TEAM SILVER BULLETS

SugarLabs Testing Analysis and Framework

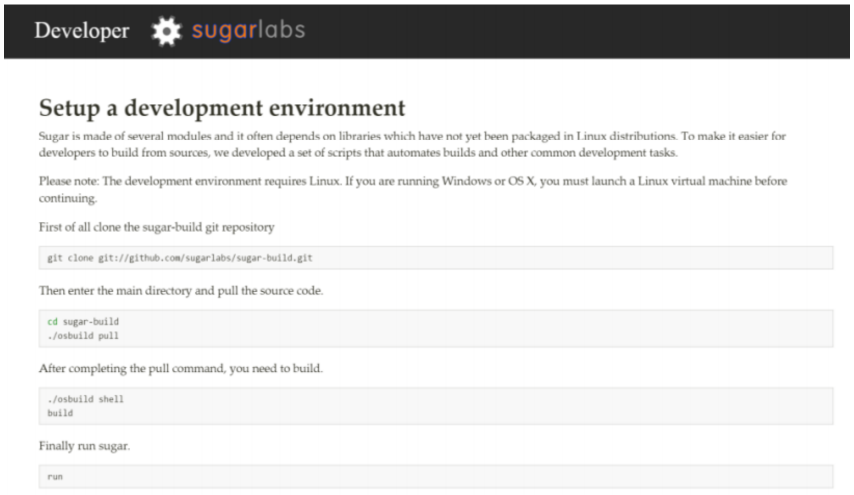
Calculator Activity

Jameson Burroughs, Christopher Lewis, Megan O'Neal, Alexander Wray

This is a detailed outline of our testing framework built to test the Calculator activity within SugarLabs. It follows our team's progress throughout the Fall 2015 semester, as reported at the time.

Chapter 1 - Deliverable #1, dated 15 September 2015

Team Silver Bullet consists of Jameson Burroughs, Christopher Lewis, Megan O'Neal, and Alex Wray. We chose Sugar Labs [wiki.sugarlabs.org] as our open source project to compile, test, and document. In this deliverable, we have successfully cloned our project to our Ubuntu virtual machines. There were a few steps in completing this task.



Developer sugarlabs

Setup a development environment

Sugar is made of several modules and it often depends on libraries which have not yet been packaged in Linux distributions. To make it easier for developers to build from sources, we developed a set of scripts that automates builds and other common development tasks.

Please note: The development environment requires Linux. If you are running Windows or OS X, you must launch a Linux virtual machine before continuing.

First of all clone the sugar-build git repository

```
git clone git://github.com/sugarlabs/sugar-build.git
```

Then enter the main directory and pull the source code.

```
cd sugar-build
./osbuild pull
```

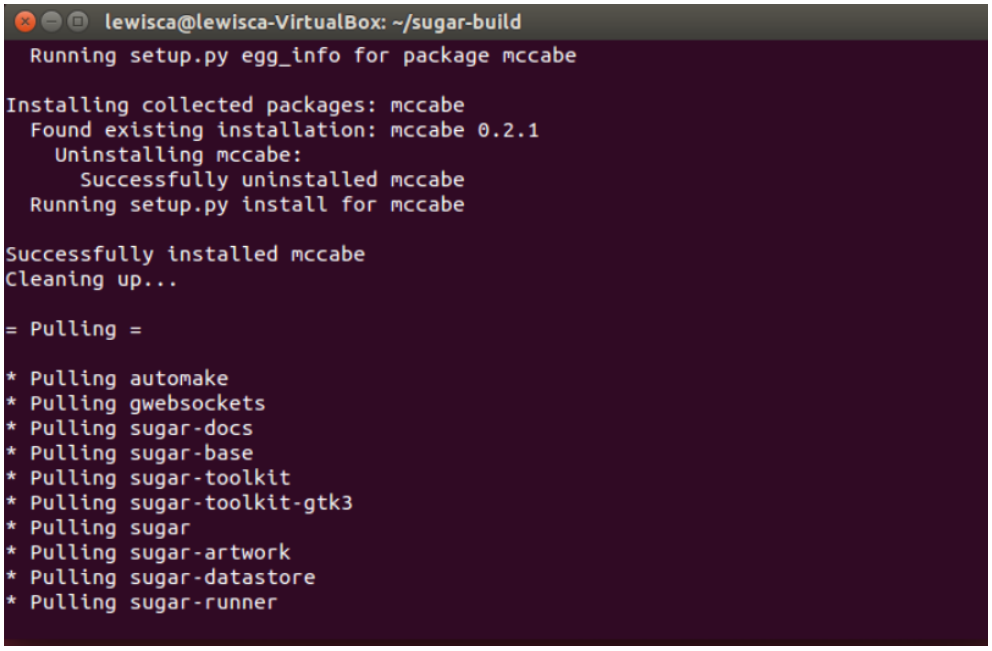
After completing the pull command, you need to build.

```
./osbuild shell
build
```

