Introduction to Software Engineering

Final Assignment Report

Buruvuru Emmanuel ID 20750194

23/05/2022

Introduction

This report addresses aspects of software production covering test implementation, version control, black box testing, white box testing, modularity and lastly ethics. Code implementations include code for the category one that has functions to convert a given string to Upper case and to lower case, to determine if the string contains a numeric value and to remove a numeric value in each string and then convert that string to upper case. Category two covers the implementation of conversion of the time unit from Hours to Minutes and vise-versa and Minutes to Seconds and vise-versa. All the code where then tested either using white box testing or black box testing after they had passed the modularity requirement check list. A discussion is later reviewed at the end of the report on what the program could have done to improve the code for future improvements.

Preliminary Descriptions

Category 1

Catergory1.py - A file that contains a class to convert a string to various variations and edit the string. It is also used to verify if a string contains numeric values.

Function1

- Name: UppertoLower.
- Description: Used to convert a string from lowercase to uppercase. Input: Takes in a string parameter.
- Output: Returns a string.
-

Function2

- Name: LowertoUpper()
- Description: Used to convert a string from Uppercase to lower case. Input: Takes in a string parameter.
- Output: Returns a string

Function3

- Name: NumericsinString()
- Description: used to see if a string has a digit contained in it.
- Input: Takes in a string parameter.

- Output returns a Boolean (True / False).

Function4

- Name: IsStringValid_number()
- Description: used to see if a string is a valid number. Input: Takes in a string parameter
- Output: It returns a Boolean (String is a valid number. / String is not a valid number.).

Function5

-
- Name: remove_numerics()
- Description: used to remove a numeric value in a string and convert that string to upper case via user input. Input: Takes in a string parameter
- Output: returns the inputted string as uppercase.

Category 2

Catergory2.py - A file that contains a class that converts given time from Hours to minutes and vise-versa and Minutes to seconds and vis-versa.

Function1

- Name: HrsToMinMinToSec()
- Description: Converts hours to minutes and minutes to seconds, it checks if the number is not a negative number, and the inputted number is not equal to zero and then multiply the time unit with 60.
- Input: keyboard input or passed as a parameter in the function. Output: returns a float value of the conversion.

Function2

- Name: MinToHrsSecToMin()
- Description: Converts a minute to hours and seconds to minutes, it checks if the number is not a negative number, and the inputted number is not equal to zero and then divides the time unit by 60.
- Input: keyboard input or passed as a parameter in the function. Output: returns a float value of the conversion.

Function3

- Name: SaveData()
- Description: function that takes in four arguments (filename, Hours, minutes, seconds) and then uses the functions to do the conversions and save the results in the file name.
- input: passed as a parameter in the function.
- Output: writes floats into a filename with the conversion units.

Modularity

Modularity checklist

1. Is the system free of global variables -> No global variables in the submodules?
2. Is each submodule free of control flags – > The submodules are free from control flags.
3. Does each submodule perform a well-defined task -> Yes?
4. Do each submodule Deal with the same data -> Yes.

5. If it’s free of duplicate modules –> The submodules have duplicate code in category2.py conversions.
6. Do submodules do not perform overlapping tasks-> Yes, submodules don't perform overlapping tasks.

Fixing Duplicate code

```python
class category2:

    def HourstoMinutes(time: int):

        conversion = time * 60

        return conversion

    def MinuestoHours(time: int):

        conversion = time / 60

        return conversion

    def MinutestoSecond(time: int):

        conversion = time * 60

        return conversion

    def secondstoMinutes(time: int):

        convesion  = time / 60

        return convesion
```

The code had duplicate codes and this was not satisfying the checklist, so the code was fixed to fix this modularity error.

```python
#Author : Emmanuel Buruvuru
#Catergorey2.py : for the version control of the assignment
#Refrences:
#


class category2:

    def HrsToMinMinToSec(time: int):   #mutiplying time

        conversion = time * 60

        return conversion

    def MinToHrsSecToMin(time: int):  #divinding time

        conversion = time / 60

        return conversion
```

The code was then fixed so that it does not contain duplicate codes. The function that had the duplicate codes where merge into one to archive a more efficient code we just must change the code that we have edited in this section.

The modularity check list was passed and then the production code was ready.

Black Box test case

Equivalence Testing.

LowerCase to UpperCase function.

Category	Test Data	Expected data
lowerCase string	emmanuel buruvuru	EMMANUEL BURUVURU
UpperCase string	BURUVURU	BURUVURU
Not a string	0194	""
Lower and upper	Doctor Strange	DOCTOR STRANGE

UpperCase to LowerCase function.

