**18-КБ-ПР2 Буниф М.**

**Отчёт по лабораторной работе №2**

1. **Наименование и цель работы**

**Наименование:** Тестирование методом черного ящика

**Цель работы:** Изучить подход к тестированию методом черного ящика.

1. **Задание на лабораторную работу**

1) Создать класс (в соответствии с вариантом задания из п.5), реализующий проверку принадлежности точки различным областям плоскости, задаваемых пересечением фигур.

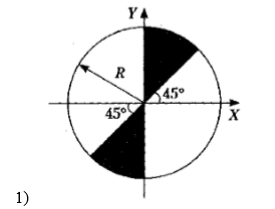
2) Выделить классы эквивалентности.

3) Проверить граничные значения для каждого из классов эквивалентности.

4) Составить наборы тестовых данных для созданного метода.

5) Протестировать метод на основе тестового набора с использованием программных отладочных средств.

6) Составить отчет о результатах проведенного тестирования.



**3. Схема алгоритма, текст программы на алгоритмическом языке.**

**Проект 1, Ronde.py**

import math  
import types  
  
  
# def complex(radius,x, y):  
# return isinstance(x, types.Tuple) and isinstance(x, types.Tuple)and isinstance(x, types.Tuple)  
# return complex()  
  
def check(radius, x, y):  
 angle = math.atan2(x, y) \* (180 / math.pi)  
 if 0 <= x <= radius and 0 <= y <= radius and x \*\* 2 + y \*\* 2 <= radius \*\* 2 and angle == 45 \  
 or (-1) \* radius <= y <= 0 and (-1) \* radius <= x <= 0 and x \* x + y \* y <= radius \* radius and angle == 45:  
 return True  
 else:  
 return False  
 return check()  
 # or -radius <= y <= 0 and -radius <= x <= 0 and x\*x + y\*y <= radius\*radius and math.atan2(x, y) == -(math.pi /

**Проект 2, test\_Ronde.py**

from unittest import TestCase  
from Ronde import check  
  
  
  
class TestPointArea1(TestCase):  
 def test\_check\_Coordinates(self): # the coordinates are inside the circle exactly in the black area  
 # quarter I  
 radius = 4  
 x = 1  
 y = 1  
 expected = True  
 actual = check(4, 1, 1)  
 self.assertEqual(expected, actual)  
  
  
class TestPointArea2(TestCase):  
 def test\_check(self): # the coordinates are outside the circle  
 radius = 4  
 x = 7  
 y = 1  
 expected = False  
 actual = check(4, 7, 1)  
 self.assertEqual(expected, actual)  
  
  
class TestPointArea3(TestCase):  
 def test\_check(self):  
 radius = 0  
 x = 0  
 y = 0  
 expected = False  
 actual = check(0, 0, 0)  
 self.assertEqual(expected, actual)  
  
  
class TestPointArea4(TestCase): # the coordinates are in 3rd quarter the white area  
 def test\_check(self):  
 radius = -8  
 x = -3  
 y = 4  
 expected = False  
 actual = check(-8, -3, 4)  
 self.assertEqual(expected, actual)

**Результат работы программы**