Finally run sugar.

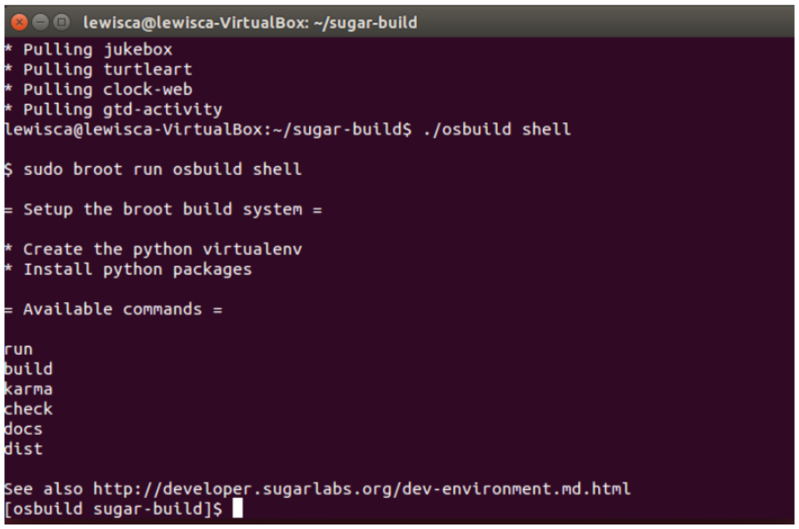
```
run
```

The wiki provided for developers by Sugar Labs provided us with the following steps.



```
lewisca@lewisca-VirtualBox: ~/sugar-build
 Running setup.py egg_info for package mccabe

Installing collected packages: mccabe
  Found existing installation: mccabe 0.2.1
    Uninstalling mccabe:
      Successfully uninstalled mccabe
  Running setup.py install for mccabe

Successfully installed mccabe
Cleaning up...

= Pulling =

* Pulling automake
* Pulling gwebsockets
* Pulling sugar-docs
* Pulling sugar-base
* Pulling sugar-toolkit
* Pulling sugar-toolkit-gtk3
* Pulling sugar
* Pulling sugar-artwork
* Pulling sugar-datastore
* Pulling sugar-runner
```

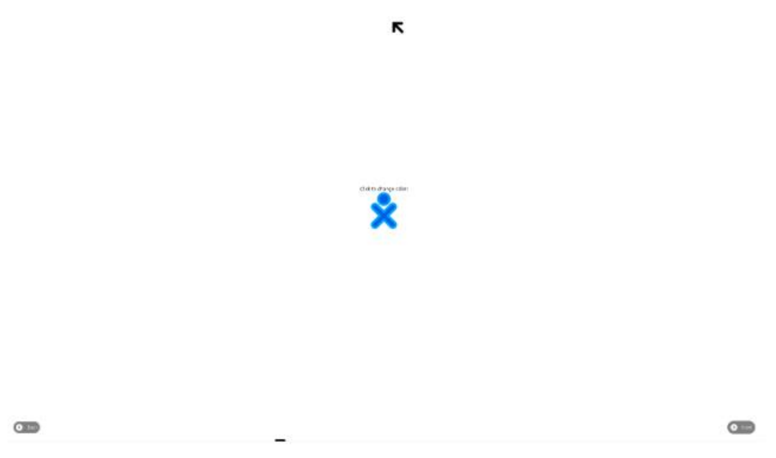
After running './osbuild pull', the project proceeded to download.

