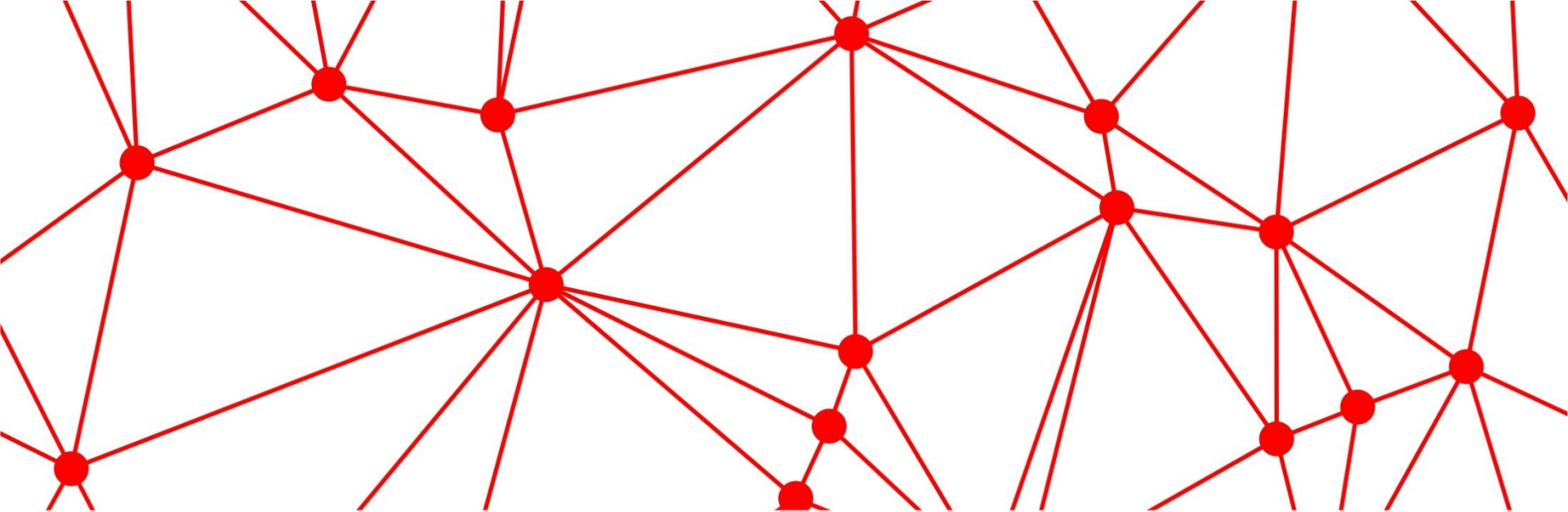


Методы тестирования

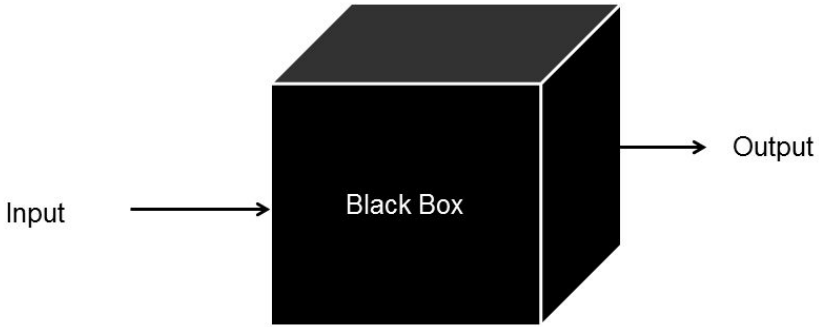
2015

Agenda

1. Техника анализа классов эквивалентности
2. Техника анализа граничных значений
3. Тестирование состояний переходов

Testing Methods

- Black Box
  - Equivalence Class Testing
  - Boundary Value Testing
  - State Transition Testing



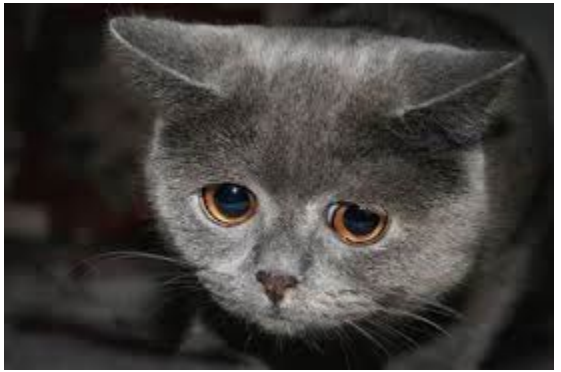
Internal behavior of the code is unknown

- White Box
  - Code Coverage

Equivalence classes Example

- Программа должна принимать два числа от -99 до 99 и складывать их

- Всего вариантов 199*199 = 39 601



Equivalence classes

Почему?

