CSCI 362: Final Report

Team H/Foss Project: Beets

Team Isn't This Fun:

Ben Byrd, Zachary Davis, Kaitlyn Fulford, and Adam Sugarman

Dr. Bowring

Section 3

December 1, 2015

Table of Contents

Introduction

Our team name is Isn'tThisFun. This final report shows our progress and experiences throughout the Fall 2015 semester on the testing of an H/FOSS project of our choice. After realizing quickly that our first choice, Open Trip Planner, was not going to work for our project, we chose Beets, a media and music library management system. The software is written in Python. Therefore, this helped us understand and work easily with the project source code. We chose methods that our entire team understood. Capitalizing on each other's skills, we quickly found out our strengths and weaknesses and assigned roles accordingly. In order to test Beets and its methods, we planned and implemented a testing framework, including a script, drivers, test case text files, and html reports. This cohesive report consists of six chapters that include the test plan, the testing framework details, and injected faults.

Chapter One

Cloning, Building, and Compiling H/FOSS Project: Open Trip Planner

Cloning and Building

We were able to clone the project, Open Trip Planner, from Github. Working from OTP's "Getting OTP" documentation, we went through the steps, which were all seemingly simple enough. We downloaded openJDK 8, maven, and git. Then, we cloned the project and ran a build and clean through Maven command. Initially, we received a "build failure" result with the error message outputted to the screen that we needed a higher version of Maven. On further inspection, we discovered that the version of Maven that we downloaded, which was the default that OTP told us to use in the command line, was an older version of Maven. Updating Maven proved more difficult than we thought. After searching for a solution through trial and error, we finally found a very recent tutorial that was extremely helpful. We downloaded the newer Maven and made sure it had downloaded correctly. Therefore, we ran the command to build OTP and "clean package" once again. After a few minutes, once we all got the message "build success", we tried to compile it. This proved troublesome because we had to decide whether to compile it on the command line, Eclipse, NetBeans, or some other compiler. After some research, it seemed as if Eclipse would be too complicated for this complex project; therefore, one group member tried to compile it on NetBeans and another on the command line from the terminal. This was easier said than done. We debated on if we should abandon OTP and start another; however, we decided to continue with OTP.

Compiling and Tests

Downloading NetBeans on Ubuntu was tricky as first because the NetBeans Linux installation instructions produced errors on Ubuntu. Once again, a team member searched for a NetBeans tutorial for installation on Ubuntu and followed instruction from ubuntuhandbook.org. Their steps were very straightforward and there were no problems downloading NetBeans. Once the project was opened in NetBeans, everything seemed to be running great since there was no code underlined in red. A few tests ran with success; however, NetBeans decided to download a bunch of plugins randomly and after that, the tests would no longer run with failed exception messages in the output that it needed a different Maven plugin, which was not found. It also seems that NetBeans has duplicated the dependencies in that it downloaded them twice and that is the reason for the fail.

Experience

Overall, the trial and error process taught a lot in how to figure out other ways to do a task when one way would not work. Although frustrating at times because OTP was more complex than realized (code seemed very organized), the "build success" we received boosted our confidence. The plugins for Maven and running the tests gave us difficulty. However, our greatest problem was scheduling meeting times to work on the project because we all have very different schedules. However, once we found a time, we successfully built OTP on Ubuntu. The tests

seemed to have failed because of the other dependencies such as getting data for different cities to use and various other files that OTP requires.

Chapter Two

Test Plan H/FOSS Project: Beets

Introduction

Our team has decided to switch from our original choice, Open Trip Planner, to our other choice, the project Beets. Open Trip Planner contained too many bugs and problems and would be a problem to actually test. Beets, a music manager, is a python program and runs straight from command line. Additionally, Beets provides a fair number of testable classes. Our goal for the project is to create an automatic testing framework for the open source project Beets through collaboration. Ultimately, the test results, pass or fail, will be displayed on a HTML page. In this deliverable, we will outline five of our tests cases and detailed specifications of our test plan.

