Faculty of Engineering & Applied Science

Software Quality

SOFE 3980U - WINTER 2021

Final Project Report

Group 13

Name: Taha Hashmat
Student ID: 100689792

Name: Mitchell Hicks
Student ID: 100707709

Name: Austin Page
Student ID: 100725236

Name: Yakhneshan Sivaperuman
Student ID: 100703400

Table of Contents

I. Details of the Project

For this project we used the agile design process as our design template. We felt that the agile design process was best suited for our team and this project, because even though the requirements of the project did not change, as we kept testing and improving our program we undoubtedly had to revisit some stages of the design process.

During the requirements gathering phase of our project we wanted to make sure that we properly understood what was required in order to achieve success. This led us to find an open source library called Pylightxl (this will be discussed in more detail later on). After picking a library that met all the outinelined requirements we started designing the structure of how we would improve Pylightxls functionality. The first thing we realized was that Pylightxl had a pretty lightweight infrastructure and wasn't documented well. So we decided to create a manual that a new user could use to search up key functions of the program and receive proper instruction on what they are and how they are used. Next, we realized that Pylightxl could save and write contents to a .csv file but not a .txt file, We wanted to fix that. Finally, we found that there were no statistics on the data being imported. This prompted us to make a function that would let the user know how much data they were approximately importing.

After analyzing what we wanted to fix/improve in Pylightxl we started planning how we would code and test our new functionality. For this we used a python, as our programming language, VScode, as coding environment, PyTest, as our unit testing library (which will be discussed further later on), and parameterized (which will be discussed in detail later on), as our data generation software for test cases. After we established what coding environment and testing libraries we were going to use we then moved onto the implementation phase.

Background Application Information

A. Pylightxl

Pylightxl is a lightweight microsoft excel reader/writer. The Pylightxl library allows developers to access data represented in an excel file and make computations on it.

- **Why'd we choose Pylightxl?**
    - Pylightxl is not very popular which allows us to make valuable upgrades to the code
    - Pylightxl is used for data analytics and getting data from excel file, which as a group we are all interested in data analysis
    - Pylightxl is not as complex as a library like Pandas, this will help us focus on a specific improvements without overwhelming us

B. PyTest

PyTest is a testing framework that allows users to write test cases for python programs. Pytest allows users to make simple and scalable test cases.

- **Why'd we choose Pytest?**
    - Pytest allows users to make their test cases as complex as they want and provides more functionality to programmers compared to the built in unit test library.
    - PyTest hass built-in modules that allow users to generate large sets of data quickly to test functions (like parameterize, fixture, etc.)
    - PyTest gives users the ability to implement test cases with large test coverage

C. Parameterize

Parameterize is a built-in module in the PyTest library that gives users a way to quickly generate large amounts of data that can be used for testing.

- **Why'd we choose Parameterize?**
    - Is simple to use, allows programmers to generate lot's of data and is part of the Pytest library

II. Design Choices

Since Pylightxl is not currently very popular or relatively known amongst users, a noticeable lack of features that can vastly improve the quality of the software is noticeable. For the duration of the project, our group focused on four aspects of the code base. These include implementing new features that benefit the users directly, performing test cases on our new features to make sure our changes are efficient and reliable, generating test data for our test cases and finally, making any improvements we can to the new features. In the forthcoming discussion, our group will talk about all these specific changes alongside an explanation of the snapshots of the code.

A. Statistics Feature

- **Reasons for Implementation:**

  As individuals who have a keen interest in the world of data analytics, one of the first changes that our group implemented in PylightXL was the Statistics feature, which gave the user a clearer idea of the data they are working with. Working with data in excel, .csv or .txt files can at times prove to be tricky for the user due to the large amounts of data being processed. In order to ensure a simplified process of figuring out the specifics of the data a user is working with, we implemented a feature that displays the total number of rows and columns for the worksheets in our database.

