building, and running the “Martus” codebase. Martus was designed to be an extremely safe data storage service for sensitive personal issues, primarily documenting domestic abuse. We downloaded and tried to build the java project in eclipse, but we were immediately met with incompatibility errors and limited documentation to help us navigate them. After researching some more, we realized that the last updates to the project were almost 4 years ago and that the servers that supported the service were no longer online.

What did that mean for our project? Could we still even work with the code? Technically it was possible for us to work with the code if we had more time and resources to troubleshoot the failed builds. Even if we were to overcome these issues, however, we would have to either dedicate one of our machines to run as a server or all run individual private servers. Running local servers would make it difficult to ensure we were working with the same data set, but running a live server from one of our homes was not a viable option for our group either. We decided that the project had too many hurdles for us to navigate in a short time and decided it would be a better idea to focus our energy on a project with more recent updates, and one that was more optimized for newer hardware and software.

## Exploring STEM (The Spatiotemporal Epidemiological Modeler)

After the difficulty Martus provided, our group was worried moving into The Spatiotemporal Epidemiological Modeler (STEM). Luckily for us, STEM is still being updated regularly and has an in depth guide to getting it set up in Eclipse. This made it far easier to get working in Linux. [A step-by-step guide is provided on the project wiki](#), including download links for necessary installers and detailed instructions for ensuring version comparability and dependencies.

Following these guides, we thankfully had an uneventful process of getting the source code downloaded, installed, and ready to run in Eclipse. There were some build errors in our group, but these were resolved by reinstalling the latest version of Java. We didn't see any tests on the wiki, but there were several demos. Upon running STEM initially we were concerned about another error we received, but interestingly this was part of the design. This was in order to specify and set up initial configurations for the local STEM instance.

Continuing to follow the tutorial, we eventually got STEM built:

Once the code was ready on our devices, we decided to start running some of the tutorials to test out how the user experiences the code. The tutorials gave us a good insight into how the code is supposed to be run and how the input and output looks so we can automate the inputs to send any range of values into the program and harness the outputs to test their accuracy based on the results.

# Chapter 2

## Directory Structure

/TestAutomation

  /project

    /src

      /org/eclipse/stem

        /test

          /driver (**source code for drivers**)

          /analysis (**source code for classes tested**)

          /core (**source code for classes tested**)

          /graphgenerators (**source code for classes tested**)

  /dependencies

    automaticexperiment.jar

    Analysis.jar

    ...

  /scripts

    build.sh

    runAllTests.sh

  /testCasesExecutables

    /org/eclipse/stem

      /test

        /driver (**.class files for driver**)

        /analysis (**.class files for classes tested**)

/core (**.class files for classes tested**)

/graphgenerators (**.class files for classes tested**)

/docs

README.txt

/reports

report.html

## Testing Framework Description

runAllTests.sh, under the directory /scripts, builds the source files the drivers will test and the drivers themselves. Next, runAllTests.sh reads input from the first file (in alphanumerical order) under the folder /testCases, where each test case is in its own file that contains the names of the class and method to test, the name of the driver to run the test, and finally, csv-formatted inputs and an oracle. runAllTests.sh then executes the test case by executing the *java* command on the driver specified, using the inputs and oracle as input to the driver's main method. Once the test has been executed, our drivers all rely on a class called TestReporter that simply prints the results of the test (the output and whether or not it matched the oracle) to System.out. The script then reads this output and prints it to a file called report.html in a format suitable for readability. The above process (excluding the compilation of tested classes and drivers) is repeated for every test case, building the report document incrementally. Once runAllTests.sh opens the report.html in the system's default browser once there are no more test cases to execute. The report document contains the following information for each executed test case:

Test Case XXX - Passed/Failed

Driver used

Component tested (or class tested)

