Team Term Project Final Report

CSCI 362 Fall 2019

Team: Seawolves

Members: Shaina Mainar, AJ Williams, David Spry

Table of Contents

Introduction…….3

Chapter 1: Deliverable 1…….4

Chapter 2: Deliverable 2…….5

- Test Plan

Chapter 3: Deliverable 3…….8

- Framework
- How to Run

Chapter 4: Deliverable 4……...10

Chapter 5: Deliverable 5……...11

- Fault Injections

Chapter 6: Overall Experiences……12

Introduction

This is the final report for our term project for CSCI 362. The project was to design an automated testing framework for an open source project of our choosing. The framework is designed to run in Ubuntu and invoked using a single script being run from the command line. Featured in this report are our experiences during and after the project as well as details concerning our test plan and test framework, instructions on running the framework, and examples of the output and other parts of the project,

Chapter 1: Deliverable 1

The first step in the project was choosing which open source project we would be building the framework for. After considering a couple different options we decided to go with SugarLabs, a learning software tool for children. We settled on this project due to its simplicity when compared with other projects, its lack of online or database components and relative lack of dependencies being two examples. This simplicity would allow us to focus more on the testing framework rather then spending a lot of time understanding how the parts of sugar that we were testing work.

While by the end this overall turned out to be true, we ran into some problems early on. We knew that there were going to be some issues with the python scripts running on Linux. We had anticipated it, but it still gave us some troubles. The first issue is running the tests. There was test runner on the main repository's source file, however it did not run as expected. So, it was decided that just running the test cases themselves should suffice. The problem that was not anticipated was the libraries and dependencies. The first error that was encountered was a ModuleNotFound error. This is most likely due to how the interpreter is set up on Linux and how the python path works. Shaina researched the many ways of getting the proper modules in the python path and the system path. She was not successful. So, the next step, was to drop the libraries needed in the same directory as the test cases. Although the library issue was solved, another one manifested. There was an ImportError stating "cannot import name X." After some research, it was found that the most likely cause of this is a circular dependency issue. Knowing that circular dependencies can be tricky to maneuver, it was time to find another option.

After some more thought and research, David found a way to test python script through Linux with pytest. During installation, the first instinct is to immediately type, "sudo apt install pytest" or "pip install pytest" and this works, yes. However, this project has been recently ported to python3 therefore, some libraries do not exist in python2 which is what most of the team's machines had installed pip with. Therefore, back to Google. Shortly after, a solution was found. Pip-3.2 had to be installed or the test case had to ran using the following command, "python3 -m pytest testcase.py." Quite frankly, the latter was the easiest to do because it was the one that had a 100% success rate. Some of the test cases worked, some of them still had issues with the modules.

These problems mostly came from us trying to compile the whole project and use a built-in testing system that the developers had included. Fortunately, this turned out to be mostly unnecessary since we would be only using very specific methods to test our framework which allowed us to ignore most dependencies. Also, thanks to the code being in Python we ended up not needing to compile the code in order to run the methods that we were using either.

Chapter 2: Deliverable 2

Once we had settled on a project and gotten an understanding of exactly what we were going to need running for this project the next step was to set up a test plan for our framework and decide on the methods that we were going to test as well as the test cases that we were going to use for said methods. There were some changes made to this plan as the project progressed, such as us adding or changing what methods we were using, or the exact format of our test case files. Below you can see the final test plan for our project. At the end of the test plan 5 examples of our test case files can be seen. In the framework these are each contained in a separate text file.

Deliverable 2: Sugar Test Plan
Prepared by: Shaina Mainar, David Spry, AJ Williams

Requirements:

We are testing methods within different activities in the Sugar ecosystem. Therefore, the requirements will be varied.

- Requirements for agepicker.py method: calculate_birth_timestamp
  - Method needs to take in an integer and return an integer value
- Requirements for agepicker.py method: calculate_age
  - Method needs to take in an integer and return an integer value
- Requirements for calculate.py method: findchar
  - Method needs to take in a string and a character and output the index of that character in the string or a -1 if the character is not found
- Requirements for functions.py method: gcd
  - Method needs to take in two integers and return the gcd of those two integers as an integer
- Requirements for functions.py method: factorial
  - Method needs to take in an integer and return the factorial of that number as an integer
  -

Tested items:
src/jarabe/intro/agepicker.py
calculate-activity/calculate.py
calculate-activity/functions.py

Testing schedule:

Deliverable 1: September 12, 2019

- Checkout and clone HFOSS project. Compile project and run.

