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.

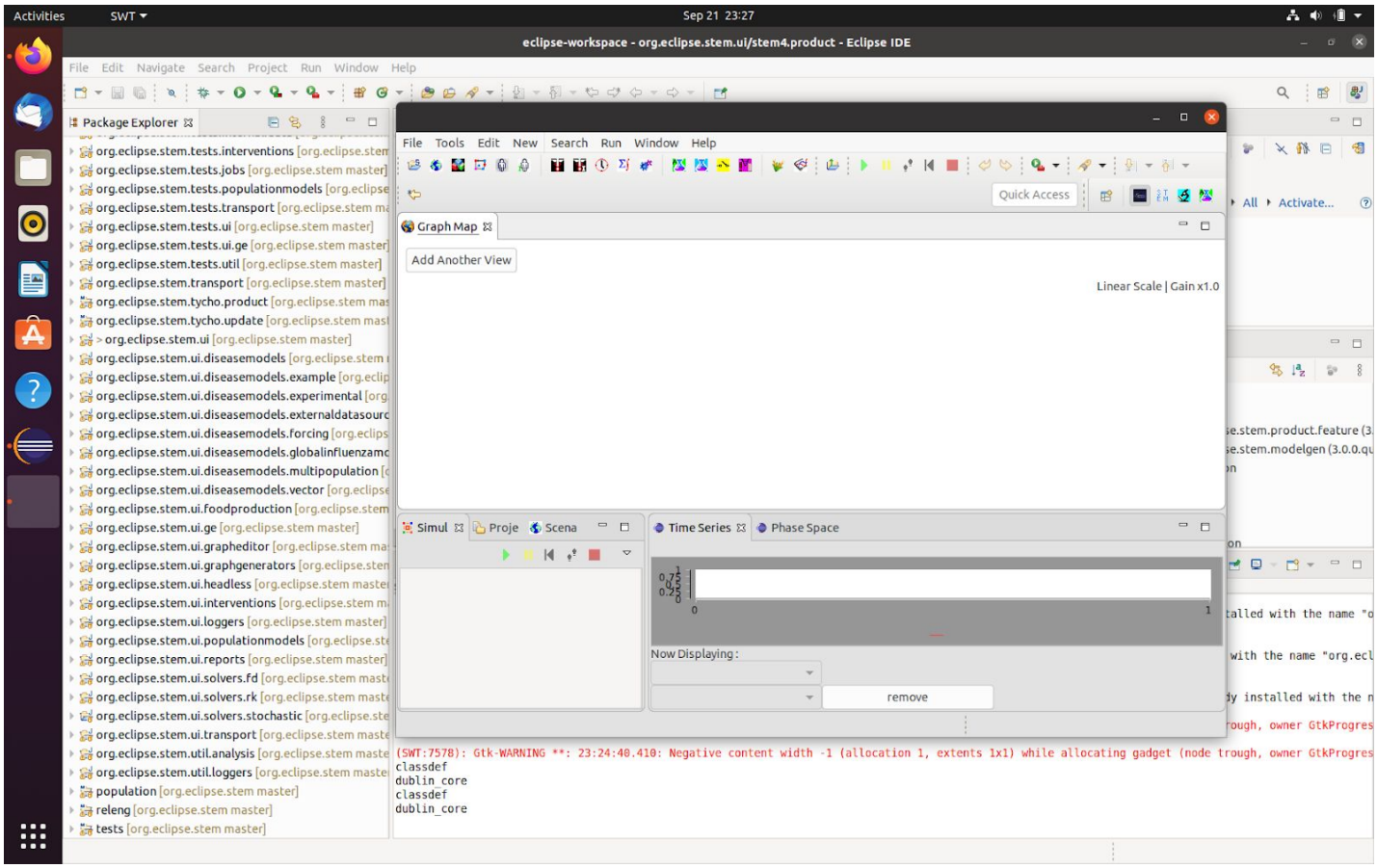
What did that mean for our project? Could we still even work with the code? Technically it was possible for us to work with the code if we had more time and resources to troubleshoot the failed builds. Even if we were to overcome these issues, however, we would have to either dedicate one of our machines 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 to 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: Testing Plan

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
closeEnough(double d1, double d2)	**.000001, .000002**	**FALSE**
closeEnough(double d1, double d2)	**2.000001, 2.000002**	**TRUE**
closeEnough(double d1, double d2)	**2.000003, 2.000004**	**TRUE**
closeEnough(double d1, double d2)	**-.000001, -.000002**	**FALSE**
closeEnough(double d1, double d2)	**-2.000001, -2.000002**	**TRUE**