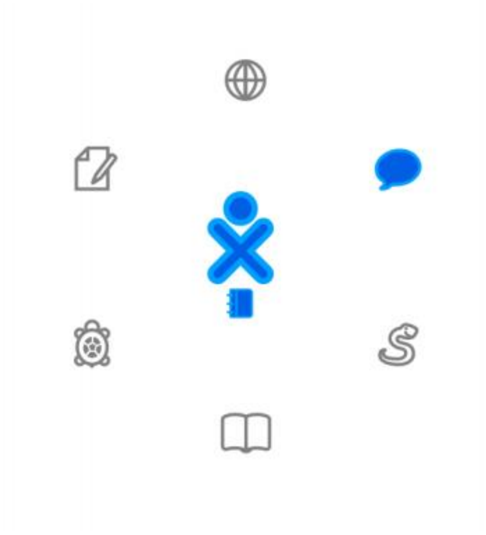


```
lewisca@lewisca-VirtualBox: ~/sugar-build
* Pulling jukebox
* Pulling turtleart
* Pulling clock-web
* Pulling gtd-activity
lewisca@lewisca-VirtualBox:~/sugar-build$ ./osbuild shell

$ sudo broot run osbuild shell

= Setup the broot build system =

* Create the python virtualenv
* Install python packages

= Available commands =

run
build
karma
check
docs
dist

See also http://developer.sugarlabs.org/dev-environment.md.html
[osbuild sugar-build]$ 
```

After the downloads completed, we built the shell.

We ran the project from sugar's shell, and were greeted with the completed, running introduction page to Sugar Labs! The main menu was not too much farther past this.





Our experiences so far have been rather pleasant. Once we figured out how to clone the code to our Ubuntu machines, compiling the code and running it was quite simple. From the main menu, pictured above and to the left, we have, starting at the 12 o'clock spot and going clockwise, we have: web, chat, Pippy (a python editor), read, TurtleBlocks (a visual drag and drop style programming teaching tool), and write. Each one works as expected, and each is quite testable with its limitations. There is a more fleshed out version of Sugar Labs on their website in the form of an .iso file, which is undesirable for our team's purpose. We plan on working with the program we have downloaded as-is. There are more than enough possible test cases to analyze. We found existing test files and ran them in the osbuild shell of sugar. We currently do not know exactly what they do or how to interpret the results. We do know that the python test file we have run multiple times tests the applications within Sugar Labs, so it's a good framework to start out with. We have also yet to select an activity within Sugar Labs to test yet, however our team will accomplish this task quite soon.

Chapter 2 - Deliverable #2, dated 26 September 2015

Introduction

This Test Plan will help communicate and document testing plans to team members. It includes objectives, test cases, test strategy, Hardware/software requirements, testing schedule and constraints.

Objective of Test plan is to test the sugar activity calculator on SugarLabs. We will test for a variety of valid and invalid inputs. Since we are testing with a small program, there will be no need to break this into the testing of individual sub-systems. Methods within the class will be targeted.

Process for testing :

1. Outline possible correct and incorrect inputs
2. Define them, and pass them to the program
3. Compare actual result to expected result
4. Documentation of test cases

Test Strategy

We will mainly use unit testing to test individual functions. Exploratory testing will play a large part in testing methods as the team is still new to the calculator activity and will learn as we go.