Deliverable 2: October 1, 2019

- Detailed test plan specifying 5 of 25 test cases.

Deliverable 3: October 29, 2019

- Re-working of test plan and building automated testing framework.

Deliverable 4: November 12, 2019

- Complete the design and implementation of the testing framework. Create 25 test cases for the framework

Deliverable 5: November 19, 2019

- Design and inject 5 faults into source code, test, and analyze the results.

Final Report: November 21, 2019

- All deliverables, code, and files are collected and delivered.

Test recording procedures:

The results of the tests will be recorded in an html document that is timestamped with the date that the tests were run

Hardware and software requirements:

The project should be run on Linux-based. For our project, we all used Ubuntu 18 LTS and Python 3.

Constraints:

Limited knowledge of scripting and testing framework. We will be researching methods to use these throughout the project. For this project, we are only utilizing unit testing in which we test the individual methods from the software ecosystem.

Template:

Test Number: Integer number of the test
Requirement: The Requirement being tested
Component: The name of the class that contains the method to be tested
Method: The method to be tested
Test Input: The input to be used in the test
Expected Outcomes: The oracle or expected outcome from the input

Tests:

01
Accepts an age and outputs the birth time stamp in seconds
agepicker
agepicker.calculate_birth_timestamp
5
1077212919

02
Accepts an age and outputs the birth time stamp in seconds
agepicker
agepicker.calculate_birth_timestamp
0
1234892919

03
Accepts an age and outputs the birth time stamp in seconds
agepicker
agepicker.calculate_birth_timestamp
-5
1392572919

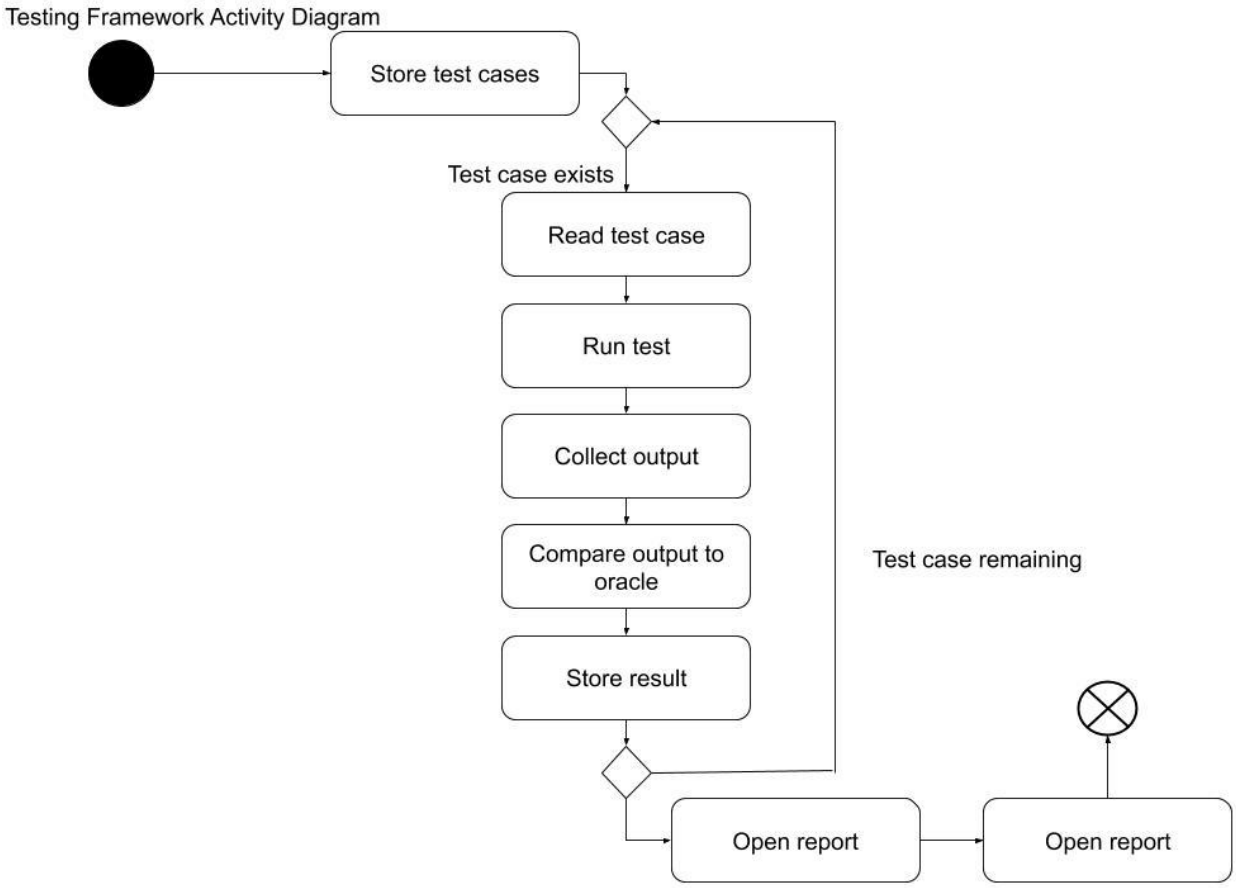
04
Accepts an age and outputs the birth time stamp in seconds
agepicker
agepicker.calculate_birth_timestamp
5.5
1061444919

