Практическое занятие №8

Тема: Тестирование методом черного и белого ящика

Тестирование черного ящика

Основные аспекты:

1. Не знаем/Игнорируем устройство тестируемого объекта
2. Можем управлять входными параметрами
3. Среда, в которой проводим эксперименты, может считаться входным параметром
4. Можем измерять выходные параметры

Области для поиска ошибок:

1. Неправильные или пропущенные функции
2. Ошибки интерфейсов
3. Инициализация и завершение
4. Производительность
5. Структура данных

Область применения:

1. Unit-тестирование
2. Интеграционное тестирование
3. Системное тестирование
4. Приемочное тестирование

Шаги тестирования:

1. Изучение спецификаций и требований
2. Выбор входных значений
3. Определение ожидаемых выходных значений

Входные значения	Ожидаемые выходные значения
Вход 1	Выход 1
Вход 2	Выход 2
…	…
Вход N	Выход N

4. Исполнение тестов
5. Сравнение полученных результатов и ожидаемых

Стратегии тестирования:

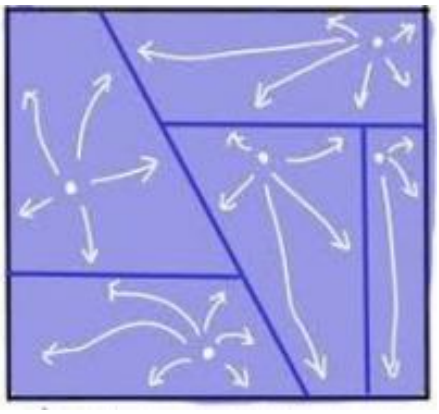
Число тестов определяется числом входов и диапазоном возможных значений входных параметров.

Перебор всех возможных входных параметров, как правило, невозможен.

Пример: сложение двух 4х-байтовых целых – 2^{64} входных параметра.

Стратегии уменьшения числа тестов:

Классы эквивалентности



Граничные значения

Преимущества:

1. Тестирование с точки зрения пользователя

2. Не требует специальных знаний (например, конкретного языка программирования)

3. Позволяет найти проблемы в спецификациях

4. Можно создавать тесты параллельно с кодом

5. Тестировщик может быть отделен от разработчиков

Недостатки:

1. Эффективность зависит от выбора конкретных тестовых значений

2. Необходимость наличия четких и полных спецификаций

3. Невозможность сконцентрироваться на особо сложных частях кода

4. Трудность локализации причины дефекта 5. Возможность не протестировать часть кода

Тестирование белого ящика

Принципы тестирования:

Используется знание об устройстве тестируемого объекта.

В случае ПО – имеется полный доступ к тестируемому коду.

Область применения:

1. Unit-тестирование

2. Интеграционное тестирование

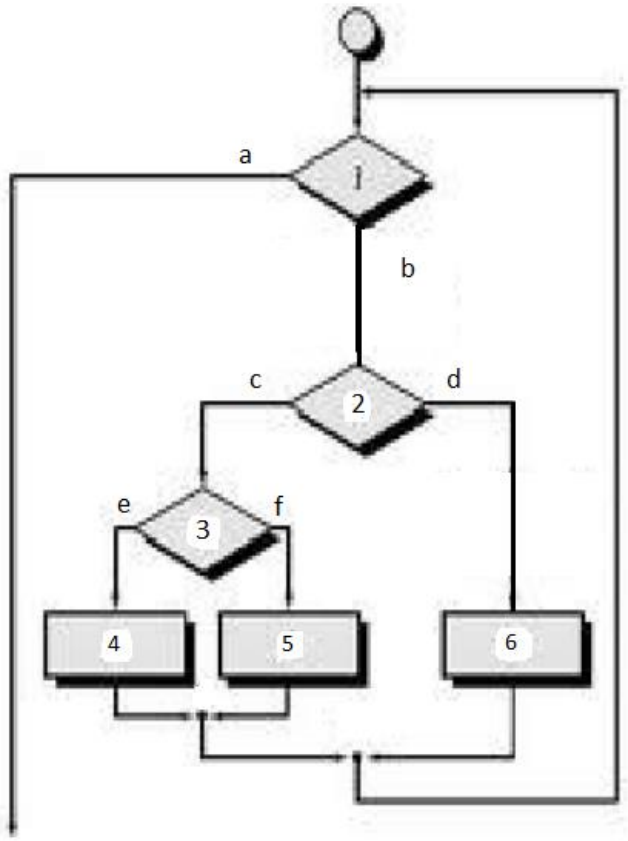
Шаги:

1. Представляем программу в виде графа

При таком представлении операторы программы необходимо обозначать арабскими цифрами, а соответствующие ветви программы латинскими буквами.

```
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Random rand = new Random();
            int[] value = new int[100];
            int minimum=-10, maximun=10;
            int n = rand.Next(20);
            //Цикл ввода массива
            for (int x = 0; x < value.Length; x++)
            {
                value[x] = rand.Next(100) - rand.Next(50);
            }
            int i = 0, sum = 0;
            while (i<n) {
                if (value[i] >= minimum)
                {
                    if (value[i] <= maximun)
                    {
                        sum += value[i];
                        i++;
                    }
                    else
                    {
                        i++;
                    }
                }
                else
                {
                    i++;
                }
            }
            Console.Write(sum);
        }
    }
}
```

1¶ 2¶ 3¶ 4¶ 5¶ 6¶



2. Создаем тестовые сценарии чтобы:

– Попасть в каждое ветвление

– Пройти хоть раз все вершины

– Пройти всеми возможными путями

– Пройти через вновь добавленные участки

– Пройти через известные проблемные участки

Покрытие программного кода

Покрытие программного кода (code coverage) – мера измерение оттестированнности имеющегося программного кода.