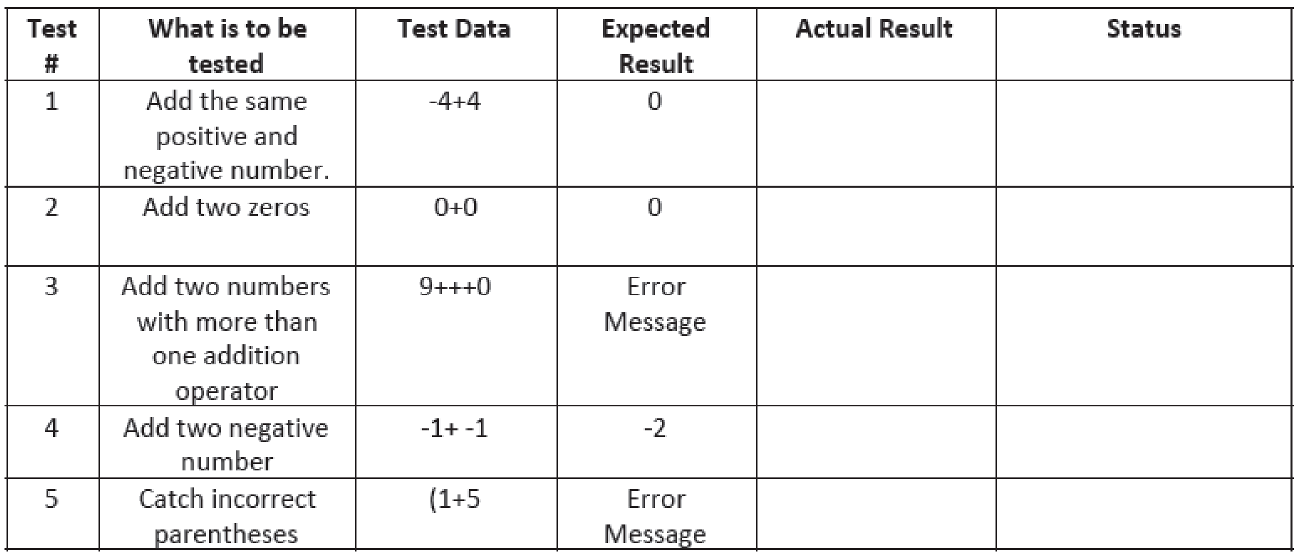
Functions to test -

Add function
Div function
Factorial function
Mul function
Mod function

Test recording procedures-

Test cases for methods will all recorded in the Test Case Documentations.

Silver Bullets
Test Case Documentation
Test Case ID: 1, 2, 3, 4, 5
Test Designed by: Silver Bullests
Test Priority (Low/Medium/High): High
Test Designed date: 09/26/2015
Module Name: Add Function (1-4), Parentheses (5)
Test Executed by: SilverBullets
Test Title: Verify Add function is properly working, verify parentheses catches incomplete pairs
Test Execution date: TBD
Description: Test Add Function, Test Incomplete Parentheses
Pre-conditions: None



Test #	What is to be tested	Test Data	Expected Result	Actual Result	Status
1	Add the same positive and negative number.	-4+4	0		
2	Add two zeros	0+0	0		
3	Add two numbers with more than one addition operator	9+++0	Error Message		
4	Add two negative number	-1+ -1	-2		
5	Catch incorrect parentheses	(1+5	Error Message		

(Note: These were not test cases that we ended up using, as they were not designed with the parameters of the methods in mind.)

Hardware and software requirements

Testing will require Linux and a functional version SugarLabs with the calculator activity.

Constraints

Constraints include time, availability, and unknown variables that may occur.

Chapter 3 - Deliverable #3, dated 27 October 2015

Our current framework is written fully in python. It loops through each of the test case files to retrieve the test inputs. It runs each matching test case executable to acquire a result, and stores it in a text file. After this is done for all of the test cases, it compares each one to its respective oracle, and determines if a pass or fail has occurred. The script then creates an HTML file, complete with table, and inputs the data and opens it.

Our current file build doesn't include each test case executable adding data to the HTML file, only the main python script. Currently, the file pathing isn't set up to run in any location that the parent TestAutomation folder is inside of, but that isn't a complicated issue to solve. Running the program as-is currently only works on Jameson's laptop, as the file paths are set up with his Ubuntu's file structure naming in mind. This will be changed.

Current test cases:

TestNumber: 1
TestedRequirement: Test that passing two whole integers will return their sum
TestedComponent: functions.py
TestedMethod: add(x, y)
driver: TestCase1.py
TestInputs: 1 3
Oracle: testCase1oracle.txt

TestNumber: 2
TestedRequirement: Test that passing one number and one string will return a Type Error
TestedComponent: functions.py
TestedMethod: add(x, y)
driver: TestCase1.py
TestInputs: 1 testString
Oracle: testCase2oracle.txt

TestNumber: 3
TestedRequirement: Test that adding two negative numbers returns the correct negative sum
TestedComponent: functions.py
TestedMethod: add(x, y)
driver: TestCase1.py
TestInputs: -1 -1
Oracle: testCase3oracle.txt

TestNumber: 4
TestedRequirement: Test that the sum of zero and an integer return that integer
TestedComponent: functions.py
TestedMethod: add(x, y)

driver: TestCase1.py
TestInputs: 0 7
Oracle: testCase1oracle.txt

TestNumber: 5
TestedRequirement: Test that two very precise decimals, to the 20th decimal place, will have a sum of zero
TestedComponent: functions.py
TestedMethod: add(x, y)
driver: TestCase1.py
TestInputs: .99999999999999999999 .00000000000000000001
Oracle: testCase1oracle.txt