The Testing Process

Team Isn't This Fun will create a script to run five automated tests on five methods within the project Beets, for a total of 25 tests. These tests will compare the output of each method with an oracle for that method. The script will then display the results in a web browser. The way our team will do this is running executable files, which create output files to be compared to the corresponding oracle files. The results of this comparison between the output file and oracle file will determine whether the test passes, and then the results will be recorded and then opened and displayed on an HTML page within a web browser.

Requirements Traceability

Users are most interested in the system meeting its requirements and individually testing all requirements. Requirements of methods:

1. human_bytes(size) - The method should take in an integer corresponding to file size and output a human readable string displaying the file size.
2. human_seconds(interval) - The method should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.
3. human_seconds_short(interval) - The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.
4. _sc_decode(soundcheck) - The method should take soundcheck files as input and output a decoded full
   scale value.
5. _string_dist_basic(str1, str2) - The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation.

Tested Items

The methods of the project, Beets, which are to be tested are as follows.

Method and its Location:

human_bytes(size) → beets/ui/*init*.py

human_seconds(interval) → beets/ui/*init*.py

human_seconds_short (interval)→ beets/ui/*init*.py

_sc_decode(soundcheck) → beets/mediafile.py

_string_dist_basic(str1, str2) → beets/autotag/hooks.py

Testing Schedule

Overall Project Testing Schedule Based on Syllabus

9/29: Deliverable #2 - Create Test Plan with 5 of 25 test cases

10/22: Deliverable #3 - Design and build an automated testing framework including architectural description (with at least 5 test cases)

11/12: Deliverable #4 - Complete design and implementation of testing framework and create 25 test cases

11/24: Deliverable #5 - Design and inject five faults into code

12/1: Team Project Presentations and Final Report

Detailed by Week Schedule

9/28 - 10/4: Revise Test Plan if necessary

10/5 - 10/11: Driver frame working (Oracle output, etc) / Basic script working (call driver, read from file)

Designing automated testing framework /scripting / 5 Test Cases

10/12 - 10/18: Driver more functional / Script more functional

Designing automated testing framework / scripting / 5 Test Cases

10/17-20: Fall Break / Driver/Oracle/Script working for at least 5 test cases

10/21- 25: 5 Test Cases

10/26 - 11/1: Scripting framework / 5 Test Cases

11/2 -11/8: Scripting framework

11/9 - 11/15: Scripting framework / Finish Deliv #4 / Injecting faults into code

11/16 - 11/22: Inject 5 faults into code & working on Final Presentation

11/23 - 11/29: Finish Deliverable #5 / Thanksgiving Break!

12/1, 3 - Final Presentations and Final Report

Test Recording Procedures

Test cases will be recorded on a table with columns, test ID, method name, inputs, expected outcomes, actual outputs, and results (pass or fail). Once the tests are completed, they will be outputted html text file and reviewed and checked by all team members.

Hardware and Software Requirements

- Linux
- Python 2.7
- Levenshtein (optional)
- Python IDE

Constraints

Scheduling is a major hindrance affecting the testing process. Since the team members have different schedules that usually do not align, unfortunately only half of the team members were able to meet and work on the project in person. The rest of the collaboration had to be done virtually.

Test Cases

Test ID: 1a
Requirements being tested: The method should format file size in a human readable way.
Component being tested: beets/ui/__ init __
Method being tested: human_bytes(size) // int size Test inputs containing command-line arguments: human_bytes(1023)
Expected Outcome: 1023.0 B // String size

Test ID: 1b
Requirements being tested: The method should format file size in a human readable way.
Component being tested: beets/ui/__ init __

Method being tested: human_bytes(size) // int size Test inputs containing command-line arguments: human_bytes(-1)
Expected Outcome: -1.0 B // String size