Виды покрытия кода

Функциональное (**Function coverage**) – каждая функция вызывается хотя бы раз

Строковое **(Statement coverage) –** каждая строка кода выполнялась хотя бы раз

Решения (**Decision/Branch coverage**) – в каждом условном операторе прошли по всем веткам выбора

Условия (**Condition coverage**) – каждое атомарное булево выражение приняло значения и «истина» и «ложь»

Параметров (**Parameter Value coverage**) – если метод имеет параметры, все значения параметра были использованы

Пути (**Path Coverage**) – все возможные пути в коде были пройдены

Циклы (**Loop coverage**) – Все циклы исполнялись 0,1,..N раз

Преимущества:

1. Позволяет найти «скрытые» в коде дефекты
2. Позитивные побочные эффекты (например, обучение команды)
3. Нахождение проблем производительности
4. Более надежное разбиение на классы эквивалентности
5. Как правило, ускорение цикла нахождение-исправление

Недостатки:

1. Не найдем пропущенное в коде
2. Дорого

Отличия черного и белого ящиков

Критерий	Черный ящик	Белый ящик
Основной уровень применимости	Приемочное тестирование	Юнит-тестирование
Ответственный	Независимый тестировщик	Разработчик
Знание программирования	Не обязательно	Необходимо
Знание реализации	Не обязательно	Необходимо
Знание сценариев использования	Необходимо	Необязательно
Основа тестовых сценариев	Спецификации	Код программы

Исходная задача:

Необходимо определить, можно ли из трёх отрезков составить треугольник. В случае утвердительного ответа определить его тип: остроугольный, прямоугольный или тупоугольный.

Вход: Три числа a, b, c – длины трех отрезков.

Выход: Строка, содержащая информацию о треугольнике: «ACUTE», если он остроугольный, «RIGHT», если прямоугольный и «OBTUSE» если тупоугольный. Если из трех отрезков составить треугольник нельзя, то вывести «NONE».

Задание

Напишите набор тестовых сценариев для указанной программы, используя методы черного и белого ящика.

Тестовые сценарии для черного ящика запишите в виде таблицы разделив все тесты по классам эквивалентности;

Проверяемый критерий	Входные данные			Ожидаемый результат	Результат программы
	a	b	c		
Первый класс эквивалентности					
Первый критерий					
Второй критерий					
Второй класс эквивалентности					
Третий критерий					
Четвертый критерий					
…					

Для белого ящика необходимо обеспечить 100% покрытия всех путей (Path coverage) отдельно для методов белого и черного ящика. a b c Ожидаемое выходное значение Заданы длины трех отрезков a, b, c.

Проверяемый путь	Входные данные			Ожидаемый результат	Результат программы
	a	b	c		
Первый путь (указание вершин графа в порядке обхода)					
Второй путь (указание вершин графа в порядке обхода)					
…					

Программа

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
        int a;
        int b;
        int c;
        string res;
        Console.Write("Введите a: ");
        a = int.Parse(Console.ReadLine());
        Console.Write("Введите b: ");
        b = int.Parse(Console.ReadLine());
        Console.Write("Введите c: ");
        c = int.Parse(Console.ReadLine());
        int aa = a * a;
        int bb = b * b;
        int cc = c * c;
        if (a >= b + c || b >= a + c || c >= a + b)
        {
            res = "NONE";
        }
        else
        if (aa == bb + cc || bb == aa + cc || cc == aa + bb)
        {
            res = "RIGHT";
        }
        else
        if (aa < bb + cc && bb < aa + cc && cc < aa + bb)
        {
            res = "ACUTE";
        }
        else
        {
            res = "OBTUSE";
        }
        Console.WriteLine(res);
        }
    }
}
```

```
using System;

namespace ConsoleApp1
{
    Ссылок: 0
    class Program
    {
        Ссылок: 0
        static void Main(string[] args)
        {
            int a;
            int b;
            int c;
            string res;
            Console.Write("Введите a: ");
            a = int.Parse(Console.ReadLine());
            Console.Write("Введите b: ");
            b = int.Parse(Console.ReadLine());
            Console.Write("Введите c: ");
            c = int.Parse(Console.ReadLine());
            int aa = a * a;
            int bb = b * b;
            int cc = c * c;

            if (a >= b + c || b >= a + c || c >= a + b)
            {
                res = "NONE";
            }
            else
            if (aa == bb + cc || bb == aa + cc || cc == aa + bb)
            {
                res = "RIGHT";
            }
            else
            if (aa < bb + cc && bb < aa + cc && cc < aa + bb)
            {
                res = "ACUTE";
            }
            else
            {
                res = "OBTUSE";

            }
            Console.WriteLine(res);
        }
    }
}
```