Chapter 4 - Deliverable #4, dated 12 November 2015

Team Silver Bullet was able to successfully design, implement, and test 25 test cases to check the limitations and abilities of the Calculator activity within Sugar. For the most part, everything went easily. The subtraction and division methods went off without a hitch. We ran into a small snag when working with multiplication, however. In Python, from what we found, when one multiplies a string and a number, the result will be a string that is simply the original string, but repeated X times, where X is the number in the equation. As an example, we originally expected a ValueError to result from multiplying the following:

“silver”
4

However, our outcome file contained “silversilversilversilver”. Truth be told, we initially had 373 as our number, so the file was quite long, so we shortened it to 3. Going from here, we had an interesting problem; when we read the test result strings from the expected output text files, we saw through the terminal that both the expected outputs and oracles were the same, however they failed when we attempted to compare them. We tried many different ways to fix the issue, and eventually found out that there was a line break in the files that we had to remove before testing would continue.

We had a small snag with our mod function testing as well. On one of our tests, we attempted to run a mod function with a decimal, however what we found was that the Calculator activity wasn’t designed to handle decimals in that function! After some brief discussion, we decided that it was a limitation of the program, and the test ended up failing.

Testing Framework:
/scripts/ - contains main python running script, runAllTests.py
/testCases/ - contains text files detailing each test case. Holds information to be passed
/project/src/ - location of the python Calculator activity that is being tested
/oracles/ - contains text files with expected outcomes of each test case
/testCaseExecutables/ - contains the python executable files to run each test case
/reports/ - contains the HTML file containing the aggregated result information
/Outputs/ - contains text files of test case executable results after the program is run

(Note: The following instructions still apply to the current project!)

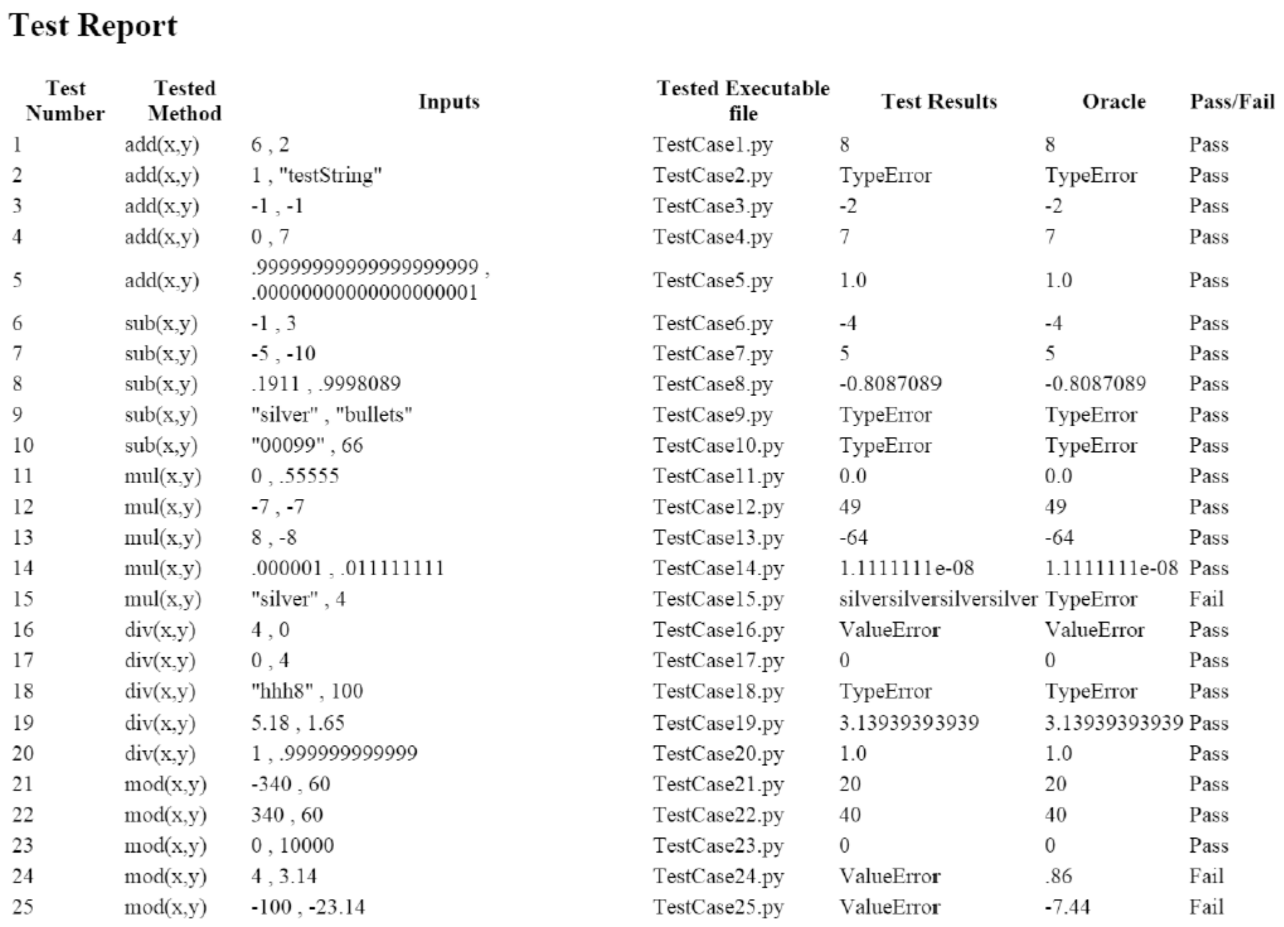
Running Instructions:
1. Clone the project from the TestAutomationProject branch to any desired filepath