05
Accepts an age and outputs the birth time stamp in seconds
agepicker
agepicker.calculate_birth_timestamp
3000000000
-94607998765107088

One last thing to note is concerning the calculate_birth_timestamp and calculate_age methods. In order to more easily create and test our framework we had to make a change to these methods in the source code. The methods originally used the python time.time() method to get the current time when calculating the values. However, this made being able to calculate the oracle for each test tricky, since it would be different every time we ran the tests. So to solve this problem we changed the time.time() method to a static number that was similar to the number that would be returned by that method for the purposes of the project.

Chapter 3: Deliverable 3

With a plan in place and the methods decided on the next step was to begin working on the script that would drive the framework. The real question that then needed to be answered was how we were going to run the methods that we needed to test. We first investigated running the methods through the python interpreter that is in Linux. While this would work manually, we were unsure of its viability when it came to preforming through a script. Fortunately, after some research Shaina was able to find a way to run python methods through the Linux command line, this simplified the design of our framework immensely because it eliminated the need for any drivers or complicated script commands. Below you will find an activity diagram that describes our framework as well as an explanation. After that will be instructions on how to run the framework.



The script begins by creating an array of all the test case file names. It then will then being with the first test case file and read each line and store the information in the proper variable in the script, which variable to store the information in is based on the line being read in, for example line 3 is always the name of the module that the method being tested is in. After reading all the information for that test case the script will then run the method that was specified using the input that was given in the test case. It will then compare the output that the method returned

with the oracle gotten from the test case and store the result in an array. It will then loop back to the start and repeat this for every test case. After all the tests have been run it will generate a html report using the test data and open this report in a web browser.

How to run

Before using the framework, some prerequisites must be first installed.

Prerequisites:

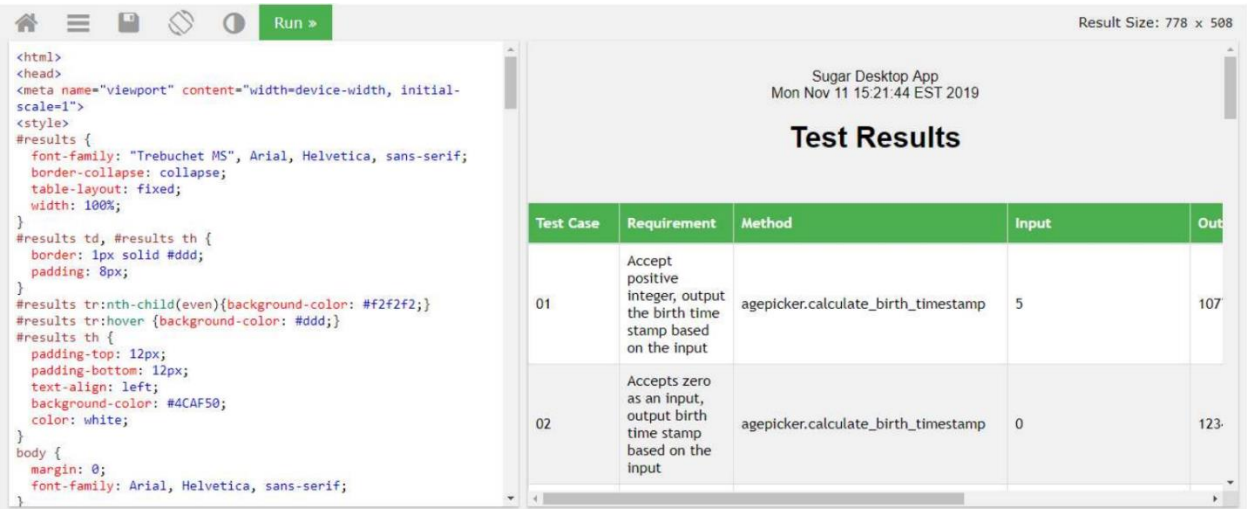
1) Python 2
    a. sudo apt install python
2) The Python gi library
    a. sudo apt install python-gi

