CSCI 362-02
Fall 2015
Final Report

Sugar Labs: Calculate Activity

Team Unknown:
Aaron Gaynor, Jake Hennessy, Aaron Walton, Sabrina Yongue

Table of Contents

Chapter 1: Introduction

What is Sugar Labs?

Sugar Labs is an open-sourced software development community that utilizes the Sugar platform. The focus is on making Sugar readily available and accessible, and Sugar Labs is used as a learning tool. It is a FOSS (Free Open Source Software); it is free to use by anyone. Sugar is an accumulation of Activities, or learning applications. Potential users are encouraged to explore Sugar and develop new activities or modify existing ones.

The Sugar Activity we selected is the Calculate Activity. It is a calculator application that allows for various standard calculator functions, from factorial to sine and cosine.

Choosing Our Project

We had some difficulties from the outset while deciding on our project. We selected Sugar Labs initially, as it has a very welcoming community and welcomes newcomers with open arms. It has very thorough documentation, and is an active project, so if we ran into any problems, we could definitely find information to assist us. However, we had difficulties with downloading the Sugar platform and the accompanying activities. We ran into problems with trying to run the main driving file for Calculate for the command line. After much work and failed attempts, we elected to switch projects. We believed that as it was at the start of the semester, instead of wasting time foolishly on a dead end, switching projects made the most sense.

We defaulted back to another option we had considered, called pybliographer. It is an application that is used to access bibliographic databases for ease in citations and other referencing. It seemed like it would be straightforward to test, and as an added bonus it is coded entirely in python, which we are all at least somewhat familiar with. We downloaded the necessary files and installed it successfully, and were able to run it from our Linux command line. Success! Or so we thought. From the outside, it appeared that it ran when prompted. However, a closer inspection would reveal that it was in fact not actually working. When queried, an error would appear. The driving file would seek out a file with a particular absolute path that had not been installed during the download. Much research later, and we discovered that the file it needed was a dead end. It was outdated; we realized that the project itself was essentially outdated as well. No new updates or progress had been made for quite some time. So we were stuck with no clear way to resolve our problem, and no community to contact. After what can be chalked up to a lot of wasted time.

With a lot less bounce in our stride, we went back to attempt Sugar again. We could still not figure out how to access the Activity files we needed in order to run from the command line, until suddenly after much work, a breakthrough happened. We discovered that the file that contained the actual basis of the Activity, all its math capabilities, could be run without dependency on the Sugar platform. It's just a standard python file, functions.py, that contains all of the functions that the Calculate Activity is capable of. By figuring out that we could isolate it and test it separately, we were finally on the path of being able to construct our testing framework.

Chapter 2: Test Planning

Developing a Test Plan

Our next goal was to develop a test plan to later implement for our eventual testing framework. We needed to select which methods we were going to test, and develop a set of at least five prospective test cases for use to use as a guide later on for the next deliverable.

We decided to test five different methods inside of the functions.py file, and we developed ten different test cases.

Methods to Test

1. def add (x , y)
   Takes in two values and returns the result of the addition

2. def sub (x , y)
   Takes in two values and returns the result of the subtraction of the second value from the first

3. def mul (x , y)
   Takes in two values and returns the result of multiplying the first value and the second

4. def div (x , y)
   Takes in two values and and returns the result of dividing the first value by the second

5. def factorial (x)
   Takes a single value and returns the factorial of it

Initial Test Cases

TestId: 1
Requirement: adds two positive integers
Component: Functions
Method: add

Input: 10 10
Expected Outcome: 20

TestId: 2
Requirement: adds a positive integer and zero
Component: Functions
Method: add
Input: 10 0
Expected Outcome: 10

TestId: 3
Requirement: adds a negative and positive integer
Component: Functions
Method: add
Input: 10 -10
Expected Outcome: 0

TestId: 4
Requirement: adds a negative integer and zero
Component: Functions
Method: add
Input: -10 0
Expected Outcome: -10

TestId: 5
Requirement: adds two negative integers
Component: Functions
Method: add
Input: -10 -10
Expected Outcome: -20

TestId: 6
Requirement: subtracts positive integers
Component: Functions
Method: sub
Input: 10 5
Expected Outcome: 5

TestId: 7
Requirement: subtracts a positive integer and zero
Component: Functions
Method: sub
Input: 10 0
Expected Outcome: 10

TestId: 8
Requirement: subtracts zero and a positive integer
Component: Functions
Method: sub
Input: 0 10
Expected Outcome: -10

TestId: 9
Requirement: subtracts a positive and negative integer
Component: Functions
Method: sub
Input: 10 -10
Expected Outcome: 20

TestId: 10
Requirement: subtracts negative integers
Component: Functions
Method: sub
Input: -10 -10
Expected Outcome: 0

Test Plan

Testing Schedule: Develop test cases for the remaining methods and verify the initial test cases. As the methods to be tested are very straightforward, they are simple to check and no further action is required.