- **Code Walkthrough: def get_stats(db,workiesheet):**

```
def get_stats(db, workiesheet):

    db.ws(ws=workiesheet)
    rowz = db.ws(ws=workiesheet).rows
    colz = db.ws(ws=workiesheet).cols

    rowcnt = (sum(1 for _ in rowz))
    colcnt = (sum(1 for _ in colz))
    val = [rowcnt, colcnt]
    return val
```

Our get_stats() function takes in 2 arguments and returns a single value. The 2 arguments it takes in are ‘db’, and ‘workiesheet’, where the **‘db’** parameter refers to the PylightXL database in which all of the sheets (excel, csv, txt) are being stored and the **‘workiesheet’** parameter is the name of the

worksheet (**ws**) that the user wants the statistics for. The value being returned **'val'** is the number of rows and columns of the worksheet in the form [row,column]. Our get_stats function initially instructs the program what worksheet to pull from our database using the method **db.ws().** Two variables in **'rowz'** and **colz'** are then defined with rowz returning _____ and colz returning _____ respectively. **'rowcnt'** and **'colcnt'** are then used to calculate the total number of rows and columns in the specific worksheet, with that value, as discussed, is being stored and returned through the **'val'** variable.

- **Test Cases with Placeholder Data:**

```
def test_file_stats_function():

    testdata = db
    sheetname = "Sheet1"
    testresult = xl.get_stats(db, sheetname)
    assert testresult == [5,3], "Test Passed"

    sheet2name = "Sheet2"
    testresult2 = xl.get_stats(db, sheet2name)
    assert testresult2 == [5,5], "Test Passed"

    sheet3name = "Sheet3"
    testresult3 = xl.get_stats(db, sheet3name)
    assert testresult3 == [6,6], "Test Passed"
```

Our test_file_stats() function is used to write test cases for our get_stats() function in our main PylightXL program. We have 3 variables in our test case which are **'testdata'**, **'sheetname'** and **'testresult'**. 'testdata' stores db which is the PylightXL database which stores all of our worksheets and 'sheetname' references the name of the worksheet being stored in the PylightXL database. Both 'testdata' and 'sheetname' are later going to be passed into the get_stats function. 'testresult' stores our method call to the get_stats function in the main pylightxl file which is being referenced to in the test case file as 'xl' (hence xl.get_stats). Our test case finally asserts 'testresult' and the test case passes if the value that is being asserted (testresult) stores the correct number of rows and columns of the worksheet.

- **Test Data Generation and Output:**

```
@pytest.mark.parametrize("data1, data2, expected", [(db, "Sheet1", [5,3]), (db, "Sheet2", [5,5]), (db, "Sheet3", [6,6]), (db, "Sheet4", [7,7]),
(db, "Sheet5", [8,8]), (db, "Sheet6", [9,9]), (db, "Sheet7", [10,10]), (db, "Sheet8", [15,17]), (db, "Sheet9", [12,15]), (db, "Sheet10", [14,11])])

def test_stats_function(data1, data2, expected):
    val = xl.get_stats(data1, data2)
    assert val == expected
```

The test data that will be passed through our test cases is being generated with the help of a built in pytest library called parameterize. As you can see the @pytest.mark.paramertize call in the snapshot above is being used to generate the test data. Since our get_stats function in the original PylightXL file requires two input arguments in db and sheetname, we need to have 2 inputs in our test data as well. These two inputs are then followed by the expected value when the test data is being passed into our test_stats test case. This test data is being written to parameterize in the form of (db (input1), sheetname (input 2), [rowcnt,colcnt] (expected value)). We then use this format to write a large number of test data that is going to be passed into the test case for example: [(db, “Sheet1", [5,3]), (db, “Sheet2”, [5,5]). After all of the test data is generated we include a call to our test_stats() test case which now takes in data1, data2, expected as its arguments where data1 is the first input (db), data2 is the second input (sheetname) and expected is the expected output. Finally the test case asserts whether it has passed or not.

- **Improvements:**

```
35    #start time
36    starthelp2 = time.time_ns()
37
38    #calling help function from pyliht file
39    stat = xl.get_stats(db, "Sheet1")
40    print(stat)
41
42    #end time and caculate total
43    endhelp2 = time.time_ns()
44    totalhelp2 = endhelp2 - starthelp2
45    print("Total Execution Time for Statistics Feature: ", totalhelp2)

PROBLEMS  OUTPUT  DEBUG CONSOLE  COMMENTS  TERMINAL
[Running] python -u "/Users/tahahashmat/Desktop/SQ_Project/env/lib/python3
[5, 3]
Total Execution Time for Statistics Feature:  151000
```

In order to measure the improvements of our features we included timestamps before and after the function call. As you can see in the snapshot, the execution time measured for our get_stats function was 151000 ns or 0.000151s. Our group removed the unnecessary for loops that we had in our initial function, which improved the total execution time of the improved function by a decent amount.