Test ID: 1c
Requirements being tested: The method should format file size in a human readable way.
Component being tested: beets/ui/__ init __
Method being tested: human_bytes(size) // int size Test inputs containing command-line arguments: human_bytes(0)
Expected Outcome: 0.0 B // String size

Test ID: 1d
Requirements being tested: The method should format file size in a human readable way.
Component being tested: beets/ui/__ init __
Method being tested: human_bytes(size) // int size Test inputs containing command-line arguments: human_bytes(2000000000)
Expected Outcome: 1.9 GB // String size

Test ID: 1e
Requirements being tested: The method should format file size in a human readable way.
Component being tested: beets/ui/ __ init __
Method being tested: human_bytes(size) // int size Test inputs containing command-line arguments: human_bytes(None)
Expected Outcome: TypeError

Experiences

Beets was extremely well documented and had great instructions for cloning and building the source code into Ubuntu. All that was necessary was cloning the GitHub repository and installing pip. All team members were able to build beets correctly. Beets became frustrating once we started looking deeply at the code because many of the methods modify and use databases. After analyzing each method and deciding we should make sure there were several testable ones, we finally found five methods, three understandable and two more complex. The challenging part of this deliverable was to decide that we were definitely changing projects. Initially, we were unsure of where to begin this deliverable; however, in researching and looking through the website provided on the syllabus, it became clear how to format correctly a test plan. This was a definitely a learning experience. It has taught us that, when pressed for time to meet a deadline, making good, fast decisions and changing from OTP to beets was necessary in order to complete the deliverable and move on in our team project.

References

Resources used to complete the test plan: http://iansommerville.com/software-engineering-book/web/test-planning/

Chapter Three

Testing Framework Architecture

Team Progress Experience

Our team is working well since we have a better grasp of the project and each other's skills. We decided as a team that, since we all knew Python, it would be best to capitalize on that skill. This made it easier to understand what our code was doing. Additionally, our team has emphasized each other's strengths by dividing some of the work. For example, we each wrote test cases. So far, this project has definitely helped in learning how to collaborate with others on a project and how important it is to have a team leader to keep everyone on task. As planned out in our Deliverable 2, our detailed week-by-week schedule has helped us not fall behind on what is required with each deliverable.

Architectural Description of Framework

Our full script plan is to read the file, located in /testCases, and populate a list for each test case. The script then parses the file, meaning the script partitions the lines in the test case file to locate the code to be tested, the driver to be used, and the information needed for the report. It also handles importing the driver and the method to be tested. After that, it is sent to the driver and calls the functions from parse. The script executes the function specified inputs retrieved by parse function according to the test case. A compare helper method compares the results with the expected results, the oracle, and saves results in the test report. Then, the script uses the information, outputs to the report function, and generates a Html document table in a web browser. The testing framework will be called by a single script using ./scripts/runAllTests.sh and will access the test case folder of test case specifications, which has a single test case specification file for each test case. These specification files follow a strict specification format by following the template. Each test case specification file has the data to start and execute the test case and to collect the results of the test case execution of whether it passed or failed.

Framework Directory Structure

TestAutomation:
/project
/scripts runAllTests.sh
tests.py

/testCases → contains 25 test cases
testCase1.txt
testCase2.txt
...
/testCasesExecutables
testCase1to25

/temp (for output from running tests ... cleans at the start of runAllTests)
testCase1results (might be folder or file)
...
/oracles → (contains expected output)
testCase1Oracle (might be folder or file)
...
/docs → (deliverables and readme files)
README.txt

/reports → (html report document)
testReport.html

How-To Documentation

Prerequisites: Python 2.7+

The first step is to clone the TeamIsntThisFun repository. This is done using the command:

"git clone git@github.com:https://github.com/CSCI-362-03-2015/TeamIsntThisFun.git"

Afterwards, navigate to the top-level directory in the cloned repository /TeamIsntThisFun. The next step is to clone the repository located at Beets Github into the /project/src folder in the cloned TeamIsntThisFun repository. This is done using the command:

"git clone git@github.com:https://github.com/sampsyo/beets.git /project/src"

To run the automated testing framework navigate to the top-level directory /TeamIsntThisFun.