Category	Test Data	Expected data
lowerCase string	emmanuel buruvuru	emmanuel buruvuru
UpperCase string	BURUVURU	buruvuru
Not a string	0194	""
Lower and upper	Doctor Strange	doctor strange

Is String a valid number function.

Category	Test Data	Expected data
lowerCase string	emmanuel buruvuru	"String is not a valid number.."
UpperCase string	BURUVURU	"String is not a valid number.."
Valid number	0194	"String is a valid number.."
String with a number	Doctor Strange2	"String is not a valid number.."
Valid number with letter	10.e	"String is a valid number"

Remove numbers in a string and change string to UpperCase function.

Category	Test Data	Expected data
String without a number	emmanuel buruvuru	"final string: EMMANUEL BURUVURU"
A number	0194	"final string: "
Lower and upper string with number	Doctor Strange2	"final string: DOCTOR STRANGE"

Numeric values in a string function.

Category	Test Data	Expected data
String without numerics	emmanuel buruvuru	"False"
UpperCase string	BURUVURU	"False"
A valid number	0194	"True"
String with a numeric value	Doctor Strange2	"True"

Boundary value analysis

HrsToMinMinToSec function.

Boundary	Test Data	Expected Result
less than zero/sero	-1 / 0	"input a positive int" \ "Number should not be zero"
zero/valid	0 / 1 hour	"Number should not be zero" \ "60"
valid hours/ valid minutes	2 hours / 120min	120 / 7200

MinToHrsSecToMin function.

Boundary	Test Data	Expected Result
less than zero/zero	-1 / 0	"input a positive int" \ "Number should not be zero"
zero/valid	0 / 1 min	"Number should not be zero" \ "60 "
valid hours/ valid minutes	2 hours / 120min	"120" / "7200"

White Box test case

White box testing was implemented on the Module Catergory1.py on two functions. The testing was done by taking the keyboard input and the asserting if the function will return the anticipated output.

Testing the function numericstring from the modulue catergory.py

```
def testNumericsinstring(self):

    #testing with a valid number
    captureOut = io.StringIO()
    sys.stdout = captureOut
    sys.stdin = io.StringIO("25")
    Category1.IsStringValid_number("25")
    self.assertEqual("String is a valid number..\n",captureOut.getvalue())
    sys.stdout = sys.__stdout__
    sys.stdin = sys.__stdin__

    #testing with a string
    captureOut = io.StringIO()
    sys.stdout = captureOut
    sys.stdin = io.StringIO("emmanuel buruvuru")
    Category1.IsStringValid_number("emmanuel buruvuru")
    self.assertEqual("String is not a valid number..\n",captureOut.getvalue())
    sys.stdout = sys.__stdout__
    sys.stdin = sys.__stdin__

    #testing with a valid number that contains a letter
    captureOut = io.StringIO()
    sys.stdout = captureOut
    sys.stdin = io.StringIO("10.e")
    Category1.IsStringValid_number("10.e")
    self.assertEqual("String is a valid number..\n",captureOut.getvalue())
    sys.stdout = sys.__stdout__
    sys.stdin = sys.__stdin__
```

Testing the function remove_numerics from module Catergory.py

```
    def testRemovenumricsfromstring(self):

        #string without numerics
        captureOut = io.StringIO()
        sys.stdout = captureOut
        sys.stdin = io.StringIO("emmanuel buruvuru")
        Category1.remove_numeric("emmanuel buruvuru")
        self.assertEqual("final string: EMMANUEL BURUVURU\n",captureOut.getvalue())
        sys.stdout = sys.__stdout__
        sys.stdin = sys.__stdin__

        # testing with a string that has numeric values
        captureOut = io.StringIO()
        sys.stdout = captureOut
        sys.stdin = io.StringIO("Doctor strange2")
        Category1.remove_numeric("Doctor Strange2")
        self.assertEqual("final string: DOCTOR STRANGE\n",captureOut.getvalue())
        sys.stdout = sys.__stdout__
        sys.stdin = sys.__stdin__

        #testing with numeric values only

        captureOut = io.StringIO()
        sys.stdout = captureOut
        sys.stdin = io.StringIO("0194")
        Category1.remove_numeric("0194")
        self.assertEqual("final string: \n",captureOut.getvalue())
        sys.stdout = sys.__stdout__
        sys.stdin = sys.__stdin__
```