Method tested

Requirement

Test Inputs

Expected Results

Computed Results

## Test Cases

org.eclipse.stem.analysis.automaticexperiment.NelderMeadAlgorithm

Method	Inputs	Oracles
execute()	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, .5, .5, SampleFunction2	2, 1, -4
execute()	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, .1, .1, SampleFunction2	2, 1, -4
execute()	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.2, 1.6, .5, .5, SampleFunction2	2, 1, -4
execute()	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, .5, .5, SampleFunction	3, 2, -7
execute()	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.6, 1.2, .1, .1, SampleFunction	3, 2, -7

org.eclipse.stem.core.model.STEMTime

Method	Inputs	Oracles
addIncrement()	86400000	true
addIncrement()	0	true
addIncrement()	-300	true
addIncrement()	-1000000000	true
addIncrement()	1000000000	true

**org.eclipse.stem.core.math.BinomialDistributionUtil**

Method	Inputs	Oracles
fastPickFromBinomialDist()	.24, 9000	true
fastPickFromBinomialDist()	.3, 0	true
fastPickFromBinomialDist()	.5, 1000000000	true
fastPickFromBinomialDist()	.0, 500000	true
fastPickFromBinomialDist()	1.00, 100000	true

**org.eclipse.stem.graphgenerators.impl.PajekNetGraphGeneratorImplOld**

Method	Inputs	Oracles
getSqrdEdgeRange()	5.253, 3.567, 4, 10, 10, -10, 10, -10, -10, 10, -10	63.9175
getSqrdEdgeRange()	2.658, 2.685, 5, 10, 10, -10, 10, -10, -10, 10, -10, 5, 5	10.8442
getSqrdEdgeRange()	515.256, 125.582, 4, 10, 10, -10, 10, -10, -10, 10, -10	268642.8243
getSqrdEdgeRange()	10, 10, 4, 1, 1, -1, 1, -1, -1, 1, -1	162.0
getSqrdEdgeRange()	-11.65812885, 10, 4, -10, -10, -11, -11, -12, -12, -13, -13	402.7494



`org.eclipse.stem.analysis.impl.ReferenceScenarioDataMap`

Method	Inputs	Oracles
<code>closeEnough(double d1, double d2)</code>	<code>.000001, .000002</code>	<code>FALSE</code>
<code>closeEnough(double d1, double d2)</code>	<code>2.000001, 2.000002</code>	<code>TRUE</code>
<code>closeEnough(double d1, double d2)</code>	<code>2.000003, 2.000004</code>	<code>TRUE</code>
<code>closeEnough(double d1, double d2)</code>	<code>-.000001, -.000002</code>	<code>FALSE</code>
<code>closeEnough(double d1, double d2)</code>	<code>-2.000001, -2.000002</code>	<code>TRUE</code>

## Resource Constraints

STEM will require the user to have an internet connection to install open-jdk-8. The memory required to install our testing framework will be minimal, likely within the range of a few megabytes.

## Dependencies and System Requirements

Our testing framework is only guaranteed to work on Ubuntu 20.04.1 LTS. The framework should also work on other Debian-based Linux distributions, but we do not know for certain. The only dependency needed is open-jdk-8. If open-jdk-8 is not already on your machine, you can run the script “dependencies.sh” which will run “sudo apt-install open-jdk-8.”

