|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ |
| УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ |
| «СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» |
|  |
| Институт информационных технологий и управления в технических системах |
| (полное название института) |
|  |
| кафедра «Информационные системы» |
| (полное название кафедры) |

**ОТЧЁТ**

по лабораторной работе №3

на тему«Исследование способов структурного тестирования программного обеспечения»

по дисциплине **«**Тестирование программного обеспечения»

Вариант 22

Выполнил

студент группы ПИ/б-18-1-о

Маринин Игорь

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | | | | | | | |
|  |  | (должность, учёная степень преподавателя) | | | | | | | | |
|  |  |  | | | | | | | | |
|  |  | (ФИО преподавателя) | | | | | | | | |
|  |  | « |  | » |  |  |  | 20 | 20 | г. |
|  |  |  | | | | | | | | |
|  |  | (оценка) | | | | | | | | |
|  |  |  | | | | | | | | |
|  |  |  | | | | | | | | |

Севастополь 2020

* 1. **Цель работы**

Исследовать основные подходы к модульному тестированию программного обеспечения. Приобрести практические навыки составления модульных тестов для объектно-ориентированных программ.

* 1. **Вариант задания**

На примере предыдущей лабораторной работы создать тест с возможностью записи результатов в текстовый файл.

В данной лабораторной работе тестируется класс Lab4. Он содержит три метода: PositiveD\_3x3(matr), принимающий матрицу 3 на 3, и возвращающий булево значение – является ли дельта матрицы положительным числом; ReplaceAllAfterDiez(s), принимающий строка, и возвращающий ту же строку, но заменив все символы, идущие после символа «#» на символ «@»; foundMinimalStrFromFile(fName), принимающий название файла, и возвращающий минимальную по количеству символов строку из этого файла.

* 1. **Ход работы**
     1. Определим области эквивалентности:

3.3.1.1 для метода PositiveD\_3x3:

1. По расположению элементов:

1. Все элементы одинаковые
2. Все элементы 0
3. Все элементы отрицательные
4. Все элементы положительные

3.3.1.2 для метода ReplaceAllAfterDiez(s):

1. По размеру строки

1. Строка из одного символа;
2. Строка, состоящая из более чем одного символа;
3. Пустая строка.

2. По расположению символов:

1. Все символы «#»;
2. Без символов «#»;
3. Символ «#» в начале строки;
4. Символ «#» в конце строки;
5. Символ «#» в середине сроки

3.3.1.3 для метода foundMinimalStrFromFile(fName):

1. По размеру файла:

1. Файл состоит из одной строки;
2. Файл состоит из более чем одной строки;
3. Файл пустой.

2. По расположению символов:

1. Все строки одного размера;
2. Строка с наименьшей строкой стоит в конце файла;
3. Строка с наименьшей длиной стоит в начале файла;
4. Строка с наименьшей длиной стоит в середине файла;
5. Все строки файла пустые.

3.3.2 Подготовим тестируемый класс

**class** **Lab4(object):**

**def** PositiveD\_3x3**(**matr**):**

**return** **(**matr**[**0**][**0**]** **\*** matr**[**1**][**1**]** **\*** matr**[**2**][**2**]**

**+** matr**[**1**][**0**]** **\*** matr**[**2**][**1**]** **\*** matr**[**0**][**2**]**

**+** matr**[**0**][**1**]** **\*** matr**[**1**][**2**]** **\*** matr**[**2**][**0**])** **-** **(**

matr**[**0**][**2**]** **\*** matr**[**1**][**1**]** **\*** matr**[**2**][**0**]**

