Test Plan

The testing process

Test-driven development will be used with continuous integration so unit tests will be conducted throughout.

We will be designing an automated testing framework in order to more efficiently conduct unit tests.

Component testing will take place as specified stages of the timeline.

System testing will be conducted at the end of each development stage.

Acceptance testing will be conducted before each release.

Requirements traceability

At each iteration of the conducted component or system tests, both the results of the system or component tests and a copy of the unit tests results will be recorded at that time period.

Tested items

All individual units, components, and the overall system are going to be tested. This will include the core math logic used, the integration of each of these parts, as well as the user interface.

Testing schedule

Unit testing will be done at all stages of development, and further time will be allotted for component testing, system testing and acceptance testing. 2 weeks of time will be allotted for each component to be tested and 2 months time will be allotted for system testing. Acceptance testing will be given 3 months time.

Test recording procedures

Each time system or component testing is conducted, the results will be stored in data files showing the requirement being tested and satisfied, the inputs, and the resulting output. Any other data necessary to document the software and prove its trustworthiness will be collected as well.

Hardware and software requirements

Eclipse and a current version of Java are required. At least 4 gigabytes of RAM are recommended as well as a dual core processor with at least 1.8 ghz. A monitor is also required as well as 10 gigabytes of free disk space.

Constraints

This is an open-source project, so developers are probably not going to be able to devote themselves to STEM full-time. Our limited budget could also affect the testing process.

System tests

Jerry is an epidemiological scientist for the World Health Organization. Part of his job is analyzing and containing outbreaks of infectious diseases. In order to visually model the spread of disease, Jerry decides to use the STEM tool to visualize models that he created. In addition, a few of his coworkers sent him copies of the models they had created and he wants to plug those in to his application and visualize them for himself. Jerry will then use these models to make educated decisions on how to control the spread of the disease and how to proceed.

We will use this scenario and others to derive system tests.

GCD

Requirement being tested: the ability to compute the greatest common divisor
Component: Math Operations
Function: gcd
Input: Two integers
Output: The greatest common divisor of those 2 integers

Test cases for GCD:

Test case #1:
1. Test Number: 1
2. Requirement being tested: The ability to calculate the greatest common divisor between two positive integers where the first is greater than the second.
3. Component being tested: Math Operations
4. Method being tested: gcd
5. Test Inputs: 30, 12
6. Expected Outcome: 6

Test case #2:
1. Test number: 2
2. Requirement being tested: The ability to calculate the greatest common divisor between two identical integers.
3. Component being tested: Math Operations
4. Method being tested: gcd
5. Test Inputs: 20, 20
6. Expected Outcome: 20

Test case #3:
1. Test number: 3
2. Requirement being tested: The ability to calculate the greatest common divisor between 0 and another integer.
3. Component being tested: Math Operations
4. Method being tested: gcd
5. Test inputs: 0, 50
6. Expected outcome: undefined

Test case #4:
1. Test number: 4
2. Requirement being tested: The ability to calculate the greatest common divisor in an edge case scenario that verifies correctness in the most extreme of cases
3. Component being tested: Math Operations
4. Method being tested: gcd
5. Test inputs: 2147483647, 5
6. Expected outcome: 1

Test case #5
1. Test number: 5
2. Requirement being tested: the ability to calculate the greatest common divisor between a negative integer and a positive integer.
3. Component being tested: Math Operations
4. Method being tested: gcd
5. Test inputs: -10, 17
6. Expected Outcome: 1

LCM

Requirement being tested: the ability to compute the lowest common multiple
Component: Math Operations
Function: lcm
Input: two integers
Output: The lowest common multiple of the two input integers

ArgMin

Requirement being tested: the ability to find the index of the smallest number in an array
Component: Math Operations
Function: argMin
Input: an array of doubles, an array of the indices of the elements in the array of doubles
Output: the index of the smallest element in the array of doubles

ArgMax

Requirement being tested: the ability to find the index of the largest number in an array
Component: Math Operations
Function: argMax
Input: an array of doubles, an array of the indices of the elements in the array of doubles
Output: the index of the largest element in the array of doubles

Sum Logs

Requirement being tested: the ability to add logarithmic values
Component: Math Operations
Function: sumLogs
Input: an array of doubles (logs)
Output: the sum of the logs

Report on our Experiences

We found a class in our open-source project called MathOps.java that contains the 5 methods we are going to test using our automated framework. The methods are testable with simple numeric inputs, which is why we chose them. 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 denominator) 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. We have also included a case in which the output should be undefined. 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.