## Chapter 3

Overall, the latter half of this project went much more smoothly for us. We were able to entirely automate the process of building and running our test cases. All that was left for us to accomplish at this point was the reporting system. Below is a snapshot of Bash building and running our project along with an example of our test case files:

```

noah@noah-VirtualBox: ~/CS31362_Project/Team-4/TestAutomation
deleted: TestAutomation/scripts/runAllTests_old.sh
deleted: TestAutomation/scripts/test.sh

no changes added to commit (use "git add" and/or "git commit -a")
noah@noah-VirtualBox:~/CS31362_Project/Team-4$ git add .
noah@noah-VirtualBox:~/CS31362_Project/Team-4$ git commit -m "Cleanup for final presentation"
[master 66063ab] Cleanup for final presentation
3 files changed, 1 insertion(+), 238 deletions(-)
delete mode 100755 TestAutomation/scripts/runAllTests_old.sh
delete mode 100644 TestAutomation/scripts/test.sh
noah@noah-VirtualBox:~/CS31362_Project/Team-4$ cd TestAutomation/
noah@noah-VirtualBox:~/CS31362_Project/Team-4/TestAutomation$ ./scripts/runAllTests.sh
./org/eclipse/stem/graphgenerators/Impl/PajekNetGraphGeneratorImplOld.java
./org/eclipse/stem/test/driver/GetSqrEdgeRange/GetSqrEdgeRangeDriver.java
./org/eclipse/stem/test/driver/STEMTimeTest/STEMTimeTest.java
./org/eclipse/stem/test/driver/referencescenariodatanap/PartsPerMllionTestDriver.java
./org/eclipse/stem/test/driver/nelderheadalgorithm/NelderHeadTestDriver.java
./org/eclipse/stem/test/driver/binomDist/binomDist.java
./org/eclipse/stem/test/driver/TestReporter.java
./org/eclipse/stem/core/math/BinomialDistributionUtil.java
./org/eclipse/stem/core/model/Impl/STEMTimeImpl.java
./org/eclipse/stem/analysis/automaticexperiment/NelderHeadAlgorithm.java
./org/eclipse/stem/analysis/Impl/ReferenceScenarioDataMapImpl.java
./org/eclipse/stem/analysis/Impl/ReferenceScenarioDataMapTestingFactory.java
noah@noah-VirtualBox:~/CS31362_Project/Team-4/TestAutomation$
  
```

**001\_nelder.txt**

```

1 001
2 Function minimization
3 org.eclipse.stem.analysis.automaticexperiment.NelderHeadAlgorithm
4 execute()
5 org.eclipse.stem.test.driver.nelderheadalgorithm.NelderHeadTestDriver
6 0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, 0.5, 0.5, SampleFunction2
7 2, 1, -4
  
```

**006\_ppm.txt**

```

1 006
2 Parts per million check
3 org.eclipse.stem.analysis.impl.ReferenceScenarioDataMapImpl
4 closeEnough(double d1, double d2)
5 org.eclipse.stem.test.driver.referencescenariodatanap.PartsPerMllionTestDriver
6 0.000001, 0.000002
7 false
  
```

**010timeTestCase.txt**

```

1 011
2 Non-zero and Non-negative Inputs
3 org.eclipse.stem.core.math.BinomialDistributionUtil
4 fastPickFromBinomialDist()
5 org.eclipse.stem.test.driver.binomDist.binomDist
6 .24, 9000
7 true
  
```

**021\_SqrEdgeRange.txt**

```

1 021
2 Squared edge Range
3 org.eclipse.stem.graphgenerators.PajekNetGraphGeneratorImplOld
4 getSqrEdgeRange(double qx, double qy, Polygon p)
5 org.eclipse.stem.test.driver.GetSqrEdgeRange.GetSqrEdgeRangeDriver
6 5.253, 3.567, 4, 10, 10, -10, 10, -10, -10, 10, -10
7 63.9175
  