Results

Results generated by running the Unit test testing framework imported in the testing module as Testcase:

```
emmanuelburuvuru@Emmanuels-MacBook-Pro code % python3 -m unittest Category1_Unittest.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
emmanuelburuvuru@Emmanuels-MacBook-Pro code %
```

Test Implimentation and execution

Production code file	Submodule.	Testing file.
category1.py	LowertoUpper	Category1_Unittest.py
category1.py	UppertoLower	Category1_Unittest.py
category1.py	NumericsinStrings	Category1_Unittest.py
category1.py	IsStringValid_number	Category1_Unittest.py
category1.py	remove_numeric	Category1_Unittest.py
category2.py	HrsToMinMinToSec	Category2_Unittest.py
category2.py	MinToHrsSecToMin	Category2_Unittest.py
category2.py	saveData	Category2_Unittest.py

Category1.py test inplementation.

The testing of the module Catergory1.py was done with Black box equivalence testing using the unit test frame to assert and check whether the values were used in the functions were putting out the right output.

Testing Function 1:

Testing lower to upper case was done by asserting the expected output to the function LowertoUpper.

```
class testCategory1(unittest.TestCase):

    def testLowertoUpper(self):

        string = "BURUVURU"
        string2 = "emmanuel buruvuru"
        string3 = "0194"
        string4 = "Doctor Strange"

        assert "BURUVURU" == Category1.LowertoUpper(string)
        assert "EMMANUEL BURUVURU" == Category1.LowertoUpper(string2)
        assert "0194" == Category1.LowertoUpper(string3)
        assert "DOCTOR STRANGE" == Category1.LowertoUpper(string4)
```

Testing Function 2:

Testing upper to lower case was done by asserting the expected output to the function and comparing the returned string.

```
def testUppertoLower(self):

    string1 = "BURUVURU"
    string2 = "emmanuel buruvuru"
    string3 = "0194"
    string4 = "Doctor Strange"

    assert "buruvuru" == Category1.UppertoLower(string1)
    assert "emmanuel buruvuru" == Category1.UppertoLower(string2)
    assert "0194" == Category1.UppertoLower(string3)
    assert "doctor strange" == Category1.UppertoLower(string4)
```

Testing Function 3:

The function of testing if there are numeric in a string was done with the assertion function since, we know the output of the function as it is returning a Boolean.

```
def testifstringcontainsnumerics(self):

    string1 = "BURUVURU"
    string2 = "emmanuel buruvuru"
    string3 = "0194"
    string4 = "Doctor Strange2"

    assert "False" == Category1.NumericsinStrings(string1)
    assert "False" == Category1.NumericsinStrings(string2)
    assert "True" == Category1.NumericsinStrings(string3)
    assert "True" == Category1.NumericsinStrings(string4)
```

Testing Function 4:

The function of validating if a string is a valid number or not was tested by taking input from the keyboard and the assessing if the inputs string is a valid number of not. This was done by capturing the input that the user puts and the using the function to validate the string.

```
def testisStringValid_number(self):

    #testing with a valid number
    captureOut = io.StringIO()
    sys.stdout = captureOut
    sys.stdin = io.StringIO("25")
    Category1.IsStringValid_number("25")
    self.assertEqual("String is a valid number..\n",captureOut.getvalue())
    sys.stdout = sys.__stdout__
    sys.stdin = sys.__stdin__

    #testing with a string
    captureOut = io.StringIO()
    sys.stdout = captureOut
    sys.stdin = io.StringIO("emmanuel buruvuru")
    Category1.IsStringValid_number("emmanuel buruvuru")
    self.assertEqual("String is not a valid number..\n",captureOut.getvalue())
    sys.stdout = sys.__stdout__
    sys.stdin = sys.__stdin__

    #testing with a valid number that contains a letter
    captureOut = io.StringIO()
    sys.stdout = captureOut
    sys.stdin = io.StringIO("10.e")
    Category1.IsStringValid_number("10.e")
    self.assertEqual("String is a valid number..\n",captureOut.getvalue())
    sys.stdout = sys.__stdout__
    sys.stdin = sys.__stdin__
```