**+** matr**[**0**][**0**]** **\*** matr**[**1**][**2**]** **\*** matr**[**2**][**1**]**

**+** matr**[**0**][**1**]** **\*** matr**[**1**][**0**]** **\*** matr**[**2**][**2**])** **>=** 0

**def** ReplaceAllAfterDiez**(**s**):**

**try:**

first\_in **=** s**.**index**(**'#'**)** **+** 1

length **=** **len(**s**)**

**if** first\_in **>** length**:**

**return** s

**else:**

**return** s**[**0**:**first\_in**]** **+** '@' **\*** **(**length **-** first\_in**)**

**except** **ValueError:**

**return** s

3.3.2 Подготовим класс с тестами

**import** os

**from** main **import** Lab4

**import** unittest

**class** **testPositiveD\_3x3(**unittest**.**TestCase**):**

**def** allElementsAreSame**(**self**):**

matr **=** **[[**1**,** 1**,** 1**],**

**[**1**,** 1**,** 1**],**

**[**1**,** 1**,** 1**]]**

expected **=** **True**

actual **=** Lab4**.**PositiveD\_3x3**(**matr**)**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** allElementsAreZeroes**(**self**):**

matr **=** **[[**0**,** 0**,** 0**],**

**[**0**,** 0**,** 0**],**

**[**0**,** 0**,** 0**]]**

expected **=** **True**

actual **=** Lab4**.**PositiveD\_3x3**(**matr**)**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testPositive**(**self**):**

matr **=** **[[**5**,** 2**,** 3**],**

**[**2**,** 6**,** 1**],**

**[**3**,** 1**,** 7**]]**

expected **=** **True**

actual **=** Lab4**.**PositiveD\_3x3**(**matr**)**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testNegative**(**self**):**

matr **=** **[[-**11**,** **-**6**,** **-**6**],**

**[-**6**,** **-**6**,** **-**3**],**

**[-**6**,** 3**,** **-**6**]]**

expected **=** **False**

actual **=** Lab4**.**PositiveD\_3x3**(**matr**)**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**class** **testStringReplacer(**unittest**.**TestCase**):**

**def** testOneSymbolString**(**self**):**

s **=** 'A'

expected **=** 'A'

actual **=** Lab4**.**ReplaceAllAfterDiez**(**s**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testEmptyString**(**self**):**

s **=** ''

expected **=** ''

actual **=** Lab4**.**ReplaceAllAfterDiez**(**s**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testAllDiezString**(**self**):**

s **=** '##########'

expected **=** '#@@@@@@@@@'

actual **=** Lab4**.**ReplaceAllAfterDiez**(**s**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testStringWithoutDiez**(**self**):**

s **=** 'Hello, world!'

expected **=** 'Hello, world!'

actual **=** Lab4**.**ReplaceAllAfterDiez**(**s**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testStringWithDiez**(**self**):**

s **=** 'Hello,#world!'

expected **=** 'Hello,#@@@@@@'

actual **=** Lab4**.**ReplaceAllAfterDiez**(**s**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testStringWithManyDiez**(**self**):**

s **=** 'Hel#o,#world!'

expected **=** 'Hel#@@@@@@@@@'

actual **=** Lab4**.**ReplaceAllAfterDiez**(**s**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testStringFirstDiez**(**self**):**

s **=** '#ello, world!'

expected **=** '#@@@@@@@@@@@@'

actual **=** Lab4**.**ReplaceAllAfterDiez**(**s**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testStringLastDiez**(**self**):**

s **=** 'Hello, world#'

expected **=** 'Hello, world#'

actual **=** Lab4**.**ReplaceAllAfterDiez**(**s**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**class** **testFoundMinimalStrInFile(**unittest**.**TestCase**):**

**def** testOneString**(**self**):**

fName **=** 'testEqual.txt'

strDat **=** **[**'12345'**]**

**with** **open(**fName**,** 'w'**)** **as** f**:**

**for** s **in** strDat**:**

f**.**write**(**s **+** '\n'**)**

expected **=** strDat**[**0**]**

actual **=** Lab4**.**FoundMinimalStrFromFile**(**fName**)**

os**.**remove**(**fName**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testEmptyFile**(**self**):**

fName **=** 'testEmpty.txt'