With these commands, "chmod +x runAllTests.py" and "./scripts/runAllTests.py", the testing framework should execute automatically and output a test report in a web browser.

Chapter Four

Updating Testing Framework and Finishing the Test Cases

Updates and Changes

We made several updates and changes since the last deliverable. Initially we added the last ten test cases. Then, we formatted the html page so that the pass/fail was easily readable. We found a sort function for our test cases and they now display in the proper order. We also updated deliverable 3 to include more in depth how-to documentation. Additionally, we changed our code so that it could handle multiple input values. Our report generating functionality uses various Strings to generate an HTML page, and these are now defined where they are needed instead of at the top of our script.

Analysis of our Twenty-Five Test Cases

Twenty of our test cases have been specified correctly. Initially, the fifth method we chose to test decoded mp3 sound files, but decided to choose something less complicated (_boo_fallback) so that we would not spend too much time developing a method to obtain a proper oracle. Assertion errors and Unicode strings gave us some trouble in some of Beets methods, but we were able to figure it out. Our test cases specified the following information, separated by lines, to be parsed by our script:

1. Test ID
2. Requirement being tested
3. Component being tested
4. Method being tested
5. Input
6. Expected output
7. Driver name
8. Input type

Tested Items

The methods of the project, Beets, that are to be tested are as follows.
Methods and Locations:

- human_bytes(size) → beets/ui/*init*.py
- human_seconds( ) → beets/ui/*init*.py
- human_seconds_short ( ) → beets/ui/*init*.py
- _sc_decode(soundcheck) → beets/mediafile.py
  We replaced this original chosen method and chose: *_bool_fallback --> beets/ui/_init.py*
- _string_dist_basic(str1, str2) → beets/autotag/hooks.py

Team Experiences

This Deliverable, overall, was a great experience. Since we now have a good grasp on how the architectural framework should run and how to script in python, we were able to make the necessary changes required to implement fully the testing framework. It helped that, after we presented our deliverable 3, we created a to-do list so we knew what needed to be completed. This kept all team members on the right track. After finishing the other ten test cases, we tried to sort and format how it was outputted onto the html because oddly it was outputting it in a random order. This gave us trouble because we found an example sort code in jquery and did not know how to call it correctly, but we were able to find a function that would sort it for us. Another aspect that we found somewhat difficult was how we were going to generalize our parse function to take multiple inputs. In-person meetings have been the most challenging aspect, in terms of organization. It has been very difficult to find a time to get all team members together. But, class time has helped tremendously to be able to talk face-to-face. We are far more productive when working in a group, because communication is easier and we can all look at the same computer screen. Fortunately, this would not be so much of an obstacle for a real development team. By implementing the to-do list mentioned earlier, we were able to maintain some organization and complete separate tasks remotely. We started creating Google Docs for deliverables so that team members can work together easily.

Chapter Five

Fault Injection

For deliverable 5, team IsntThisFun designed and injected five faults into Beets' source code, which we planned to cause at least five tests to fail, but not all, following the deliverable specifications. We decided to inject one fault into each of our five methods.

The test cases that we planned would fail were 1d, 2a-c, 2e, 3a-b, 3e, 4c-d,5a-c.

For the first fault, our team changed code within the method, human_bytes. Within this function, we changed the string which indicates the memory size unit from 'iB' to 'iBytes'.