To run:

1) Run the framework through the terminal
    a. Ensure you are in the TestAutomation folder
    b. From this folder run the command: bash ./scripts/runAllTests.sh
2) The results will be put into an html document that will automatically open in the default browser
    a. If you wish to save this document elsewhere or need to reopen it, the html document is stored in the reports folder within TestAutomation

Chapter 4: Deliverable 4

With the framework functioning we turned our attention to the results page. While creating the framework our results page was just a simple html table, functional but not particularly interesting to look at. So, we took some time to add some css to the html that made up the report in order to make it more readable. An example of the improved version as well as a snippet of the css that we are using can be seen below.



There was one other change that we made at this time. As mentioned before in order to make things easier we replaced the time.time() method that was used in calculate_birth_timestamp and calculate_age with a static number. Originally the number that we were using was 1500000000. We concluded, after comments from Professor Bowring, that this number was not as representative of the time.time() method as it could be so we changed that number in the source code to 1234892919. This required us to go back through the test cases for these two methods and recalculate the oracles using this new number.

Chapter 5: Deliverable 5

The last thing for us to do in the project was fault injection to ensure that our framework could detect changes in the code. For this we changed the code in five places, injected five "faults", to see how that would change the results of our test cases. Below you will see a list of the changes that we made.

functions.factorial at line 302
WAS res *= n - 1
CHANGED TO res = n - 1

functions._do_gcd at line 266
WAS return _do_gcd(b, a % b)
CHANGED TO return _do_gcd(b, a / b)

agepicker.calculate_birth_timestamp at line 48
WAS birth_timestamp = int(1234892919.655932 - age_in_seconds)
CHANGED TO birth_timestamp = (1234892919.655932 - age_in_seconds)

agepicker.calculate_age at line 54
WAS age = int(math.floor(age_in_seconds / _SECONDS_PER_YEAR) + 0.5)
CHANGED TO age = int(math.floor(age_in_seconds / _SECONDS_PER_YEAR))

calculate.findchar at line 60
WAS level -= 1
CHANGED TO level += 1

The test cases that we noticed failed were: 1, 2, 3, 4, 5, 9, 16, 17, 19, 20, 21, 22, 24, 29

Test Case	Requirement	Method	Input	Output	Oracle	Result
05	Accepts an age and outputs the birth time stamp in seconds	agepicker.calculate_birth_timestamp	3000000000	-9.46079987651e+16	-94607998765107088	FAIL!
06	Accepts a birth time stamp and outputs the age	agepicker.calculate_age	-1000000	39	39	Pass
07	Accepts a birth time stamp and outputs the age	agepicker.calculate_age	1000000	39	39	Pass
08	Accepts a birth time stamp and outputs the age	agepicker.calculate_age	0	39	39	Pass
09	Accepts a birth time stamp and outputs the age	agepicker.calculate_age	3000000000	-56	-55	FAIL!
10	Accepts a birth time stamp and outputs the age	agepicker.calculate_age	1.5	39	39	Pass

Chapter 6: Overall Experiences

David:

Overall, I felt that this project was a good showcase for creating test cases for methods as well as a simple example of how an automated test system would work. As an unintended consequence of the open source project that we used I also have a newfound appreciation for documentation, both inside and outside of the code. This is because one of the early spots of frustration for me personally was the lack of any comments within the code or any detailed outside documentation. This led to the need to use some trial and error and speculation when understanding what the code was doing, though fortunately we were able to find some self-explanatory methods that made that task easier. I also feel that the difficulty of the project has some correlation to the open source project that is chosen at the beginning. Other than some minor hicupps in the beginning working with SugarLabs was fortunately fairly easy. It being written in python also helped as we were able to just run the modules and methods from the command line allowing us to avoid using drivers. Had we chosen a different project I have a feeling that our experience might have been at least slightly different.

As for what I learned during this project there one of the obvious things would be scripting, even if AJ took care of a majority of the scripting, I was able to pick up some stuff by reading over and understanding his work. I also feel more comfortable in Linux now after having to us it for a whole semester, I typically us Windows so it was good to get exposure to a different OS. Also as stated before this project helped me better understand test case construction and a better way to approach testing in the future, which I think will even go so far as to help me write better code.

I feel that our team delivered a quality product that meets the project specifications. There is more work that we could have done with the results page to make it more interactive and possibly contain more information, but I think it is functional for what it needs to do.