Resource Constraints

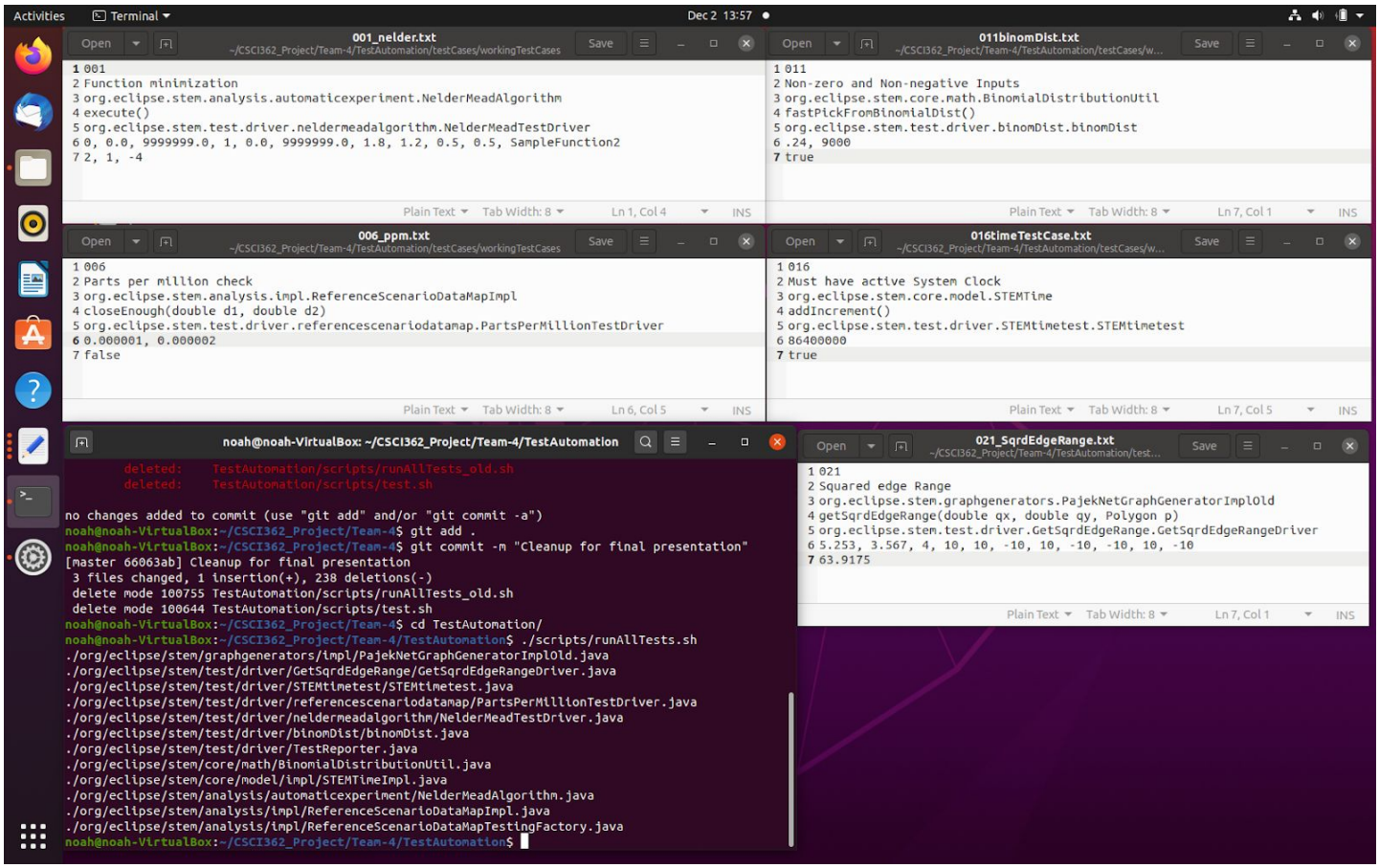
STEM will require the user to have an internet connection to install open-jdk-8 and a tool called “txt2html” which we use to help create our reports. The memory required to install our testing framework will be minimal, likely within the range of a few megabytes.

Dependencies

One of the main issues we are facing running our code in the terminal is the sheer number of dependencies required to actually run many parts of the code. Finding parts of the code that don't have an astronomical amount of dependencies proved interesting as some of the required packages to run code were needed for code that was extended several times. This made actually finding all of the required dependencies/packages the most frustrating part of trying to get some pieces of code to work. To avoid this issue we are going to have to be more selective in how we choose code to make into unit tests. The code we actually select must have limited dependencies to streamline that aspect of the test creation and allow us to test the code more thoroughly.