1. Потому что могут использоваться на разных уровнях – от отдельных функций до целого продукта
2. Потому что многие тестировщики пользуются ими интуитивно каждый день
3. Потому что неправильное использование этих техник может привести к пропуску серьезных ошибок

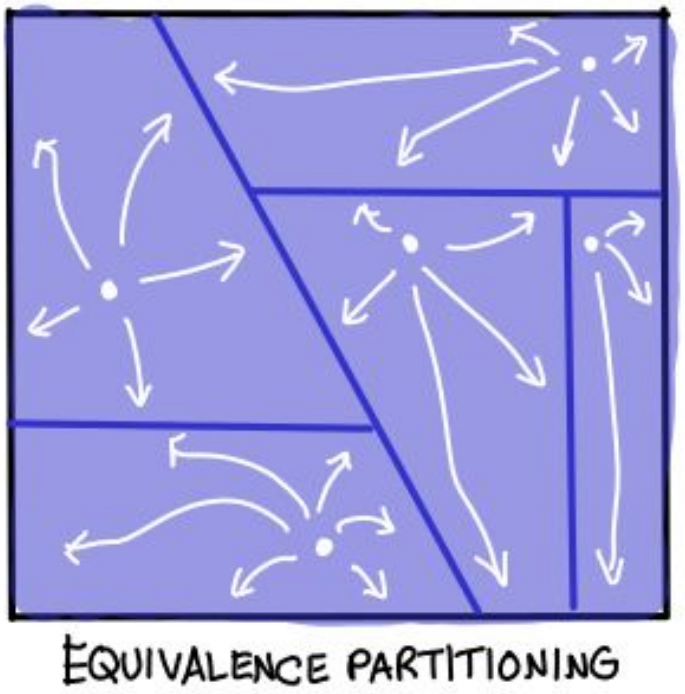
Equivalence classes

- Это техника, которая заключается в разбиении всего набора тестов на классы эквивалентности с последующим сокращением числа тестов

Equivalence classes

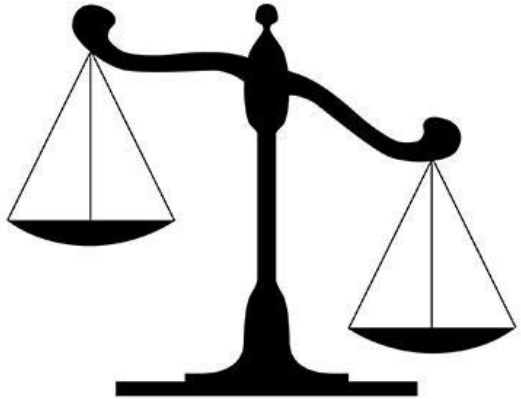
Цель:

- Сокращение числа тестов с сохранением тестового покрытия



Equivalence classes

- Слишком большое количество эквивалентных классов увеличивает вероятность того, что множество тестов будет избыточным
- Слишком малое число эквивалентных классов увеличивает вероятность пропуска ошибок



Equivalence classes Definition

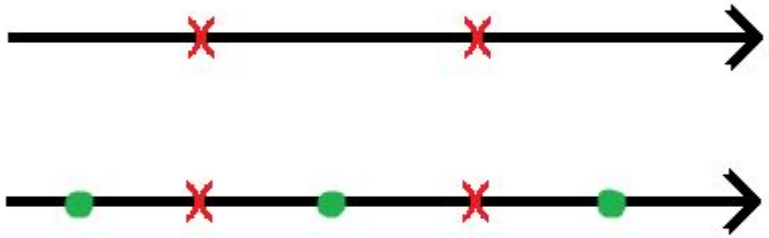
Два теста считаются эквивалентными, если:

- Они тестируют одну и ту же вещь (функцию, модуль, часть системы)
- Если один из тестов ловит ошибку, то второй скорее всего тоже ее поймает
- Если один из тестов не ловит ошибку, то другой тоже скорее всего ее не поймает

$$\begin{cases} \{O_1, O_2\} \\ \{O_3, O_5, O_7, O_9, O_{10}\} \\ \{O_4, O_6, O_8\} \end{cases}$$

