Test Suite
For
Sugar Labs

Developed by Team JAB-
Julius Alipala, Alexander Austin, Bryce Barrett

Table of Contents

Introduction:

This report details the process of completing the term project for CSCI 362: Software Engineering taught by Dr. Bowring at the College of Charleston. The project described herein was completed by team JAB consisting of students Julius Alipala, Alexander Austin, and Bryce Barrett. For this project, our team was tasked with selecting an HFOSS (Humanitarian Free Open Source Software) project to create an automated testing framework for, with further specifications that the project runs in an Ubuntu environment and performs 25 tests on the software.

Chapter 1: Selecting, Building, and Running Preexisting Tests on an HFOSS Project

Initially, our team choose to work with Miradi, which is a piece of software that is designed to allow Nature Conservation project managers to have an easy and intuitive way to organize and meet their goals. Unfortunately, an investigation into their GitHub revealed that it was a dead project. We then decided to go with Sugar, a desktop environment created by Sugar Labs in order to provide an activity-driven learning application for children. We chose to work on this project because naturally we feel that any software with the goal of facilitating education is very important, it has a fairly active community, and due to the fact that it is an entire desktop environment, there are a great deal of components available for us to test.

As far as building the software and running the pre existing tests goes, we found ourselves facing a certain degree of difficulty and confusion. The primary confusion came from the fact that there are actually multiple repositories of Sugar on GitHub, and it can be built in different ways. At first, we cloned from the sugar-build repository, as it appeared to be designed for the type of work we were to do (in fact, it even included a “runAllTests” executable. However, we could not get it to function properly, and after some research, we found out that the issue was that sugar-build was designed to only be used on Fedora Linux (an older version at that) and we were using Ubuntu. After that we decided to switch to the packaged style Sugar environment which provided a smoother building experience. We still encountered some problems with running the existing tests, but these problems came more from our lack of experience with python testing. After we spent more time with the code, we were able to get a few tests running. The pre-existing tests included with the code are shown below along with the results of running the tests.

Test #	Name	Result
1	test_activitieslist	Terminates early due to an ImportError for "uitreep". Even after verifying that the file to import did in fact exist, this issue could not be resolved at the time of this publication UPDATE: 9/14/2017. It appears that "uitreep" is actually a typo in the code, and when the test was edited to simply call"uitree" the test ran through to the end. However, the test still failed citing the error "AssertionError: unexpectedly None" UPDATE: 9/15/2017. Running the test while in the Sugar Desktop Environment results in the test passing
2	test_backup	Test Passes Properly
3	test_bundleregistry	Tests run through, but fail citing KeyError as 'SUGAR_ACTIVITIES_DEFAULTS'
4	test_desktop_config	Test Passes Properly
5	test_downloader	Test Passes Properly
6	test_journaltoolbox	Test Passes, however many warning messages stating things such as "WARNING:root:No icon with the name edit-duplicate was found in the theme."
7	test_microformat	Test file contains two tests, one passes and the other fails citing ERROR: test_html_parser (test_microformat.TestMicroformat), ValueError: too many values to unpack. This seems to be some sort of running error in the source files themselves
8	test_modemconfiguration	Terminates early, also stating an ImportError: No module named mock.
9	test_user_profile	Terminates due to ImportError: No module named

		cpsection.aboutme
10	test_webaccount	Test Passes Properly
11	test_webservice	Test Passes Properly

As previously stated, as of 9/13/2017, we have been unable to discover why precisely these tests are failing. However many of them seem to have issues pulling code from libraries. Since we inspected the source files and found that the specified files are in the specified places, the cause of this remains a mystery. However, we believe that, since we attempted several different builds of the software, this may have caused some issues with the files that are throwing errors. Another possible issue is the fact that the tests are 2 years old. Updates to the software may have caused certain tests that previously passed to fail. Aside from that, all team members lack experience in running python testing scripts, especially within an Ubuntu environment, which may be keeping us from solving the issue at hand. Naturally, this is an obstacle that must be overcome so that the goals of this project can be met. Through this, we shall gain a greater understanding of the software we are working with.