```

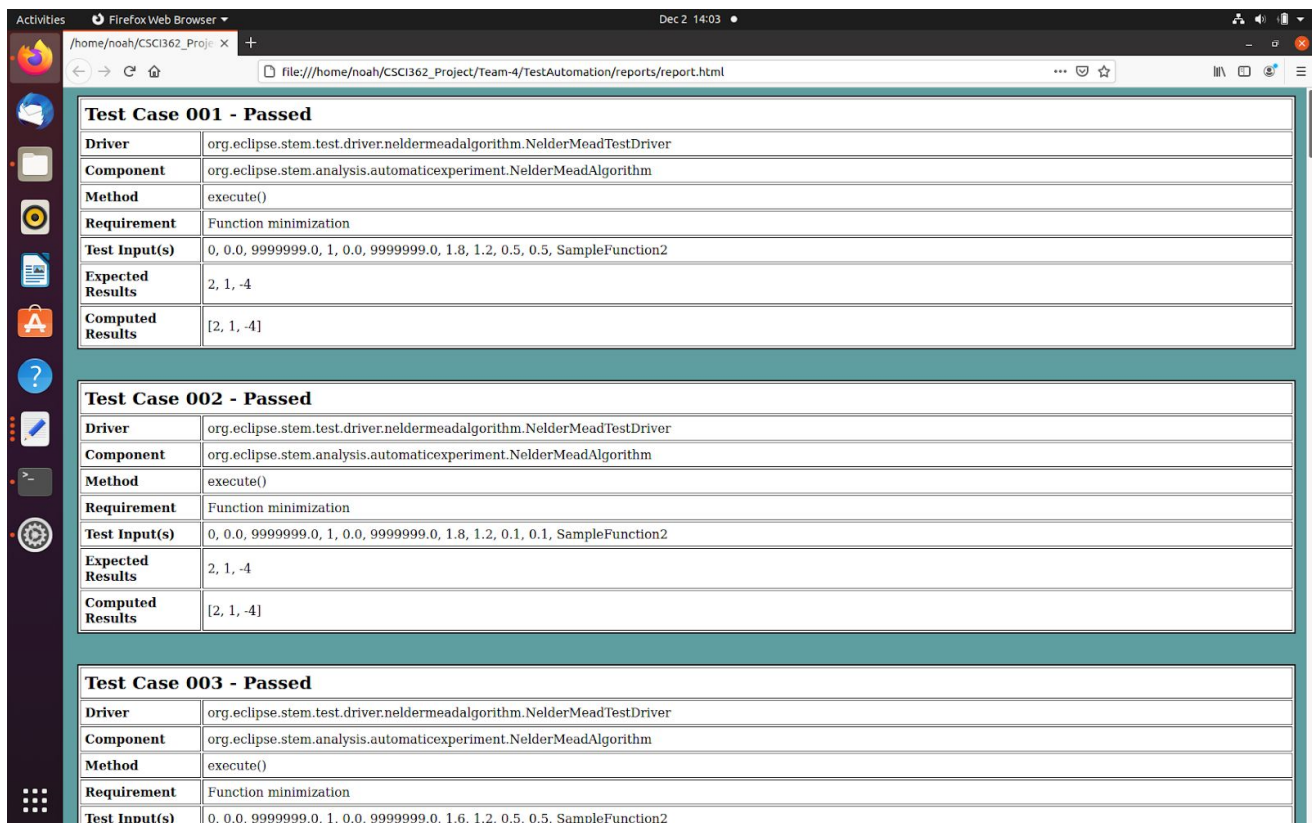
Our test automation framework utilizes a Data-Driven Framework model. A driver was designed for each method that we test, and our testing script reads inputs and other relevant information (including which driver to use) from a test file. This allows each test to be run with multiple data sets just by modifying an appropriate test file and including it in the correct directory. Because test data is not hard coded into the scripts, it is easier to include new test cases for methods after the driver has been implemented.

We worked within a predefined framework directory structure that was provided to us. The root directory, called TestAutomation, held subdirectories for project dependencies, script files, test case files, documents, etc. The objective was for the tests to be run and a report generated when the command “./scripts/runAllTests.sh” was run from the Test Automation top level directory. The script would then access the folder of test case specification files.

The test case specification template provided included the following information, with each piece of information on its own line in a text file: test number/ID, requirement being tested, component being tested, method being tested, test input(s) including command-line argument(s), and expected outcome(s)\*. This information, along with the expected outcome from a separate oracles directory, was compiled to be presented in the test report.

## Chapter 4

Adding on to the previous chapter, we created our HTML report document and merged our build and run scripts into a single script (runAllTests.sh). You can still run build.sh on its own. More importantly, we made it so that both scripts must be run from the top level directory of the project (TestAutomation). We also added new drivers: PartsPerMillionTestDriver, GetSqrdEdgeRangeDriver, STEMtimetest, and binomDist to finish off the rest of our 25 total test cases. Below is a snapshot of the HTML report document:



Test Case 001 - Passed	
Driver	org.eclipse.stem.test.driver.neldermeadalgorithm.NelderMeadTestDriver
Component	org.eclipse.stem.analysis.automaticexperiment.NelderMeadAlgorithm
Method	execute()
Requirement	Function minimization
Test Input(s)	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, 0.5, 0.5, SampleFunction2
Expected Results	2, 1, -4
Computed Results	[2, 1, -4]