Fault 1

Original Code	Code After Fault

def human_bytes(size):
"""Formats size, a number of bytes, in a human-readable way."""
powers = ['', 'K', 'M', 'G', 'T', 'P', 'E', 'Z', 'Y', 'H']
unit = 'B'
for power in powers:
if size < 1024:
return "%3.1f %s%s" % (size, power, unit)
size /= 1024.0
unit = 'iB'
return "big"
	def human_bytes(size):
	"""Formats size, a number of bytes, in a human-readable way."""
	powers = ['', 'K', 'M', 'G', 'T', 'P', 'E', 'Z', 'Y', 'H']
	unit = 'B'
	for power in powers:
	if size < 1024:
	return "%3.1f %s%s" % (size, power, unit)
	size /= 1024.0
	unit = 'iBytes'
	return "big"

This change affects all tests cases with a memory unit value above 1023. Therefore test case 1d fails, as shown below, because the expected value '1.9 GiB' is not equal to the new value of '1.9 GiBytes'.

Original Result

1d	The method human_bytes should format file size in a human readable way as a string (e.g. "1.9 GB").	ui	human_bytes	2000000000	1.9 GiB	1.9 GiB	Pass

Result After Fault

1d	The method human_bytes should format file size in a human readable way as a string (e.g. "1.9 GB").	ui	human_bytes	2000000000	1.9 GiB	1.9 GiBytes	Fail

---

For the second fault, our team changed code within the method, human_seconds. Within this function, we changed it to increment units of time incorrectly.

Fault 2

Original Code

```
def human_seconds(interval):
    """Formats interval, a number of seconds, as a human-readable time
    interval using English words.
    """
    units = [
        (1, 'second'),
        (60, 'minute'),
        (60, 'hour'),
        (24, 'day'),
        (7, 'week'),
        (52, 'year'),
        (10, 'decade'),
    ]
    for i in range(len(units) - 1):
        increment, suffix = units[i]
        next_increment, _ = units[i + 1]
        interval /= float(increment)
        if interval < next_increment:
            break
    else:
        # Last unit.
        increment, suffix = units[-1]
        interval /= float(increment)

    return "%3.1f %ss" % (interval, suffix)
```

Code After Fault

```
def human_seconds(interval):
    """Formats interval, a number of seconds, as a human-readable time
    interval using English words.
    """
    units = [
        (1, 'second'),
        (60, 'minute'),
        (60, 'hour'),
        (24, 'day'),
        (7, 'week'),
        (52, 'year'),
        (10, 'decade'),
    ]
    for i in range(len(units) - 1):
        increment, suffix = units[i]
        next_increment, _ = units[i + 1]
        interval /= float(increment)
        if interval > next_increment:
            break
    else:
        # Last unit.
        increment, suffix = units[-1]
        interval /= float(increment)

    return "%3.1f %ss" % (interval, suffix)
```

This change affects all tests cases with a number output value. Therefore test case 2a, 2b, 2c, and 2,e fail. Instead of outputting the next highest unit of time, it will only now designate it as seconds. This causes all test cases to fail that were previously passing.

Original Result

2a	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	600	10.0 minutes	10.0 minutes	Pass
2b	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	0	0.0 seconds	0.0 seconds	Pass
2c	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	583	9.7 minutes	9.7 minutes	Pass

2d	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	-55	TypeError	-55.0 seconds	Fail
2e	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	60.5	1.0 minutes	1.0 minutes	Pass

Result After Fault

2a	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	600	10.0 minutes	600.0 seconds	Fail
2b	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	0	0.0 seconds	0.0 decades	Fail
2c	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	583	9.7 minutes	583.0 seconds	Fail
2d	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	-55	TypeError	-0.0 decades	Fail
2e	The method human_seconds should take in an integer corresponding to a number of seconds and output a human readable string displaying the time interval using English words.	ui	human_seconds	60.5	1.0 minutes	60.5 seconds	Fail

For the third fault, our team changed code within the method, human_seconds_short. Within this function, we add a line which truncates the interval to create an hour field. We then added the hour field to the return value by adding it the the string formatting.

Fault 3

Original Code	Code After Fault
def human_seconds_short(interval): """Formats a number of seconds as a short human-readable M:SS string. """ interval = int(interval) return u'%i:%02i' % (interval // 60, interval % 60)	def human_seconds_short(interval): """Formats a number of seconds as a short human-readable H:M.SS string. """ interval = int(interval) hour = interval // 3600 return u'%i:%i.%02i' % (hour, interval // 60, interval % 60)

This change affects all tests cases which output a value in the M:SS format specified. Therefore test cases 3a, 3b, and 3e fail because the expected values in M:SS format are not equal to the new format of H:MM.SS.