B. A User Manual:

- **Reasons for Implementation:**

Implementing a user manual for the users of PylightXL was one of the main priorities for our group. This is because users that are navigating a software for the first time may face a lot of confusion to the inner workings and fine details of a part of the program that they may wish to utilize. An easy to understand user manual helps ensure customer satisfaction , and an improved customer support experience is what ultimately leads to retention, recommendation, and referral.

- **Code Walkthrough: def help(keyword):**

```
def help(keyword):
    # RETURN TRUE OR FALSE
    if (keyword == "read"):
        print('= READ ==========================================================================================================================')
        print("FUNCTION: readxl\n" +
                "Reads an xlsx or xlsm file and returns a pylightxl database\n\n" +
                ":param str fn        : Excel file name\n" +
                ":param str or list ws: sheetnames to read into the database, if not specified - all sheets are read\n" +
                "                       entry support single ws name (ex: ws='sh1') or multi (ex: ws=['sh1', 'sh2'])\n" +
                ":return              : pylightxl.Database class\n")

        print("FUNCTION: readcsv\n" +
                "Reads an xlsx or xlsm file and returns a pylightxl database\n\n" +
                ":param str fn           : Excel file name\n" +
                ":param str delimiter=',': csv file delimiter\n" +
                "                          entry support single ws name (ex: ws='sh1') or multi (ex: ws=['sh1', 'sh2'])\n" +
                ":param str ws='Sheet1'  : worksheet name that the csv data will be stored in\n" +
                ":return                 : pylightxl.Database class")
        print('==================================================================================================================================')
        return True

    elif (keyword == "write"):
        print('= WRITE =========================================================================================================================')
        print("FUNCTION: writexl\n" +
                "Writes an excel file from pylightxl.Database\n\n" +
                ":param pylightxl.Database db: database contains sheetnames, and their data\n" +
                ":param str/pathlib fn       : file output path\n" +
                ":return                     : None\n")

        print("FUNCTION: writecsv\n" +
                "Writes a csv file from pylightxl database. For db that have more than one sheet, will write out,\n" +
                "multiple files with the sheetname tagged on the end (ex: 'fn_sh2.csv')\n\n" +
                ":param pylightxl.Database db     :\n" +
                ":param str/pathlib/io.StringIO fn: output file name (without extension; ie. no '.csv')\n" +
                ":param str or tuple ws=()        : sheetname(s) to read into the database, if not specified - all sheets are read\n" +
                ":param delimiter=','             : csv delimiter\n" +
                ":return                          : None")
        print('==================================================================================================================================')
        return True
    else:
        print("Invalid keyword")
        return False
```

Our def help() function takes in a single argument and returns a boolean data type of either True or False. The argument that is being taken in is the keyword that a user would search the manual for, when in need for assistance. Our function mainly comprises a nested if statement which contains the key words available for the user to search up. So far the user manual consists of two keywords that the user can search up: “read” or “write”. The reason our manual only has these two words is because PylightXl is a very basic and simplistic program which only allows

the user to read or write to certain file types. Since reading and writing are the two main actions a user can take in the program, our user manual instructs the user on how to make use of these two actions. The first branch of our if statement checks whether the key word entered by the user is equal to “Read”. If that is the case, our function prints out a statement which explains what the read action does and how it can be used by them in our program. Our function then proceeds to return a True value. If the keyword entered is not equal read, the if statement moves onto its second branch where it checks if the keyword is equal to “Write”. If that is the case, our function prints out a statement which explains to the user what the write action does and how it can be used by them in our program. Our function then proceeds to return a True value. However, if the keyword entered by the user is not equal to either “Read” or “Write”, our if statement moves onto its final branch where it informs the user that the key word entered is not a valid keyword and thus returns a False Value.