Equivalence classes Algorithm

1. Определить классы эквивалентности
2. Выбрать одного представителя из каждого класса
3. Выполнить тесты



Equivalence classes Example

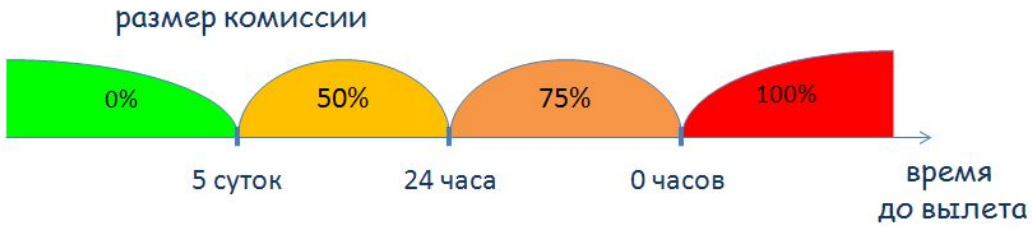
Комиссия при отмене бронирования авиабилета:

- За 5 суток – 0%
- Меньше 5 суток, но больше 24 часов – 50%
- Меньше 24 часов, но до вылета – 75%
- После вылета – 100%

Equivalence classes Example

1. Определим классы эквивалентности:

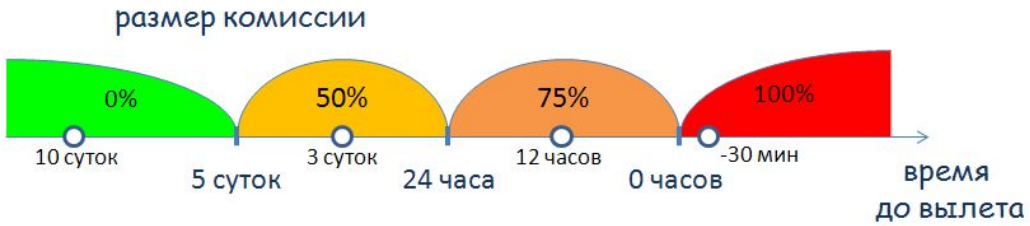
1) t > 5 суток
2) 24 часа < t < 5 суток
3) 0 часов < t < 24 часа
4) t < 0 (вылет состоялся)



Equivalence classes Example

2. Выберем представителя от каждого класса:

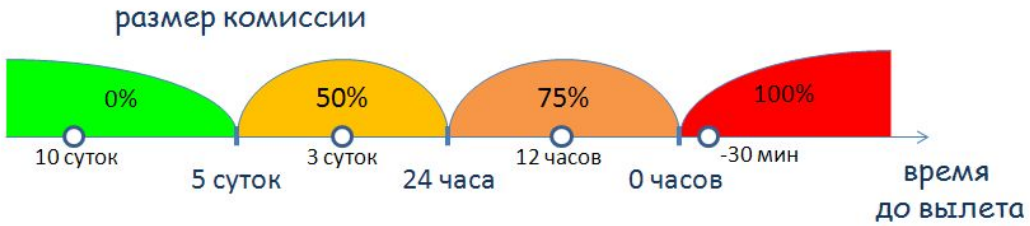
1) t = 10 суток
2) t = 3 суток
3) t = 12 часов
4) t = -30 минут



Equivalence classes Example

3. Выполним тесты:

1) t = 10 суток – комиссия 0%
2) t = 3 суток – комиссия 50 %
3) t = 12 часов – комиссия 75%
4) t = -30 минут – комиссия 100%



4 vs 950400 тестов

Equivalence classes

Плюсы:

- Сокращение времени
- Улучшение структурированности

Минусы:

- При неправильном использовании рискуем потерять баги

Equivalence classes Search

- Не забывайте о классах, охватывающих заведомо неверные или недопустимые входные данные
- Организуйте формируемый перечень классов в виде таблицы или плана
- Определите диапазоны значений
- Для полей или параметров, принимающих фиксированные перечни значений, выясните, какие из значений входят в перечень
- Проанализируйте возможные результаты выбора из списков или меню