Original Result

3a	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	0	0:00	0:00	Pass
3b	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	153	2:33	2:33	Pass
3c	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	-55	TypeError	-1:05	Fail
3d	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	None	TypeError	Error	Fail
3e	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	5.5	0:05	0:05	Pass

Result After Fault

3a	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	0	0:00	0:0.00	Fail
3b	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	153	2:33	0:2.33	Fail
3c	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	-55	TypeError	-1:-1.05	Fail
3d	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	None	TypeError	Error	Fail
3e	The method should take in an integer corresponding to a number of seconds and output a short human-readable M:SS string displaying the time.	ui	human_seconds_short	5.5	0:05	0:0.05	Fail

For the fourth fault, our team changed code within the method, _bool_fallback. Within this function, we swapped the return values within the if and else test.

Fault 4

Original Code	Code After Fault

def _bool_fallback(a, b):
"""Given a boolean or None, return the original value or a fallback.
"""
if a is None:
assert isinstance(b, bool)
return b
else:
assert isinstance(a, bool)
return a
	def _bool_fallback(a, b):
	"""Given a boolean or None, return the original value or a fallback.
	"""
	if a is None:
	assert isinstance(b, bool)
	return a
	else:
	assert isinstance(a, bool)
	return b

This change affects all tests cases which return a value instead of an error. Therefore test case 1d fails because the expected value '1.9 GiB' is not equal to the new value of '1.9 GiBytes'.

Original Result

4a	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	None, 1	AssertionError	AssertionError	Pass
4b	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	55, False	AssertionError	AssertionError	Pass
4c	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	None, True	True	True	Pass
4d	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	False, True	False	False	Pass
4e	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	None, None	AssertionError	AssertionError	Pass

Result After Fault

4a	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	None, 1	AssertionError	AssertionError	Pass
4b	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	55, False	AssertionError	AssertionError	Pass
4c	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	None, True	True	None	Fail
4d	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	False, True	False	True	Fail
4e	Given a boolean or None, return the original value or a fallback.	ui	_bool_fallback	None, None	AssertionError	AssertionError	Pass

For the fifth fault, our team changed code within the method, string_dist_basic. Within this function, we added an illegal cast from float to int.

Fault 5

Original Code	Code After Fault

def _string_dist_basic(str1, str2):
"""Basic edit distance between two strings, ignoring non-alphanumeric characters and case. Comparisons are based on a transliteration/lowering to ASCII characters. Normalized by string length.
"""
assert isinstance(str1, unicode)
assert isinstance(str2, unicode)
str1 = unidecode(str1).decode('ascii')
str2 = unidecode(str2).decode('ascii')
str1 = re.sub(r'[^a-z0-9]', '', str1.lower())
str2 = re.sub(r'[^a-z0-9]', '', str2.lower())
if not str1 and not str2:
return 0.0
return levenshtein_distance(str1, str2) / float(max(len(str1), len(str2)))
	def _string_dist_basic(str1, str2):
	"""Basic edit distance between two strings, ignoring non-alphanumeric characters and case. Comparisons are based on a transliteration/lowering to ASCII characters. Normalized by string length.
	"""
	assert isinstance(str1, unicode)
	assert isinstance(str2, unicode)
	str1 = unidecode(str1).decode('ascii')
	str2 = unidecode(str2).decode('ascii')
	str1 = re.sub(r'[^a-z0-9]', '', str1.lower())
	str2 = re.sub(r'[^a-z0-9]', '', str2.lower())
	if not str1 and not str2:
	return 0.0
	return int(levenshtein_distance(str1, str2) / float(max(len(str1), len(str2))))

This change affects all tests cases that return a float, such as 5a, 5b, and 5c. Entering ints or nulls will still result in a pass.