- **Test Cases with Placeholder Data:**

```
def test_help_function():
    testword = "read"
    testresult = xl.help(testword)
    assert testresult == True, "Test Passed"

    testword1 = "write"
    testresult1 = xl.help(testword1)
    assert testresult1 == True, "Test Passed"

    testword2 = ""
    testresult2 = xl.help(testword2)
    assert testresult2 == False, "Test Passed"

    testword3 = "hello"
    testresult3 = xl.help(testword3)
    assert testresult3 == False, "Test Passed"
```

Our test_help() function is used to write test cases for our help() function in our main PylightXL program. We have 2 variables in our test case: ‘testword’ and ‘testresult’. ‘testword’ is a variable which stores the keyword that the user wishes to search. The variable ‘testword’ is later going to be passed into the help() function in our main PylightXL program. Our final variable ‘testresult’ stores our method call to the help() function in the main pylightxl file which is being referenced to in the test case file as ‘xl’ (hence xl.help()). Our test function then asserts a True or False value depending on whether the keyword in the ‘testword’ variable was in our user manual or not.

- **Test Data Generation and Output:**

```
@pytest.mark.parametrize("data, expected", [("write", True),
    ("read", True), ("", False), ("teahee1234", False), ("pager2345", False), ("mitchypoo3456", False),
    ("teahee1234", False), ("akram", False), ("khalid", False), ("yakho1234", False), ("rav42345", False), ("sabesan3456", False),
    ("hello", False), ("dust", False), ("osford", False), ("camb", False), ("ont", False), ("bc", False),
    ("uxbridge", False), ("pickering", False), ("oshawa", False), ("scarboro", False), ("toronto", False), ("sftwarequality", False), ("sftwarequality", False),
    ("ai", False), ("os", False), ("cn", False), ("econ", False), ("spm", False),])

def test_help_function(data, expected):
    val = xl.help(data)
    assert val == expected
```

The test data that will be passed through our test cases is being generated with the help of a built in pytest library called parameterize. As you can see the @pytest.mark.paramertize call in the snapshot above is being used to generate the test data. Since our help() function in the original PylightXL file requires a single input argument which is the keyword a user searches for, we need to have 1 input in our test data as well. This input is then followed by the value that we expect when the test data is being passed into our test_help() test case. This test data is being written to parameterize in the form of (keyword (input1), True or False (expected value)). We then use this format to write a large number of test data that is going to be passed into the test case for example: [("write, True), ("read", True), ("hello", False)]. After all of the test data is generated we include a call to test_help() which now takes in data and expected as its arguments where data is the input keyword and expected is the expected output. Finally the test case asserts whether it has passed or not by checking if the returned value is equal to the expected value

- **Improvements**

```
35    #start time
36    starthelp = time.time_ns()
37
38    #calling help function from pyliht file
39    userMan = xl.help("taha")
40    print(userMan)
41
42    #end time and caculate total
43    endhelp = time.time_ns()
44    totalhelp = endhelp - starthelp
45    print("Total Execution Time for User Manual Feature: ", totalhelp)
46

PROBLEMS  OUTPUT  DEBUG CONSOLE  COMMENTS  TERMINAL
[Running] python -u "/Users/tahahashmat/Desktop/SQ_Project/env/lib/python3.
Invalid keyword
False
Total Execution Time for User Manual Feature:  46000
```

In order to measure the improvements of our features we included timestamps before and after the function call. As you can see in the snapshot, the execution time measured for our help function was 46000 nano seconds or 0.000046 seconds. The improvement for the help function was minimal since the function consisted only of if statements and print statements which are pretty efficient.

C. New File Type to Write to

- **Reasons for Implementation:**

One of the main documented limitations of PylightXL was the fact that there were only a limited number of file types that a user could edit. Currently, a user of PylightXL is only allowed to edit an Excel file or a file type with the .csv extension. Our group recognized this limitation as a major flaw in the program and decided to add a feature which would let users edit files with the .txt extension as well. Textfiles are becoming increasingly common by the day and the added feature of letting users edit a .txt file will allow Pylight Xl to expand their customer base as well as making sure that already existing users are more than satisfied with the increased options.