Chapter 2: Making a Test Plan

Testing Process

Although we will not be performing major system tests for our requirements of this project, we will be testing individual sub-systems within Sugar Labs. The main sub-system that we will be testing is the modeling system. Within the modeling system we will be testing the integrity of methods that help control the friend, neighborhood and buddy systems. We will also be testing parts of the frame, intro, journal, util, and view systems. In each system we have chosen specific methods and objects that will need to be tested and validated.

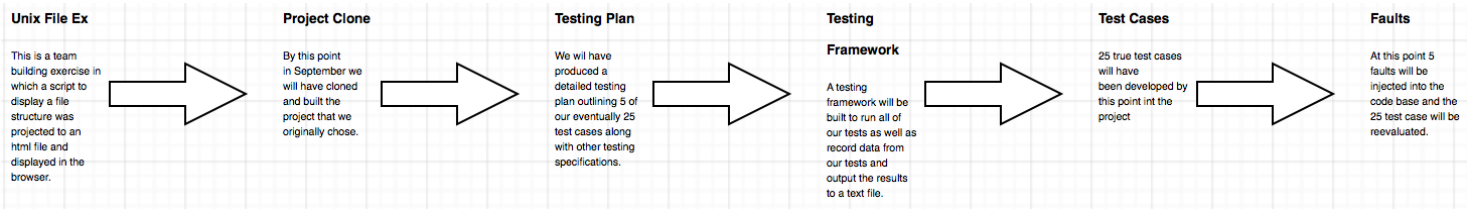
Requirements Traceability

Throughout the Sugar Labs project, there is very little documentation as to what certain classes and methods are doing. So, when we begin testing certain methods we will be creating some requirements of our own to satisfy this as we would otherwise not have any. If there is documentation as to what a method is doing or what requirement it need to satisfy, then we will develop our tests to mimic what the requirement demands. We will state in the test case what requirement we are testing.

Tested Items

The component folders that we are currently testing are the model and util folders. In these folders we are specifically testing normalize.py and friends.py. We currently have plans to use more components from these folders in future tests.

Testing Schedule



Test Recording Procedures

All test results will export to an external .html file. The results will be placed into a table that displays all relevant information about the test, whether or not the test passed or failed, and displays the reason for the failure.

Software Requirements

Currently, the only software requirements are being able to run python files as well as having a version

of UNIX installed. This is because Sugar Labs creates a desktop environment which requires a UNIX environment. As we continue working on this project of software requirements will expand.

Constraints

We are all full time students with a full slate of classes. We will try to create a thorough test schedule and adhere to it. Although creating a strict testing schedule is very helpful, it is still possible to run into unforeseen issues. If any issues arise, we should be able to quickly resolve them.

Initial Test Cases:

*Note that originally we had test cases that tested notifications.py and journalactivity.py, but these were unable to be implemented and then removed.

Test Case 1

Title: Test Remove Friends 1

Description: This test ensures that a User's Friends list stays correct after various modifications. Preconditions: All proper setup and config has been performed by user. A set of "Friends" object is instantiated by the driver found here. The AddFriends Test has already been performed and verified

ID: test_remove_friends

Drivers: friend_driver.py

Requirement: Friends should be removable from the friends list

Component: This tests the Friends module.

Method: This tests the remove_friends method.

Test Inputs: make_friend(friend1), make_friend(friend2), remove_friend(friend1)

Expected Outcomes: friends{} = {friend 2}

Test Case 2

ID: test_make_friends_valid_buddy

Drivers: friend_driver.py

Requirement: Valid BuddyModel objects should be added to the Friends' _friends attribute.

Component: This tests the Friends module.

Method: This tests the make_friends method.

Test Inputs: BuddyModel("nickOne", "keyOne, "accountOne", "idOne")

Expected Outcomes: _friends[keyOne: FriendBuddyModel("nickOne, "keyOne", "accountOne", "idOne")]

Test Case 3

ID: test_make_friends_none

Requirement: None type objects should not be accepted as friends.