Testing Function 5:

lastly in the category module one had to test the function that will check if a string contains a number and the remove the number returning the string in UPPERCASE. The function was tested by taking capturing the output of the returned string after passing a string into the function.

```
    def testRemovenumricsfromstring(self):

        #string without numerics
        captureOut = io.StringIO()
        sys.stdout = captureOut
        sys.stdin = io.StringIO("emmanuel buruvuru")
        Category1.remove_numeric("emmanuel buruvuru")
        self.assertEqual("final string: EMMANUEL BURUVURU\n",captureOut.getvalue())
        sys.stdout = sys.__stdout__
        sys.stdin = sys.__stdin__

        # testing with a string that has numeric values
        captureOut = io.StringIO()
        sys.stdout = captureOut
        sys.stdin = io.StringIO("Doctor strange2")
        Category1.remove_numeric("Doctor Strange2")
        self.assertEqual("final string: DOCTOR STRANGE\n",captureOut.getvalue())
        sys.stdout = sys.__stdout__
        sys.stdin = sys.__stdin__

        #testing with numeric values only

        captureOut = io.StringIO()
        sys.stdout = captureOut
        sys.stdin = io.StringIO("0194")
        Category1.remove_numeric("0194")
        self.assertEqual("final string: \n",captureOut.getvalue())
        sys.stdout = sys.__stdout__
        sys.stdin = sys.__stdin__
```

Results

Results generated by running the Unit test testing framework imported in the testing module as Testcase:

```
emmanuelburuvuru@Emmanuels-MacBook-Pro code % python3 -m unittest Category1_Unittest.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
emmanuelburuvuru@Emmanuels-MacBook-Pro code %
```

Category2.py test inplementation.

Testing of the module category2.py was done by reading the output from a file that the function save Data () writes to. The function save Data takes in input as a parameters that include filename (where the data is being saved), Hours , minutes and seconds and calls function one and two to perform the conversions.

Testing Function 1:

The function HrsToMinMinToSec() converts a given time unit by multiplying the unit by 60 and return a float as a conversion result. It takes in hours and minutes as time units and coverts them from hours to min and from minutes to seconds. This function is tested by outputting the conversion into a file and then asserting the file to the expected output.

```
class testCatergory2(unittest.TestCase):

    def testflileinput(self):

        #setup
        Hours = int(2)
        Minutes = int(60)
        Seconds = int(3600)
        filename = str('Correct_inputs.txt')
        Category2.saveData(filename, Hours, Minutes,Seconds)

        with open(filename) as outfile:

            #Check the string in the file
            self.assertEqual("Hours to minutes: 120\n" , outfile.readline())
            self.assertEqual("Minutes to Hours: 1.0\n" , outfile.readline())
            self.assertEqual("Seconds to minutes: 60.0\n" , outfile.readline())
            self.assertEqual("Minutes to seconds: 3600\n" , outfile.readline())
```

Testing Function 2:

The function MinToHrsSecToMin() converts a given time unit by dividing the unit by 60 and return a float as a conversion result. It takes in minutes and seconds as time units and coverts them from minutes to Hours and from seconds to minutes. This function is tested by outputting the conversion into a file and then asserting the file to the expected output.

```
    #testing the modules with a range of negative numbers

    #setup
    Hours = int(-2)
    Minutes = int(-60)
    Seconds = int(3600)
    filename = str('Negative_inputs.txt')
    Category2.saveData(filename, Hours, Minutes,Seconds)

    with open(filename) as outfile:

        #Check the string in the file
        self.assertEqual("Hours to minutes: input a positive int\n" , outfile.readline())
        self.assertEqual("Minutes to Hours: input a positive int\n" , outfile.readline())
        self.assertEqual("Seconds to minutes: 60.0\n" , outfile.readline())
        self.assertEqual("Minutes to seconds: input a positive int\n" , outfile.readline())
```

Testing Results:

Results generated by running the Unit test testing framework imported in the testing module as Testcase:

```
emmanuelburuvuru@Emmanuels-MacBook-Pro code % python3 -m unittest Category2_Unittest.py
.
----------------------------------------------------------------------
Ran 1 test in 0.003s

OK
emmanuelburuvuru@Emmanuels-MacBook-Pro code %
```

MODULE EXECUTION

Module name	BB test design(EP)	BB test design(BVA)	WB test design	EP test code (implemented/ run)	BVA test code(implemented/run)	White-Box testing (implemented/run)
Catergory1.py	Done	Not done	Done	Done	Not done	Done
Catergory2.py	Not Done	Done	Done	Not Done	Done	Done