- **Code Walkthrough def writetxt(db,fn):**

```
def writetxt(db, fn):
    # test that file entered was a valid excel file
    if 'pathlib' in str(type(fn)):
        fn = str(fn)

    worksheets = db.ws_names

    for sheet in worksheets:
            # StringIO will keep on appending each worksheet to the same StringIO
            if '_io.StringIO' not in str(type(fn)):
                new_fn = fn + '_' + sheet + '.txt'

            try:
                if '_io.StringIO' not in str(type(fn)):
                    f = open(new_fn, 'w')
                else:
                    f = fn
            except PermissionError:
                # file is open, adjust name and print warning
                print('pylightxl - Cannot write to existing file <{}> that is open in excel.'.format(new_fn))
                print('     New temporary file was written to <{}>'.format('new_' + new_fn))
                new_fn = 'new_' + new_fn
                f = open(new_fn, 'w')
            finally:
                max_row, max_col = db.ws(sheet).size
                for r in range(1, max_row + 1):
                    row = []
                    for c in range(1, max_col + 1):
                        val = db.ws(sheet).index(r, c)
                        row.append(str(val))

                    delimiter = ","

                    if sys.version_info[0] < 3:
                        f.write(unicode(delimiter.join(row)))
                        f.write(unicode('\n'))
                    else:
                        f.write(delimiter.join(row))
                        f.write('\n')
                # dont close StringIO thats passed in by the user
                if '_io.StringIO' not in str(type(fn)):
                    f.close()


    if os.path.isfile(fn + '_' + sheet + '.txt'):
        return True
    else:
        return False
```

Our def writetxt() function takes in two arguments and returns a boolean datatype of either True or False. The 2 arguments it takes in are ‘db’, and ‘fn’, where the **‘db’** parameter refers to the PylightXL database in which all of the sheets (excel, csv, txt) are being stored and the ‘fn’ parameter refers to the filename that is later going to be edited.

The algorithm of this function is actually quite simple. It first checks to see if the file is the correct format, and then loads all the sheets from the excel file into an array. That array is then looped through, sheet by sheet and the data is extrapolated. It is done by using a built-in function to find the size of the used rows and columns in that sheet, and then they are looped through by the row first, and column second. It is then written to a .txt file that was created and opened at the start of the loop, and we use a hard-coded comma as a delimiter for the data.

- **Test Cases: def test_file_stats_function():**

```
def test_writetofile():
    filename = "pager123"
    val = xl.writetxt(db, filename)
    assert val == True

    filename = "teahee123"
    val = xl.writetxt(db, filename)
    assert val == True

    filename = "mitchy123"
    val = xl.writetxt(db, filename)
    assert val == True

    filename = "yakho123"
    val = xl.writetxt(db, filename)
    assert val == True
```

Our test_writetofile(): function is used to generate test cases for our writetxt() function. The writetofile test case contains two variables: ‘filename’ and ‘val’. ‘filename’ stores the name of the file that requires editing and our final variable ‘val’ stores our method call to the writetxt() function in the main pylightxl file which is being referenced to in the test case file as ‘xl’. The method being used to write to the file is xl.writetxt, which calls the writetxt function and takes in two arguments: “filename’ and db, which is the PylightXL database which stores all of our worksheets. Our test function then asserts a True or False value depending on whether our writetxt function was able to write to a .txt file type.

- **Test Data Generation and Output:**

```
@pytest.mark.parametrize("data1, data2, expected", [(db, "teahee123", True), (db, "yakho123", True), (db, "austin123", True), (db, "mitchy123", True),
(db, "rav4", True), (db, "sabesan123", True), (db, "akram", True), (db, "owais1234", True), (db, "khalid", True), (db, "anwar", True)])

def test_writetofile(data1, data2, expected):
    val = xl.writetxt(data1, data2)
    assert val == expected
```

The test data that will be passed through our test cases is being generated with the help of a built in pytest library called parameterize. As you can see the @pytest.mark.paramertize call in the snapshot above is being used to generate the test data. Since our writetxt() function in the original PylightXL file requires two input arguments in db and filename, we need to have 2 inputs in our test data as well. These two inputs are then followed by the value that we expect when the test data is being passed into our test_stats test case. This test data is being written to parameterize in the form of (db (input1), filename (input 2), True or False (expected value)). We then use this format to write a large number of test data that is going to be passed into the test case for example: [(db, “File1", True), (db, “File2", True). After all of the test data is generated we include a call to our test_writetofile() test case which now takes in data1, data2, expected as its arguments where data1 is the first input (db), data2 is the second input (filename) and expected is the expected output. Finally the test case asserts whether it has passed or not.