2. Navigate to /TestAutomation/scripts/

3. Run 'runAllTests.py' with python2

4. An HTML file will open, the location of this file is at /TestAutomation/reports/

Test Case Overview



Test Report

Test Number	Tested Method	Inputs	Tested Executable file	Test Results	Oracle	Pass/Fail
1	add(x,y)	6 , 2	TestCase1.py	8	8	Pass
2	add(x,y)	1 , "testString"	TestCase2.py	TypeError	TypeError	Pass
3	add(x,y)	-1 , -1	TestCase3.py	-2	-2	Pass
4	add(x,y)	0 , 7	TestCase4.py	7	7	Pass
5	add(x,y)	.9999999999999999999 , .0000000000000000001	TestCase5.py	1.0	1.0	Pass
6	sub(x,y)	-1 , 3	TestCase6.py	-4	-4	Pass
7	sub(x,y)	-5 , -10	TestCase7.py	5	5	Pass
8	sub(x,y)	.1911 , .9998089	TestCase8.py	-0.8087089	-0.8087089	Pass
9	sub(x,y)	"silver" , "bullets"	TestCase9.py	TypeError	TypeError	Pass
10	sub(x,y)	"00099" , 66	TestCase10.py	TypeError	TypeError	Pass
11	mul(x,y)	0 , .55555	TestCase11.py	0.0	0.0	Pass
12	mul(x,y)	-7 , -7	TestCase12.py	49	49	Pass
13	mul(x,y)	8 , -8	TestCase13.py	-64	-64	Pass
14	mul(x,y)	.000001 , .011111111	TestCase14.py	1.1111111e-08	1.1111111e-08	Pass
15	mul(x,y)	"silver" , 4	TestCase15.py	silversilversilversilver	TypeError	Fail
16	div(x,y)	4 , 0	TestCase16.py	ValueError	ValueError	Pass
17	div(x,y)	0 , 4	TestCase17.py	0	0	Pass
18	div(x,y)	"hhh8" , 100	TestCase18.py	TypeError	TypeError	Pass
19	div(x,y)	5.18 , 1.65	TestCase19.py	3.13939393939	3.13939393939	Pass
20	div(x,y)	1 , .999999999999	TestCase20.py	1.0	1.0	Pass
21	mod(x,y)	-340 , 60	TestCase21.py	20	20	Pass
22	mod(x,y)	340 , 60	TestCase22.py	40	40	Pass
23	mod(x,y)	0 , 10000	TestCase23.py	0	0	Pass
24	mod(x,y)	4 , 3.14	TestCase24.py	ValueError	.86	Fail
25	mod(x,y)	-100 , -23.14	TestCase25.py	ValueError	-7.44	Fail