**with** **open(**fName**,** 'w'**)** **as** f**:**

f**.**write**(**'\n'**)**

expected **=** ''

actual **=** Lab4**.**FoundMinimalStrFromFile**(**fName**)**

os**.**remove**(**fName**)**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testAllEquivalent**(**self**):**

fName **=** 'testEqual.txt'

strDat **=** **[**'12345'**,** '67890'**,**

'52132'**,** '51231'**]**

**with** **open(**fName**,** 'w'**)** **as** f**:**

**for** s **in** strDat**:**

f**.**write**(**s **+** '\n'**)**

expected **=** strDat**[**0**]**

actual **=** Lab4**.**FoundMinimalStrFromFile**(**fName**)**

os**.**remove**(**fName**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testFirstMin**(**self**):**

fName **=** 'testNormal.txt'

strDat **=** **[**'12345'**,** '6722152890'**,**

'521'**,** '5123121'**]**

**with** **open(**fName**,** 'w'**)** **as** f**:**

**for** s **in** strDat**:**

f**.**write**(**s **+** '\n'**)**

expected **=** strDat**[**2**]**

actual **=** Lab4**.**FoundMinimalStrFromFile**(**fName**)**

os**.**remove**(**fName**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testLastMin**(**self**):**

fName **=** 'testNormal.txt'

strDat **=** **[**'12345'**,** '6722152890'**,**

'521'**,** '5123121'**]**

**with** **open(**fName**,** 'w'**)** **as** f**:**

**for** s **in** strDat**:**

f**.**write**(**s **+** '\n'**)**

expected **=** strDat**[**2**]**

actual **=** Lab4**.**FoundMinimalStrFromFile**(**fName**)**

os**.**remove**(**fName**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testNormal**(**self**):**

fName **=** 'testNormal.txt'

strDat **=** **[**'12345'**,** '6722152890'**,**

'521'**,** '5123121'**]**

**with** **open(**fName**,** 'w'**)** **as** f**:**

**for** s **in** strDat**:**

f**.**write**(**s **+** '\n'**)**

expected **=** strDat**[**2**]**

actual **=** Lab4**.**FoundMinimalStrFromFile**(**fName**)**

os**.**remove**(**fName**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**def** testAllEmpty**(**self**):**

fName **=** 'testNormal.txt'

strDat **=** **[**''**,** ''**,**

''**,** ''**]**

**with** **open(**fName**,** 'w'**)** **as** f**:**

**for** s **in** strDat**:**

f**.**write**(**s **+** '\n'**)**

expected **=** strDat**[**0**]**

actual **=** Lab4**.**FoundMinimalStrFromFile**(**fName**)**

os**.**remove**(**fName**);**

self**.**assertEqual**(**expected**,** actual**,**

f'Expected: {expected}\nGot: {actual}'**)**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

logFile **=** 'log\_file.txt'

**with** **open(**logFile**,** 'w'**)** **as** f**:**

runner **=** unittest**.**TextTestRunner**(**f**)**

unittest**.**main**(**testRunner **=** runner**,** verbosity**=**2**)**

3.3.3 Запустим тестирование и получим результаты в текстовом файла «log\_file.txt» (рисунок 3.1-3.2).

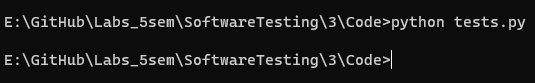


Рисунок 3.1 – Запуск тестирования

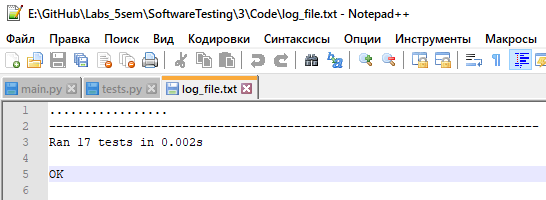


Рисунок 3.2 – Результаты тестирования

**Вывод**

В ходе выполнения лабораторной работы были исследованы основные подходы к модульному тестированию программного обеспечения. Была написана программа, тестирующая класс Lab4, а так же проверено выполнение методов данного класса, результаты проверки записаны в файл. Методы были протестированы с учётом анализа областей эквивалентности входных данных, а именно размер матрицы/строки/файла и расположение элементов/символов/строк.