Software Requirements: Linux/Ubuntu, the latest version of the sugar-labs calculate activity,

specifically functions.py file, as well as bash scripting capability.

Hardware Requirements: A device that is is capable of utilizing linux/ubuntu.

Time constraints: Complete all test cases by November 24th to meet the deliverable deadline.

Requirements Traceability: Test script for each function, and output files for each script.

Chapter 3: Testing Framework

Building the Architecture:

After brainstorming about how to construct our testing framework, we decided that creating a text handler to deal with the information inside the individual .txt test files would streamline the process a bit. Though we would have to pipeline the information stream more times than if we did not use a text handler type scenario, having a file with a specific function to handle the same process multiple times makes sense from a logical coding standpoint. We were unable to create a text handler in time for the third deliverable deadline, but we continued working to complete it.

Testing Framework Architecture:

We have a "master" script file that initializes the test case handler .py files. The master script first finds the test case text files and figures out from them which test case handler files to run. It then hands them off to a text handler which draws the inputs from the text files and returns them to the master script. The master script then hands the inputs to the designated test case handler to run. Using the input, the test handler executes, and returns the output to the master script which hands the output to a text file creator which creates the output files. There will be a total of 5 test case handlers, and each handler will be called at will depending on which method is determined in the text file. The text handler will be called by every test case handler, in order to streamline the process. After all of the test cases have been executed, the master script runs an html file builder which takes the output file and compares the information back to the oracle contained in the test case text file and determines if the test passed or failed. After iterating through all the test cases, the html file builder returns the name of the file and the master script opens the html file, ending it's execution.

Framework Structure

```
/TeamUnknown
    /Test Automation
        /docs
            README.txt
        /oracles
        /project
            _init_.py
            /src
                functions.py
                rational.py
        /reports
        /scripts
            TestCaseReader.py
            finalReportCreator.py
            outfile.py
            runAllTests.py
            validator.py
        /temp
        /testCases
            testCase1.txt
            :
            :
            testCase25.txt
        /TestCasesExecutables
            TestCaseAdd.py
            TestCaseSub.py
            TestCaseMul.py
            TestCaseDiv.py
            TestCaseFactorial.py
```

How to use:

Have the latest version of ubuntu/linux.

Download our file testing directory, and run it from the bash command line terminal by navigating to the scripts folder and executing the command ‘python runAllTests.py’.

Chapter 4: Test Cases

Running the Framework

For this deliverable, our aim was to make our framework actually functional so that it could automatically run through 25 test cases. We had a very clear idea of how we wanted to implement it, but actually getting everything to run properly took some time and effort. As we were far behind from the previous deliverable, we had to work doubly as hard to try to complete deliverable four on time. Not having a completed framework made reaching the requirements for deliverable four much more difficult. We successfully managed to coordinate a meeting time for all of us that we could all attend, and started working.

One of the roadblocks we experienced was figuring out how to go about using absolute paths in python. No one in our group had had any experience with this, so there was a steep learning curve. We already had a lot of ground to make up, and the added time needed to research how to tackle our problem made the overall process that much longer to complete. As we progressed through trial and error, minor changes had to be made here and there, but for the most part, the overall design plan remained untouched. We successfully created a text handler, and from there we caught on to some momentum to push us forward.

By the time of the deliverable deadline, our framework is still not quite where it should have been; instead of automatically outputting the html file to a web browser, it only created it. But we thought it to be an issue not difficult to change between then and the next deliverable. Overall, we managed to accomplish a great deal of work and progressed very far along with the final project.

Creating test cases was fairly simple, as the methods we test are simply mathematics. Some consideration had to be placed as to how best to thoroughly test what should be simple math. After we spent so much time developing the framework, it was a relief to have something far simpler to work on.