Version Control

Git was used throughout the assignment in tracking all the changes of the files. The git repository was initialized and set to be modified by Emmanuel as the user. A log is shown below of the commits done through the assignment. The full git directory has been attached for more information of the local git repository.

```
emmanuelburuvuru@Emmanuels-MacBook-Pro documents % git log
commit 55692ad51f5fcbe11273669d520828bb843d65d7 (HEAD -> main)
Author: Emmanuel <Emmanuel.Burvuru@student.curtin.edu.au>
Date:   Mon May 30 21:38:29 2022 +0400

    Adding report screenshotsgit

commit ba933e11f84196b146bc0e643c7763546abee0e9 (origin/main)
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Mon May 30 20:14:52 2022 +0400

    Adding modified report

commit 79f4fc3e0cf8e25dd2e3fc24be587fd06863291c
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Mon May 30 20:13:04 2022 +0400

    Modified production code

commit e15f933c0cc19bc361be63007978c50e279fc4f5
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Mon May 30 20:09:28 2022 +0400

    Adding test cases

commit 998238dbeda98d915cc8d26ac266a4d74e68af89
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Mon May 30 10:31:46 2022 +0400

    Removing duplicate files
```

```
commit 157e1184612287e00f3939cba09250d23395af3a
Merge: 6030faa 07b6259
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Thu May 26 22:40:47 2022 +0400

    Merge branch 'main' of https://github.com/Sycolee78/Buruvuru_Emmanuel_20750194_ISERepo

    For the addition of the markdown extension to the report.

commit 6030faa1d5be19de377c69a1cda8c1ca30b6442d
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Thu May 26 22:39:34 2022 +0400

    Add a markdown extension to the report

commit 07b6259b9d94eef9198ee9aa53d8fc0e608227c0
Author: Sycolee78 <97883672+Sycolee78@users.noreply.github.com>
Date:   Thu May 26 16:15:58 2022 +0400

    Update README.md

commit 33d854311f94408514f7f7165a24b32abbd1731b
Author: Sycolee78 <97883672+Sycolee78@users.noreply.github.com>
Date:   Thu May 26 16:14:04 2022 +0400

    Update README.md

commit 85f1b17f060a2cdcacdd43ee0b8545341ab60a95
```

```
commit ef4b82b350074a27ded857328472dce278a5f7bc
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Thu May 26 15:16:46 2022 +0400

    Adding the document and code files

commit c9d802ea8eaf060eebe351de4974b27973592f0e
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Thu May 26 15:11:42 2022 +0400

    Changes for the catergories

commit 2ca4dcff482c55e968a559bc51553c9a972bf587
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Thu May 26 15:07:28 2022 +0400

    Changing the Report header

commit fad0db8e34ca5cd370b339dc10a93e216b61b93b
Author: Emmanuel Buruvuru <emmanuelburuvuru@Emmanuels-MacBook-Pro.local>
Date:   Thu May 26 14:54:10 2022 +0400

    first commit
```

Ethics

Ethical and professional misconduct

Scenario 1:

A software developer who is doing the time conversion should be aware of using the right module. If a user does not use the right module for the conversion, it would return a wrong value hence leading to disaster for instance, if the module was being used to convert a time from hours to minutes to get a precise time for an impact. The conversion might lead to the event happening sooner than expected and people won't be prepared for the impact in the worse scenario people might be

killed if it was used to countdown the impact of a meteor. The event might be delayed because the user might have used the time unit multiplier instead of dividing the time unit given, leading to losses as preparation of an impact might cost a lot of money for example impact of a meteor, putting all the measures in place to reduce the impact of the meteor might cost money and releasing the impact is delayed might result in re-evaluation.

Scenario 2:

A software developer might not think its different to implement conversion from upper case to lower case by using the UppercasetoLower() function in the catergory1.py module. If he is asked to run through a database and covert names if any that have upper case to lower case. He might omit it since he thinks it does not make any difference according to his knowledge. But the database can be run by an algorithm that does not register names that have an uppercase letter in the string hence, people might be omitted from the registration process. The software developer is doing not use the code ethically and did not protect people that were omitted by the system.

ACS Code of Professional Conduct

Honesty

The person who is using the code must be honest on what are the abilities of the program to the client. I should not intentionally mislead a customer or potential client about a product's or service's appropriateness, for instance the clients want a program that then converts the converted time unit to suit a schedule by using an API. I should tell the client that the program is not capable of doing such task.

The enhancement of life quality.

The code that was designed to contribute to a good life for others for instance if one is in a mathematical situation where it is requires to change time units. If he does not have a calculator or the internet available to him, he might use the code to make the desired conversion and get required results.

Discussion

In conclusion, the production code was designed successfully and met the requirements when provided the parameters to execute the functions (seen in the test cases). Modules could have been separated into different files to ensure proper testing of the modules separately.