Chapter 5 - Deliverable #5, dated 22 November 2015

For this deliverable, our team injected five different faults into the code, one for each of the five methods we tested.

add(x,y)

Pre-faults

1	add(x,y)	6 , 2	TestCase1.py	8	8	Pass
2	add(x,y)	1 , "testString"	TestCase2.py	TypeError	TypeError	Pass
3	add(x,y)	-1 , -1	TestCase3.py	-2	-2	Pass
4	add(x,y)	0 , 7	TestCase4.py	7	7	Pass
5	add(x,y)	.99999999999999999999 , .00000000000000000001	TestCase5.py	1.0	1.0	Pass

Post-faults

1	add(x,y)	6 , 2	TestCase1.py	8.0	8	Fail
2	add(x,y)	1 , "testString"	TestCase2.py	TypeError	TypeError	Pass
3	add(x,y)	-1 , -1	TestCase3.py	2.0	-2	Fail
4	add(x,y)	0 , 7	TestCase4.py	7.0	7	Fail
5	add(x,y)	.99999999999999999999 , .00000000000000000001	TestCase5.py	1.0	1.0	Pass

For the add(x,y) method, we added a fault to return the absolute value of the addition results. This not only made the expected negative results positive, it also ended up converting the value to contain a floating point! This was unforeseen, and caused other tests to fail that we expected to pass.

Changes :

```
#fault inserted: added abs method to return value
return abs(x + y)
```

sub(x,y)

Pre-faults

6	sub(x,y)	-1 , 3	TestCase6.py	-4	-4	Pass
7	sub(x,y)	-5 , -10	TestCase7.py	5	5	Pass
8	sub(x,y)	.1911 , .9998089	TestCase8.py	-0.8087089	-0.8087089	Pass
9	sub(x,y)	"silver" , "bullets"	TestCase9.py	TypeError	TypeError	Pass
10	sub(x,y)	"00099" , 66	TestCase10.py	TypeError	TypeError	Pass

Post-faults

6	sub(x,y)	-1 , 3	TestCase6.py	-4	-4	Pass
7	sub(x,y)	-5 , -10	TestCase7.py	5	5	Pass
8	sub(x,y)	.1911 , .9998089	TestCase8.py	-0.8087089	-0.8087089	Pass
9	sub(x,y)	"silver" , "bullets"	TestCase9.py	TypeError	TypeError	Pass
10	sub(x,y)	"00099" , 66	TestCase10.py	TypeError	TypeError	Pass

We removed the if statement that would convert values to decimals.

Changes :

```
#fault inserted: took away if statement so it would not convert values to
decimals.
#if isinstance(x, _Decimal) or isinstance(y, _Decimal):
 #   x = _d(x)
  #  y = _d(y)
return x - y
```

mul(x,y)

Pre-faults

11	mul(x,y)	0 , .55555	TestCase11.py	0.0	0.0	Pass
12	mul(x,y)	-7 , -7	TestCase12.py	49	49	Pass
13	mul(x,y)	8 , -8	TestCase13.py	-64	-64	Pass
14	mul(x,y)	.000001 , .011111111	TestCase14.py	1.1111111e-08	1.1111111e-08	Pass
15	mul(x,y)	"silver" , 4	TestCase15.py	silversilversilversilver	TypeError	Fail

Post-faults

11	mul(x,y)	0 , .55555	TestCase11.py	None	0.0	Fail
12	mul(x,y)	-7 , -7	TestCase12.py	None	49	Fail
13	mul(x,y)	8 , -8	TestCase13.py	None	-64	Fail
14	mul(x,y)	.000001 , .011111111	TestCase14.py	None	1.1111111e-08	Fail
15	mul(x,y)	"silver" , 4	TestCase15.py	None	TypeError	Fail

For this method, we indented the return statement to be included in an if statement. The if statement checks if either number is a decimal. Indenting it causes Python to return nothing when the method is called, if the if statement isn't checked.