List of Test Cases

TestId: 1
Requirement: adds two positive integers
Component: Functions
Method: add
Input: 10 10
Expected Outcome: 20

TestId: 2
Requirement: adds a positive integer and zero
Component: Functions
Method: add
Input: 10 0
Expected Outcome: 10

TestId: 3
Requirement: adds a negative and positive integer
Component: Functions
Method: add
Input: 10 -10
Expected Outcome: 0

TestId: 4
Requirement: adds a negative integer and zero
Component: Functions
Method: add
Input: -10 0
Expected Outcome: -10

TestId: 5
Requirement: adds two negative integers
Component: Functions
Method: add
Input: -10 -10
Expected Outcome: -20

TestId: 6
Requirement: subtracts positive integers
Component: Functions
Method: sub
Input: 10 5
Expected Outcome: 5

TestId: 7
Requirement: subtracts a positive integer and zero
Component: Functions
Method: sub
Input: 10 0
Expected Outcome: 10

TestId: 8
Requirement: subtracts zero and a positive integer
Component: Functions
Method: sub
Input: 0 10
Expected Outcome: -10

TestId: 9
Requirement: subtracts a positive and negative integer
Component: Functions
Method: sub
Input: 10 -10
Expected Outcome: 20

TestId: 10
Requirement: subtracts negative integers
Component: Functions
Method: sub
Input: -10 -10

Expected Outcome: 0

TestId: 11
Requirement: multiplies positive integers
Component: Functions
Method: mul
Input: 10 10
Expected Outcome: 100

TestId: 12
Requirement: multiplies a positive and negative integer
Component: Functions
Method: mul
Input: 10 -10
Expected Outcome: -100

TestId: 13
Requirement: multiplies negative integers
Component: Functions
Method: mul
Input: -10 -10
Expected Outcome: 100

TestId: 14
Requirement: multiplies a negative integer and zero
Component: Functions
Method: mul
Input: -10 0
Expected Outcome: 0

TestId: 15

Requirement: multiplies a positive integer and zero
Component: Functions
Method: mul
Input: 10 0
Expected Outcome: 0
TestId: 16
Requirement: divides positive integers
Component: Functions
Method: div
Input: 10 10
Expected Outcome: 1

TestId: 17
Requirement: divides positive integers
Component: Functions
Method: div
Input: 10 1
Expected Outcome: 10

TestId: 18
Requirement: divides positive integer and zero
Component: Functions
Method: div
Input: 10 0
Expected Outcome: ERROR

TestId: 19
Requirement: divides positive integers
Component: Functions
Method: div
Input: 10 1
Expected Outcome: 10

TestId: 20
Requirement: divides a positive and negative integer
Component: Functions
Method: div
Input: 10 -10
Expected Outcome: -1

TestId: 21
Requirement: calculates factorial of positive integer
Component: Functions
Method: factorial
Input: 1
Expected Outcome: 1

TestId: 22
Requirement: calculates factorial of positive integer
Component: Functions
Method: factorial
Input: 8
Expected Outcome: 40320

TestId: 23
Requirement: calculates factorial of positive integer
Component: Functions
Method: factorial
Input: 3
Expected Outcome: 6

TestId: 24
Requirement: calculates factorial of zero
Component: Functions
Method: factorial
Input: 0
Expected Outcome: 1

TestId: 25
Requirement: calculates factorial of negative integer
Component: Functions
Method: factorial
Input: -3
Expected Outcome: -6

Chapter 5: Fault Injections

Selecting Faults:

For this deliverable, we were supposed to implement 5 complications that would cause at least 5 of our test cases to fail. The following are the errors we created and how to recreate them. All these changes were made directly to the functions.py file that we are testing, that is located inside of of our TestAutomation/project/src folder.

1. In functions.py, change the return of sub(x , y) such that it now return sub(y , x). This will cause all test cases to fail, with the exception of those that have input values that are equal.

2. In functions.py, change the return of divide(x , y) such that it now returns divide(x , x). As one would expect, this now causes the results returned to be equal to 1, which will cause some test cases to fail by producing incorrect results for those cases where the two input values are not equal to each other.

3. In functions.py, modify the add(x , y) function such that it now only returns the absolute value of the result of the addition. This will cause any test cases with an expected end result other than zero to fail, as the sign of the result will be the opposite of what it should be.

4. In functions.py, change the call to factorial(x) so that while calculating the factorial it loops through while multiplying until the total is multiplied by 0. As a result, all tests should fail, as zero is not an acceptable result for any input.

5. In functions.py, change the call to mul(x , y) so that it now calculates the multiplied value of x and the absolute value of y. This will cause test cases that have a negative value as the second input to fail, as the result will have the opposite sign than is expected.