Original Result

5a	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	blue, red	1.0	1.0	Pass
5b	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	kites, rites	0.2	0.2	Pass
5c	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	Supercalifragilisticexpialidocious, Antidisestablishmentarianism	0.764705882353	0.764705882353	Pass
5d	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	None, None	AssertionError	AssertionError	Pass
5e	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	1, Hello	AssertionError	AssertionError	Pass

Result After Fault

5a	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	blue, red	1.0	1	Fail
5b	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	kites, rites	0.2	0	Fail
5c	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	Supercalifragilisticexpialidocious, Antidisestablishmentarianism	0.764705882353	0	Fail
5d	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	None, None	AssertionError	AssertionError	Pass
5e	The method should take in two strings as input and output basic string distance normalized by string length formed by a Levenshtein calculation as a float.	autotag	_string_dist_basic	1, Hello	AssertionError	AssertionError	Pass

Team Experiences and Lessons Learned

Meeting face-to-face is still a struggle, but we have managed it quite well. It has helped that we have been coordinating online meetings through Facebook. Because of how much we have learned throughout the CSCI 362 project, Deliverable 5 was easier for us. We learned that sometimes the fewer the team members the better. Additionally, we learned how one simple change can drastically affect the test results and we learned what kind of faults to inject to test code.

Chapter Six

Our overall team experiences in completing this project was that it was difficult and challenging at times, but it has helped us in understanding other programmer's code, working with GitHub, and understanding people's skills is important to get tasks accomplished quickly. Documentation is important. We learned that sometimes projects and the assignments cannot be completed by just one person; however, it is difficult to trust and rely on other people. Therefore, we learned that a team has to be able to trust each other fully. We each learned a lot individually in executing this project.

Individual Team Members' Overall Experiences

Ben Byrd: The most valuable aspect of this project for me was becoming familiar with Version Control through GitHub. I previously had no idea how to contribute to an open source project, and knew that employers look at GitHub to evaluate their applicants' expertise. I learned how to build a project from its source, and developed some shell scripting ability. I also became acquainted with Ubuntu, whereas before I had never used Linux at all. I got the opportunity to refresh my Python skills as well, and learned how valuable a language it is for open source projects, since one does not need to build actually a project they clone if it is written in python. I also enjoyed working with a team over a semester on a large project. This was a great experience because in the software industry it is very difficult to succeed without well-developed communication, long-term organization, and cooperative skills. We encountered several difficulties, which were also excellent learning opportunities. We had to switch completely which open source project we were testing, and once we had decided which one we wanted, had to change which methods we were testing. Meeting face to face was also a challenge, because our group members had busy schedules that often conflicted.

Zach Davis: This project has served to be very valuable in many ways. For instance, I have already used the skills and experiences from this project on my resume and in interviews. I gained practical knowledge in many areas, such as Linux and shell scripting, and reinforced many others, such as GitHub and Python. Before this project, I had not used much Python since Intro to Programming. Python is a valuable language, and I am happy to have been able to revisit it. After this project, I am very comfortable writing Python code again, as well as using the language's documentation. From our first H/FOSS project, OpenTripPlanner, I gained valuable experience building code repositories from source on linux. I also enjoyed working on an open source project because it is a great way to gain software knowledge.

As far as nontechnical skills, I also gained a lot of experience. Throughout the course of the project, I took on an informal leadership role. What this role entailed was delegating tasks, planning meetings, and making design decisions about the project. Ben and Kaitlyn made this easy because they were willing to work hard to complete each deliverable both on time and to a

high standard. There was a challenge with finding common time opportunities for our meetings. We worked through this issue by communicating online and finishing individual. Another challenge our team faced is one of our team members struggled throughout the project to complete his individual tasks. He would also often communicate that he was free for our meeting time and then not attend the meeting. We worked through this difficulty well as a team and were able to overcome it. I believe this was the most valuable part of this project. Working together as a team to overcome problems and complete tasks is the backbone of software development. Over the course of the project our team was required to have strong communication and cooperation and an organized approach to meeting to discuss and complete our tasks. This project has served as an effective way for me to gain experience working in a team environment.