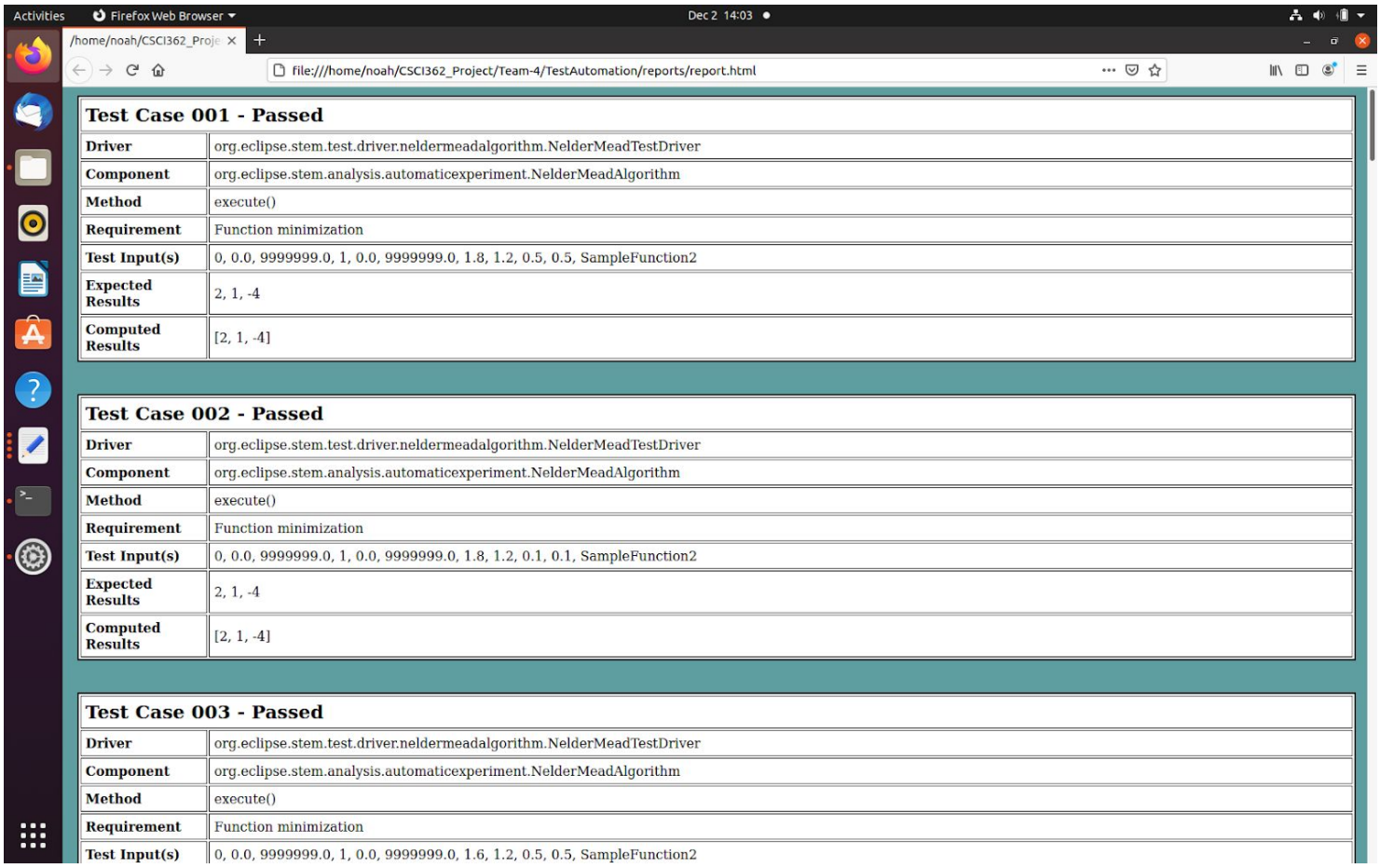
Chapter 3

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:



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 to 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 Math.abs(2.0*(d1-d2)/(d1+d2)) to Math.abs(2.0*(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*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

Ryan Ratliffe's experiences during the project:

> There were several main things I learned while working on this project: how dependencies are handled in the java ecosystem, how to fix any errors in these dependencies, how to feed outputs of code to shell scripts, how to find the exact files any program depends on to run, and how to organize them in the right format to be usable. This project was very enjoyable for me because I got to learn how professionally finished code works and how to use the linux command line to execute the code I want on the computer. I am way more comfortable with virtual machines, command line testing, and linux in general as a result of the work I have done. This project has been a fun foray into formal codebases for me to learn with.

Noah Drake's experiences during the project:

> This project mainly taught me four things: how dependencies in Java work, how to use Bash to automate a testing framework for a software project, how to work with a large codebase I haven’t seen before, and finally, how to work in a team on a software project using version control. More generally, it has given me greater confidence in my ability to program. Before I started this project, I had far less faith in my ability to code and make something that can actually be used by other people, not just run on my own machine. Now, I feel like I have come a long way and have better faith in my ability to learn how to make something work on any machine...

Tristan (dunno your last name sry)'s during the project:

Chloe (dunno your last name sry)'s during the project: