Министерство образования и науки Украины

Днепропетровский национальный университет железнодорожного

транспорта имени академика В.Лазаряна

**Лабораторная работа №1**

по дисциплине: «Качество программного обеспечения и тестирование»

Выполнил: ст. группы ПЗ1611

Барановский Максим

Принял: Шинкаренко В.И.

г. Днепр

2019 год

**Тема:** тестирование методом белого ящика.

**Цель**: получить практические навыки в тестировании белым ящиком.

**Тестирование** - процесс выполнения программы (или ее части) с целью проверки ее соответствия установленным требованиям, указанным в спецификации, и выявления дефектов (ошибок). Под **спецификацией** будем понимать точное, недвусмысленное описание действий (операций), которые должна выполнять программа (или ее часть), ее входные и выходные данные.

**Ошибка** - данные и/или состояние программы, которое является результатом работы программы и не соответствует ее спецификации.

**Тест** - набор входных и выходных данных для разового выполнения программы, удовлетворяющий спецификации.

**Хороший тест** - тест, который имеет высокую вероятность обнаружения ошибок.

Тестирование состоит в динамической проверке поведения программы на конечном множестве тестовых данных, выбранных специальным образом из бесконечного входного пространства, на соответствие установленному ожидаемому поведению.

Тестирование всегда предполагает выполнение программы. Неполнота - одна из основных проблем тестирования, поскольку на практике полное множество тестов можно рассматривать как бесконечное. Количество же тестов, которые могут быть выполнены в ограниченные сроки, - конечное.

Выделяют четыре уровня тестирования:

* модульное - проверка функциональных объектов отдельно друг от друга;
* интеграционное - проверка правильности взаимодействия между программными объектами, которые были протестированы ранее автономно;
* тестирование ПО - проверка ПО в моделируемой среде;
* системное - проверка взаимодействия ПО с другими, в том числе не программными, компонентами системы.

Выделяют следующие виды тестирования характеристик программной системы:

* функциональное тестирование;
* тестирование безопасности;
* тестирование удобства использования;
* тестирование технических характеристик;
* тестирование на надежность;
* тестирование производительности;
* тестирование конфигурации;
* сравнительное тестирование;
* тестирование восстановления.

Методы тестирования различаются подходами к проектированию тестов. Традиционно методы функционального тестирования разделяют на две категории - черным ящикок (без доступа к тексту программы) и белым ящиком (с доступом).

К **методам белого ящика** относятся:

1. покрытие операторов: предусматривает подбор таких наборов входных данных, при которых каждый оператор программы исполнится хотя бы один раз;
2. покрытие решений: под решением будем понимать набор условий и их значений, определяющих ход выполнения программы (следует включить условия всех управляющих структур: циклов всех видов (for, while, do-while) и разветвлений (if, switch). При покрытии решений каждое направление должно быть реализовано хотя бы один раз. Необходимо также учитывать покрытия операторов.;
3. покрытие условий: предусматривает создание такого набора тестов, по которым каждое условие выполнится и не выполнится хотя бы один раз;
4. покрытие условий и решений: возможны результаты каждого условия в решении получены по крайней мере один раз и все решения исполнились хотя бы один раз;
5. комбинаторное покрытие условий: необходимо протестировать все возможные комбинации условий.

**Текст программы:**

1. Спецификация: функция кодировки матрицы алгоритмом Z для получения матрицы, представляющей прозрачное изображение, в котором степень прозрачности определяется целочисленным числом от 0 до 10, где 0 полностью прозрачный пиксель, а 10 - закрашенный. Алгоритм Z к работает следующим образом: каждый элемент матрицы m(i,j) увеличивается на число равное кол-ву положительных элементов стоящих по правую сторону в строке матрицы от элемента m(i,j), далее если m(i,j) отрицательное число или больше 20 - m(i,j) приравнивается к 0, иначе в случае если m(i,j) больше 10 - приравнивается к 10

Вход: двумерный целочисленный массив, размеры массива

Выход: двумерный целочисленный массив с измененными элементами

**static int**[][] EncryptMatrix(**int**[][] m)

**{**

**if** (m == **null**)

{

**return** m;

}

**var** matrix = m.Clone() **as int**[][];

**for** (**int** i = 0; i < matrix.Length; i++)

{

**for** (**int** j = 0; j < matrix[i].Length; i++)

{

**int** k = j;

**while** (k < matrix[i].Length)

{

**if** (matrix[i][k] > 0)

{

matrix[i][j]++;

}

k++;

}

**if** (matrix[i][j] < 0 || matrix[i][j] > 20)

{

matrix[i][j] = 0;

}

**else if** (matrix[i][j] > 10)

{

matrix[i][j] = 10;

}

}

}

**return** matrix;

**}**

Тестирование данной функции:

Тест1: вход: matrix = {{10,20,1}{0,0,0}{-1,20,-20}}; выход: {{10,0,2}{0,0,0}{0,0,0}};

Тест2: вход: matrix = null; выход: null;

Результаты тестирования методом покрытия условий приведены в табл.1

Таблица 1

**Таблица покрытия условий**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Условие  Тест | k < matrix[i].Length | | matrix[i][k] > 0 | | matrix[i][j] < 0 | | matrix[i][j] > 20 | | matrix[i][j] > 10 | | m ==null | |
| + | - | + | - | + | - | + | - | + | - | + | - |
| 1 | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* |  | \* |
| 2 |  |  |  |  |  |  |  |  |  |  | \* |  |

2. Спецификация: функция изменения строковой матрицы таким образом, чтобы каждая ее ячейка содержала элемент длины n, при этом лишнее от каждого элемента будет перемещено на соседний(следующий) элемент, а в случае если у длина содержимого ячейки меньше чем n - содержимое будет дополнено символом ‘\*’, лишнее от последней ячейки отбросится .

Вход: двумерный строковый массив, размеры массива, длина содержимого ячейки

Выход: двумерный строковый массив с изменеными ячейками

**static string**[][] ClearMatrix(**string**[][] m, **int** cellLength)

**{**

**if** (cellLength <= 0 || m == **null**)

{

**return** m;

}

**var** matrix = m.Clone() **as string**[][];

**for** (**int** i = 0; i < matrix.Length; i++)

{

**for** (**int** j = 0; j < matrix[i].Length; i++)

{

**var** line = matrix[i][j];

**if** (line.Length > cellLength)

{

**var** first = line.Substring(0, cellLength);

**var** second = line.Substring(cellLength);

matrix[i][j] = first;

**if** (j + 1 < matrix[i].Length)

{

matrix[i][j + 1] = second + matrix[i][j + 1];

}

**else if** (i + 1 < matrix.Length)

{

matrix[i + 1][0] = second + matrix[i + 1][0];

}

}

**else**

{

**int** count = cellLength - line.Length;

**while** (count > 0)

{

matrix[i][j] += "\*";

count --;

}

}

}

}

**return** matrix;

**}**

Тестирование функции:

Тест 1: вход: m = null, cellLength = 0; выход: null.

Тест 2: вход: m = {{“12345”,”123”,”123456”},{“12345”,”12345”,”12345”}}, cellLength = 5; выход: {{“12345”,”123\*\*\*”,”12345”},{“61234”,”51234”,”51234”}}.

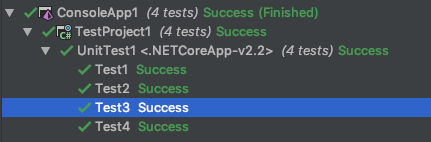
Тест 3: вход: cardNumber = {4,9,4,8,5,2,5,2,4,9,4,9}, cardNumberLength = 11; выход: Сообщение «Card is invalid", false.

Результаты тестирования методом покрытия условий приведены в табл.2.

Таблица 2

**Таблица покрытия условий**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Условие  Тест | line.Length > cellLength | | j + 1 < matrix[i].Length | | i + 1 < matrix.Length | | count > 0 | | cellLength <= 0 | | m == **null** | |
| + | - | + | - | + | - | + | - | + | - | + | - |
| 1 |  |  |  |  |  |  |  |  | \* |  | \* |  |
| 2 | \* | \* | \* | \* | \* | \* | \* | \* |  | \* |  | \* |

**Анализ результатов и выводы:** в ходе выполнения лабораторной работы были закреплены знания по тестированию методами белого ящика. Подготовлено две функции, выполнено тестирование методами белого ящика. Были разработаны и успешно пройдены юнит тесты. 

Контрольні питання

1. Основні поняття тестування (тест, тестування, хороший тест, помилка).

2. Методи тестування білою скринькою.

3. Метод покриття операторів.

4. Метод покриття умов.

5. Метод покриття рішень.

6. Метод покриття умов і рішень.

7. Метод комбінаторного покриття умов.

Контрольні питання складності

1. Переваги та недоліки тестування білою скринькою.

2. Оптимальна кількість тестів.

3. Види тестування характеристик програмної системи.

4. Рівні тестування програм.

5. Стратегія тестування.

6. Тестування багатомодульних програм.