Component: This tests the Friends module.

Method: This tests the make_friends method.

Test Inputs: None

Expected Outcomes: TypeError

Test Case 4

ID: test_make_friends_string

Requirement: Strings should not be accepted as friends.

Component: This tests the Friends module.

Method: This tests the make_friends method.

Test Inputs: ['hello', 'goodbye', 'apple', '', 'five', 'testing inputs']

Expected Outcomes: AttributeError

Test Case 5

ID: test_make_friends_invalid_buddy

Requirement: BuddyModel objects lacking a valid key attribute should not be accepted as friends.

Component: This tests the Friends module.

Method: This tests the make_friends method.

Test Inputs: BuddyModel("nickOne", None, "accountOne", "idOne")

Expected Outcomes: AttributeError

Chapter 3: Automated Testing Framework

Our current framework utilizes the directory specifications stated in the assignment. There is a script, "runAllTests.py" within the scripts folder which navigates to the testCases folder, where it reads the all the test cases, which at this point consist of:

Test Case 1

test_make_friend_string
Strings should not be accepted as friends.
Friends.py
make_friend "hello goodbye"
AttributeError

Test Case 2

test_make_friend_none
None type objects should not be accepted as friends.
Friends.py
make_friend
None
AttributeError

Test Case 3

test_make_friend_int
Ints should not be accepted as friends.
Friends.py
make_friend
5846
AttributeError

Test Case 4

test_make_friend_valid
Creates a valid Friend object and adds it to Friends
Friends.py
make_friend
'one', 'keyOne', 'accountOne', 'idOne'
True

Test Case 5

test_remove_friend
Adds then removes Friend objects and ensures data is correct
Friends.py
make_friend
'one', 'keyOne', 'accountOne', 'idOne'
False

Test Case 6

test_make_friend_stringEm
Empty strings should not be accepted as friends.
Friends.py
make_friend

AttributeError

For each test case, the first line determines which Test Case Executable to run, the second line offers a description of the test, the third line gives which program file will be tested, while the fourth line specifies the method to be tested. Line 5 consists of the inputs to be passed in, and line 6 consists of the expected outcomes. Our script runs the specified test case executable, passing in two sys.argv parameters: The input and the expected output. At the end of each test case there is an assertion that a value equals the expected value argument, and if true, "Test Passed!" is returned, but if false, returns "Test Failed" + the error specifications. All individual test results are then sent to the temp directory. After all tests have been run, the results are compiled into a single html file that is automatically opened to display the test results.

How to run the Framework:

Clone the repository to local machine.
Install dependencies. (run “sudo apt install sucrose” from the terminal)
Inside the terminal, navigate to the TestAutomation/scripts directory.
Give the script permission with the command "chmod 755 runAllTests.py"
Navigate back to TestAutomation directory.
Run the tests with the command "./scripts/runAllTests.py"

Overall this project has progressed smoothly, even though there were a few moments where we fell a bit off track, it was easy for us to pick back up with things. We maintained great communication at all times and this likely greatly facilitated our success thus far

Chapter 4: Developing Test Cases

The testing framework for Team JAB is currently running as specified. Our script walks through the 25 test cases contained in the testCases directory and executes the corresponding tests. The results of the tests, along with the relevant information from the test cases, are compiled into a single html file which is displayed upon completion.

Overall, this deliverable came together with few complications. Our main hurdle was in figuring out how to instantiate certain class objects, mainly due to a lack of documentation. For many files, simple import statements caused errors related to asynchronous processing that left us more than a bit confused at first. To fix this, we figured out that we had to use dbus statements to fix issues with concurrency. Even though we managed to get that problem solved, there were still some pieces of code that we could not test due to the fact that we simply could not figure out how precisely it worked. The best example of this is the journal module, which we had originally intended to run many of our test cases on. We spent a great deal of time inspecting all of the files and trying to work with them, but our efforts unfortunately yielded nothing. In fact, we have even reached out to one of the developers of SugarLabs, asking if they could explain it to us a bit more. We've yet to hear back from them, and even though we have satisfied all of our test case requirements, we still desire to gain a proper understanding of the code that gave us so much trouble. Anyway, once we figured out the issue with imports that we had, designing the remaining tests came fairly quickly. The tests we created consist of :

-For friends.py, we created numerous tests for the methods make_friend() and remove_friend() utilizing a variety of valid and invalid inputs.