Experiences:

From our last deliverable, we discovered that our framework had failed to meet the necessary criteria, as it could only accommodate 25 test cases, and did not use the test case files to determine which function to call. In addition, we had to of course, come up with 5 different errors to place within our framework. It did take some time to figure out what we could do to only fail some test cases, but not all. We implemented the necessary changes, and hopefully we are

closer to having our framework be correct according to the project specifications and in class correc

Chapter 6: Experiences and Reflections

Aaron Gaynor

The largest part of this experience was learning to work in a group setting to apply knowledge and skills in a practical environment. The project itself was set up in such a way that the design and implementation was completely up to us as a group to decide upon, and while we made some mistakes along the way due to ambiguous requirements documents, we bounced back well and managed to teach each other and achieve our goals efficiently.

Jacob Hennessey

The thing I learned most during this group project was the same as all of my other group members, how to work efficiently in a group setting. Scheduling was a major issue when it comes to getting working college students together. Partitioning work to group members so that we can get it done fast and correct was another hurdle. I learned a lot about file pathing, piping, and how to navigate file systems efficiently. While a large portion of our code was in python, it was also valuable to learn about bash and how to use the command line to execute programs, and explore file systems. Overall I didn't learn too much in the actual way of coding, but I did learn a lot about creating a testing framework and the parts/requirements needed to make it work.

Aaron Walton

I think the biggest takeaway from this project is how to work efficiently as a group and learned a lot about the problems that arise when you have to work on a big project with three other people. First off, scheduling a meeting once a week with three other people on totally different schedules is one of the hardest things. The second hardest thing was figuring out what

we need to do as a group, and then splitting up the workload based on people's strengths and weaknesses. The most important coding aspect I learned was more to do with pathing and navigating a file system with code. Also getting multiple pieces of the testing framework to work together and figuring out how they can talk to each other was a thing I'm glad I got to experience, although I wouldn't really call it learning. Reading from files, parsing text files, outputting an HTML, and most of the other stuff were taught a while ago, so not much was learned on that front. I think the takeaway from this project is how to work in a team, convey your ideas to others, and how to work efficiently with others. Although even that was a stretch, because unless you're working on multiple projects with multiple people, you wouldn't have to work around others that are trying to take other classes, do other homework, and working jobs outside of the project.

Sabrina Yongue

For me, this whole project was just this giant thing that was looming over us from the start of the semester. I have little to no practical coding experience; I am still currently taking classes where I am essentially spoon fed which methods to write and given a clear-cut pseudocode to follow. To say that I have been overwhelmed is about as much of an understatement as one could make. Pair my lack of knowledge with my bad track record of group work in the past, and you can guess at how nervous I was from the beginning.

I can say with certainty that I still have a ways to go before I will ever be able to boast that I am a "coder" or an "engineer," but this project certainly forced me to learn a lot over the course of this semester.

The main problem we struggled with was scheduling. In a real world scenario, working on a project in a company, everyone would constantly available at the same time, and there would be no conflicts. The same cannot be said of those of us attending college. But scheduling aside, we all worked well together, and I can say that this is the first group project I've ever participated in where everyone got along. It has been a difficult experience, but I have learned a

lot from it, and I hope that any future group projects I have to tackle will involve group members as great as those I have gotten to work with for Team Unknown.

Appendix: Suggestions

Suggestions:

We think clarity of what's expected and more in-class discussion about the project would be the most helpful. If you were to show a previous working framework from a previous class, we think that would be really helpful in terms of what they had to create to get it to work, and what the final result is supposed to look like. The expected outcomes for each deliverable and presentation along the way were ambiguous and were easy to misinterpret, especially for those of us with very little experience. An example is the last class period; a lot of teams got grilled for not injecting faults into the source code and for asking a question on where to find the guidelines for the poster. The wording for the injection was ambiguous, and you said we talked about the poster when we didn't, and there is no obvious way to find the poster resources on the class page. You've taught this class many times, but it's our first time, so unless it's specifically posted in detail on the class page, or discussed in class, we have no real idea what to expect. I know we can ask questions in class, but it's hard to ask questions when you have no idea what to expect and you think you're on the right track, but you're actually doing it completely wrong. Just as a side note, since the main focus in this class was mainly the project, I think the blogs were an inconvenience that distracted us from the project. When you have this project and other things looming over your head, it's easy to fall behind on blogs or rush them because you have bigger fish to fry.