Test Case 002 - Passed	
Driver	org.eclipse.stem.test.driver.neldermeadalgorithm.NelderMeadTestDriver
Component	org.eclipse.stem.analysis.automaticexperiment.NelderMeadAlgorithm
Method	execute()
Requirement	Function minimization
Test Input(s)	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, 0.1, 0.1, SampleFunction2
Expected Results	2, 1, -4
Computed Results	[2, 1, -4]

Test Case 003 - Passed	
Driver	org.eclipse.stem.test.driver.neldermeadalgorithm.NelderMeadTestDriver
Component	org.eclipse.stem.analysis.automaticexperiment.NelderMeadAlgorithm
Method	execute()
Requirement	Function minimization
Test Input(s)	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.6, 1.2, 0.5, 0.5, SampleFunction2

## Chapter 5

For this chapter, we injected five faults into the STEM code in order to make some of our test cases fail. To accomplish this, we had to change the way we integrated the STEM source code into our project. We simply deleted every .class file that we are testing for this project in our dependencies, and now we include the .java files for the classes we are testing in our src folder. Our script will now compile the STEM source code we are testing along with our drivers and supporting classes.

### Faults injected

#### **org.eclipse.stem.graphgenerators.impl.PajekNetGraphGeneratorImplOld**

line 1022: Remove "- x" (will cause all test cases to fail)

#### **org.eclipse.stem.analysis.automaticexperiment.NelderMeadAlgorithm**

Line 44: Change 0.5 to 0.1 (will cause test case 004 to fail)

#### **org.eclipse.stem.analysis.impl.ReferenceScenarioDataMapImpl**

Line 911: Change  $\text{Math.abs}(2.0 \cdot (d1 - d2) / (d1 + d2))$  to  $\text{Math.abs}(2.0 \cdot (d1 + d2) / (d1 - d2))$  (will cause test cases 007 and 008 to fail)

#### **org.eclipse.stem.core.math.BinomialDistributionUtil**

Line 71: Change  $p = p/100$  to  $p = p \cdot 100$  (will cause test cases 012 and 014 to fail)

#### **org.eclipse.stem.core.model.impl.STEMTimeImpl**

Line 104: Change final long newTime = newTime().getTime() + timeIncrement; to final long newTime = newTime().getTime() - timeIncrement;(will cause test cases 016, 018, 019, and 020 to fail)

## Chapter 6

Many lessons were learned over the course of this project including: how project dependencies work in a Java ecosystem, how to find and extract dependencies from within a larger project, how to fix issues with these dependencies when they arise, how to organize these dependencies correctly within a new file structure, how to use Bash to create standalone scripts instead of just running one line of code at a time in the terminal, how to feed code from one output source to a shell script to work as an input, how to design and structure an automated testing framework, how to integrate multiple coding languages in this framework, how to approach a large and unfamiliar codebase for the first time, and how to work with a team using version control software and hosting like git and GitHub.

The different pieces of software that were required for this project are nearly all things we have seen before in other classes, things like a Linux terminal, the Bash language, virtual machines and virtual environments, Linux in general, and integrating multiple coding languages into a single project. But no other class has synthesized these concepts in the same way, integrating them all into a large project with implications beyond just our grade in a class. This project allowed us to understand the complexities of professionally finished code, not just by having us analyse it or recreate it but by deconstructing it, working with it, and expanding on some piece of the base functionality. We have learned how different tools have their places within a project and have seen first-hand the merits of failing early and failing often when it comes to new ideas.

Perhaps most importantly, we have learned how to gather and clarify requirements from a customer. By working through information provided from documentation and subsequent meetings, and by watching several other groups go through this process alongside us raising their own questions and concerns for things we may not have encountered yet, we have a fuller understanding of the scope of problems that might arise and what steps we could take to resolve them. Along with this, we have learned the importance of good documentation and documenting while working on a process rather than waiting until the end to hack everything together. This is the value of good communication, which is one of the most necessary aspects of collaboration.