-For normalize.py, we created a test for both active and passive normalization.

-For clipboard.py, we tested the add_object() and delete_object() functions with several inputs/scenarios.

-For keepicon.py, we had tests to ensure proper values were set at initialization

-For agepicker.py, we had tests for calculate_birth_timestamp() and calculate_age()

-For neighborhood.py we had tests for the add_buddy function

We are now preparing to complete our last delivery as well as the last few components for our final report.

UPDATE 11/15/2017: Upon Reaching out to Sugarlabs developer Sam Parkinson, she revealed to me

that she herself was unable to successfully run tests on the Journal module citing “layers and layers in indirection, and I'm not sure how it is intended to work.”

Chapter 5: Injecting Faults

The next step of our project was to inject 5 faults into the code of the files we were testing. The faults that we injected are as follows:

Fault 1.In clipboard.py, at line 77, in the add_object() method, these lines that check for an object existing at the specified hash were commented out, making it so that objects were overwritten.

```
        #if object_id in self._objects:
#  logging.debug('Clipboard.add_object: object already in clipboard,'
 #               ' selecting previous entry instead')
 # self.emit('object-selected', object_id)
 # return None
```

Fault 2: In friends.py, in the add_friend() method, line 122 (shown below) was removed, which resulted in errors in friend adding.

```
#self._friends[buddy_info.get_key()] = buddy_info
```

Fault 3: In normalize.py, on line 30 in the normalize_string() method, we introduced a typo in the field for type of normalization(as shown below) resulting in errors normalizing strings.

Original: return normalize('NFKD', unicode_string).encode('ASCII', 'ignore').lower()
Faulty: return normalize('NFK', unicode_string).encode('ASCII', 'ignore').lower()

Fault 4: In agepicker.py, on line 54 in the calculate_age() method, we introduced an arithmetic error that calculates wrong ages

Original: age = int(math.floor(age_in_seconds / _SECONDS_PER_YEAR) + 0.5)
Faulty: age = int(math.floor(age_in_seconds + _SECONDS_PER_YEAR) + 0.5)

Fault 5: In agepicker.py, on line 46 in the calculate_birth_timestamp() method, we introduced an arithmetic error that produces wrong birthdates.

Original: age_in_seconds = age * _SECONDS_PER_YEAR
Faulty: age_in_seconds = age + _SECONDS_PER_YEAR

These Errors resulted in the failures of 8 tests, which were the test_birth_timestamp,

test_age_timestamp, test_remove_friend, test_has_buddy_valid, test_normalize, test_normalize_passive, test_add_clipboard_identicalhast, and test_remove_friend_invalid.

Chapter 6: Overall Experience

Throughout this project, many valuable things were learned by all members of the group. Aside from some of the obvious outcomes, such as learning all facets of how to create an automated testing framework, getting more familiar with Bash scripting, learning how to read through and decipher foreign code, and coordinating all of these with a team, there were several other things learned throughout our project. One vital thing that the team learned over the course of this experience is the importance of thoroughly commenting and explaining code. Unfortunately, Sugar (The chosen software) featured very little documentation in any of it's code. Early on, this gave us great difficulty in determining how functions were supposed to work, let alone how to test them. This was not eased by the fact that the project was coded in python, so in each method header, data types are not specified, and sometimes there was difficulty in determining what precisely was supposed to be passed into the method. However, after spending a good deal of time with all of the chosen methods, the team was able to get things figured out. A big issue with developing test cases was that there were problems with concurrency, but these we were able to fix by using python's dbus library. Another important skill that our team learned was the importance of constant communication. The goals of this project consisted of things that none of the team members really had any experience with, but the team remained in constant contact and met often, and because of this, things proceeded quite smoothly for the most part.