- **Improvements**

```
35    #start time
36    starthelp3 = time.time_ns()
37
38    #calling help function from pyliht file
39    newfile = xl.writetxt(db, "NewFile")
40    print(newfile)
41
42    #end time and caculate total
43    endhelp3 = time.time_ns()
44    totalhelp3 = endhelp3 - starthelp3
45    print("Total Execution Time for New File Type Feature: ", totalhelp3)
46

PROBLEMS   OUTPUT   DEBUG CONSOLE   COMMENTS   TERMINAL
[Running] python -u "/Users/tahahashmat/Desktop/SQ_Project/env/lib/python3.8/s
True
Total Execution Time for New File Type Feature:  4531000
```

As you can see in the snapshot, the execution time measured for our new and improved writetxt () function was 4531000 nano seconds or 0.00453 seconds. This is an impressive result considering the fact that this function creates a .txt file and edits the contents in it as well.

III. Implementation Challenges - Austin

Although our group was able to integrate new functionality to the PylightXL library, while also improving the usability, it didn't come easy. Improving on other developer's published work is always a chore because if it was released to the public and the average person is able to find it online, then chances are it was developed well and with little room for improvement. This is the exact reason we stayed away from larger projects like Pandas; it's unlikely a team of junior developers can improve years of hard work and constant tweaking. With that said, PylightXL falls into the smaller project category. It's light-weight with very few functions.We believed that PylightXL was written very well and served a useful purpose, but could use some quality of life improvements and some added functionality. This led us to creating 3 functions: a help function, a write-to-text function, and a get statistics function.

HELP FUNCTION

Although this function is fairly simple and very light-weight, there were still some challenges to overcome. We had to implement something that would be used by users of any skill level, while still being easy to access and understand. For this reason, we made this function output useful information in the console via a print statement, and included the core functions of the library, as well as any parameters, their type, and what each function returns.

WRITE-TO-TEXT FUNCTION

This function was the heaviest function that we wrote, and the main difficulties here would be trying to keep it efficient. Since this library is cell-based (it works off columns and rows, and their intersections), there can be a fair amount of looping to find and gather data. Our first few iterations of this function were quite slow and inefficient, but in the end we were able to to tighten things up.

GET STATS FUNCTION

This function is also very light-weight, but was a little trickier to approach. We needed to find data that was simple to understand, yet effective for any skill level of developer. At first we took a brute-force method to try to find the rows and columns, but later on we learned that the library already had built-in methods that made gathering that information a little easier.

IV. Lessons Learned

Implementing changes in the PylightXL library was the first time that a majority of our group members had worked on an open source software. For the duration of this project and course, our group has had the privilege of inheriting quite a significant amount of knowledge, from following certain software design processes such as the agile design process, to writing automated test cases and performing static and dynamic analysis on our code. In the forthcoming discussion, we talk about the major lessons our group learned from this project and how these lessons will help shape our future as software engineers.

1. We learned how to follow a certain type of software design process. As mentioned earlier, our group followed the agile design process for this project. By using the agile method, our group learned how to quickly complete our tasks in short coding bursts and make sure that time management was our utmost priority. We also learned to improve our end product with every iteration by using timestamps on our code to measure our functions execution times.

2. Our group learned how to take advantage of built in libraries to help improve the efficiency of our entire code base. The two biggest examples include using pytest and parameterized. Having never written test cases for our program prior to this course, learning how to improve the quality of our code and drastically reducing the amount of bugs or errors within our program has definitely got to be one of the biggest takeaways from this course. Furthermore, using pytest libraries such as parameterized helped us in generating test data for our test cases. Since this was a part of the code that our group had to do manually before, using parameterized was instrumental in increasing our groups time efficiency and thus output.

3. Finally let's end this section talking about the most significant takeaway from this project. Our group, as software developers, can now confidently believe in our ability to work on and improve many of the open source projects in the computer science realm.