Kaitlyn Fulford: This project has taught me many valuable skills and lessons. We each had designated roles for the project; however, with each deliverable, our roles changed depending on what skills were required. For instance, sometimes, I was the "leader", scheduled meetings, and made decisions about the project or sometimes, another teammate "led" the project. Most interestingly, I definitely learned that I am good at understanding the conceptual details and specifications of a project and this project allowed me to perfect my skills at communicating the details effectively to my teammates. In terms of programming, this course, as a whole, taught me that how important it is to understand the specifications and to have overall plan of how the project will be completed and that the code should only be implemented afterwards. Once I created, with help from Zach, deliverable two with the overall schedule by week for the semester, this helped us know what we needed to complete each week and allowed us not to fall behind. Choosing methods to test was a difficult task; however, I learned that its best to choose the methods that the team understood the most.

Additionally, I have enjoyed working with Python again. This project allowed me to re-familiarize myself with Python since I had not used Python in a while. On the side, I learned simple bash scripting and the command line coding, which was something I had never used before this class. Once I became comfortable with the project, I found myself really enjoying the process of the project and working with teammates. This project gave me a great experience in understanding the real life process of a software engineering project on a small scale. I also learned how to deal with challenges and solve problems in a timely manner.

Adam Sugarman: I think this project helped us all understand the value of design rather than implementation. The majority of the time spent was designing the overall architecture and flow of the testing suite, which made the implementation very straightforward. While implementing the automated testing framework I had to brush up on Python syntax and learned some new things, such as traversing a file structure, dynamically calling methods and importing classes, and file IO. I also learned scripting, script creation, and the relationship to program files for automated programs and products. I gained further experience with GIT and GitHub, which is good since I am not used to either as a source control choice.

Self-evaluation of Our Team's Work

Team Isn'tThisFun has worked hard to complete successfully each deliverable. We were able to work well together and meet regularly once we found common times we could meet in person. As a team, we were able to complete the project assignments more effectively because we assigned specific tasks to each team member. This ensured that we submitted quality deliverables to our team GitHub. For instance, our deliverable with the testing framework architecture was very detailed and we followed it throughout the completion of our framework over the semester. Our primary strength as a team was that we each had specific skills, which used to complete the project to a high standard. These skills helped tremendously on the project as a whole because, while we were staying up to date with the other pieces of the project, we were able to focus on our strengths. As a team, we were great at completing the tasks on time, following the task specifications, adapting to challenges, and choosing important Beets methods to test. Examples of challenges we faced include splitting the driver from the script and changing from our initial choice, Open Trip Planner. The main weakness within the team was communication; however, communication became somewhat better once we created a group message on Facebook. Another aspect was not all team members would come to team meetings. However, even with very limited participation from one teammate, we were able to complete the project and create a working testing framework.

Evaluation of Project Assignments

As a team, as the semester went on, we liked that deliverables were due periodically because this helped us stay on task. If there were no deliverables, it would be very easy to fall behind and wait until the end to start the project. One aspect that we would have liked was more feedback on each deliverable report so that we could have improved them throughout the semester. There definitely was a learning curve with the deliverables at the beginning because we were not sure exactly what was required in them especially for the architectural plan. However, after looking at the previous semester's class for examples, we were able to complete it without issues. We spent a lot of time learning bash, but realized that python would be easier to understand and worked better for our project once we switched to Beets.

There was definitely a learning curve in working with open source projects. The only suggestions we would provide for improvement for the project assignments would be that we needed more explanation in class for choosing an open source project and for completing the first two deliverables.

Appendix

The following are requirements for the installation and use of Beets in Linux:

- Linux
- Python 2.7