Changes :

```
if isinstance(x, _Decimal) or isinstance(y, _Decimal):
    x = _d(x)
    y = _d(y)
    #fault inserted: indented return statement.
    return x * y
```

div(x,y)

Pre-faults

16	div(x,y)	4 , 0	TestCase16.py	ValueError	ValueError	Pass
17	div(x,y)	0 , 4	TestCase17.py	0	0	Pass
18	div(x,y)	"hhh8" , 100	TestCase18.py	TypeError	TypeError	Pass
19	div(x,y)	5.18 , 1.65	TestCase19.py	3.13939393939	3.13939393939	Pass
20	div(x,y)	1 , .999999999999	TestCase20.py	1.0	1.0	Pass

Post-faults

16	div(x,y)	4 , 0	TestCase16.py	4	ValueError	Fail
17	div(x,y)	0 , 4	TestCase17.py	ValueError	0	Fail
18	div(x,y)	"hhh8" , 100	TestCase18.py	TypeError	TypeError	Pass
19	div(x,y)	5.18 , 1.65	TestCase19.py	3.13939393939	3.13939393939	Pass
20	div(x,y)	1 , .999999999999	TestCase20.py	1.0	1.0	Pass

We changed the clause to check for is ‘y’ is a 0 to check if ‘x’ is a 0. This causes the test to return an incorrect result when attempting to divide by zero. In test 16, we expect a ValueError, however they have a clause in there that returns ‘x’ if the ‘y’ is a 0. In test 17, a ValueError is thrown, since that is the result when it checks to see if the ‘x’ value is 0.

Changes :

```
if x == 0 or x == 0.0:#fault insert changed y to x
```

mod(x,y)

Pre-faults

21	mod(x,y)	-340 , 60	TestCase21.py	20	20	Pass
22	mod(x,y)	340 , 60	TestCase22.py	40	40	Pass
23	mod(x,y)	0 , 10000	TestCase23.py	0	0	Pass
24	mod(x,y)	4 , 3.14	TestCase24.py	ValueError	ValueError	Pass
25	mod(x,y)	-100 , -23.14	TestCase25.py	ValueError	ValueError	Pass

Post-faults

21	mod(x,y)	-340 , 60	TestCase21.py	20	20	Pass
22	mod(x,y)	340 , 60	TestCase22.py	40	40	Pass
23	mod(x,y)	0 , 10000	TestCase23.py	0	0	Pass
24	mod(x,y)	4.1 , 3	TestCase24.py	1.1	.86	Fail
25	mod(x,y)	-100 , -23.14	TestCase25.py	-7.44	-7.44	Pass

For the mod function, we removed the clause that checks if ‘y’ is an integer. This allows us to mod by decimal numbers, which the program wasn’t allowing us to do before. While this is added functionality, it is still technically a fault from what we expected the program to return.

Changes:

```
#if is_int(y):
return x % y
#else:
    #raise ValueError(_('Can only calculate x modulo <integer>'))
```

Chapter 6 - Member experiences

Jameson Burroughs
“I would have to say teamwork and independent learning.”

Christopher Lewis
“Working as a self-guided team showed me the importance of creating project milestones. Without required deliverables, our team would have certainly fallen behind schedule. Working with the deliverable guidelines allowed us to schedule fairly regular team meetings to hash out both technical problems and organizational requirements. Management of time and resources are vital to the success of a project that includes multiple steps. Hurrying to get it done would have spelled disaster for us! If I had to change one thing about the project, I’d write slightly clearer instructions for the specifications required for our testing automation framework. We had to go back and change our structure and functionality a few times to meet the specifications.”

Megan O’Neal
“I learned that deliverables are necessary with group projects because they hold people accountable and people work better under a least a little pressure. Documentation is important when explaining yourself and giving yourself credibility. Also, assigning roles are necessary to hold people responsible and give credibility, but these roles may change as you go and you have must be flexible.”

Alexander Wray
On the programming side I learned the basics of bash/Linux and how to interact with the command panel to get it to do things, I also dusted off my python knowledge some and in relation to this learned just how quickly you can forget the form of a language if you get too used to using another language. Aside from programming I learned that making sure everyone, in a group, feels like they are getting meaningful and fulfilling jobs can sometimes be difficult, especially if the main task at hand is something fairly easy, but has to be complete before other work can be done. Another important lesson of this project was if it doesn't work sometimes scraping and starting over is a

better approach, this was apparent in our script which was giving us trouble in Linux, but ended up working well in python. One of the things this project made me question the most outside of programming problems is team numbers for our assignment 4 people worked, but it could have been done with less, however what I really think is an interesting problem is how to tell the optimal number of people for a task, because if our class had chosen five members per team there would have inevitably been a lack of work for at least one or two people.