Equivalence classes Search

- Поищите переменные, значения которых должны быть равны
- Поищите классы значений, зависящих от времени
- Выявите группы переменных, совместно участвующих в определенных вычислениях, результат которых ограничивается конкретным набором или диапазоном значений
- Посмотрите на какие действия программа отвечает эквивалентными событиями
- Продумайте варианты операционного окружения

Equivalence classes Table

Событие	Допустимые классы	Недопустимые классы
Ввод числа	От -99 до 99	• <-99 • >99 • Выражение, результат которого – недопустимое число (5-5=0) • Отрицательные числа • Буквы и другие нечисловые символы
Ввод первой буквы имени	•Первый символ является заглавной буквой •Первый символ является прописной буквой	Первый символ не является буквой
Рисование прямой	От одной точки до 4 см	•Отсутствие рисунка •Длиннее четырех сантиметров •Линия не является прямой

Equivalence classes Plan

1. Ввод числа
1.1 Допустимые варианты
1.1.1 Числа от 1 до 99
1.2 Недопустимые варианты
1.2.1 0
1.2.2 > 99
1.2.3 Выражение, результатом которого является недопустимое число, например, **5 - 5**, результат которого равен **0**.
1.2.4 Отрицательные числа
1.2.5 Буквы и другие нечисловые символы
1.2.5.1 Буквы
1.2.5.2 Арифметические операции, такие как +, * , -
1.2.5.3 Остальные нецифровые символы
1.2.5.3.1 Символы с ASCII-кодами, меньшими кода нуля
1.2.5.3.2 Символы с ASCII-кодами, большими кода девятки

Equivalence classes Plan

2. Ввод первой буквы имени
2.1 Допустимые варианты
2.1.1 Первый символ является заглавной буквой
2.1.2 Первый символ является прописной буквой
2.2 Недопустимые варианты
2.2.1 Первый символ не является буквой
2.2.1.1 Первый символ имеет ASCII-код, меньший кода буквы "A"
2.2.1.2 Первый символ имеет ASCII-код, лежащий между кодами букв "Я" и "а"
2.2.1.3 Первый символ имеет ASCII-код, больший кода буквы "я"

Equivalence classes Plan

3. Рисование прямой

3.1 Допустимые варианты

3.1.1 От одной точки до четырех сантиметров длиной

3.2 Недопустимые варианты

3.2.1 Отсутствие рисунка

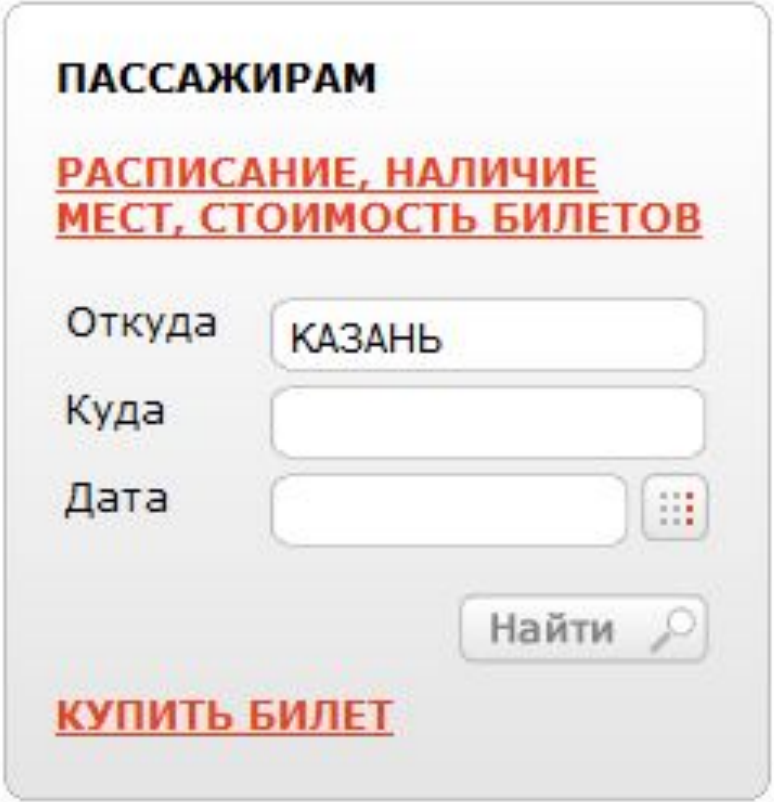
3.2.2 Длиннее четырех сантиметров

3.2.3 Линия не является прямой

3.2.3.1 Что это может быть??? Кривая? Окружность?

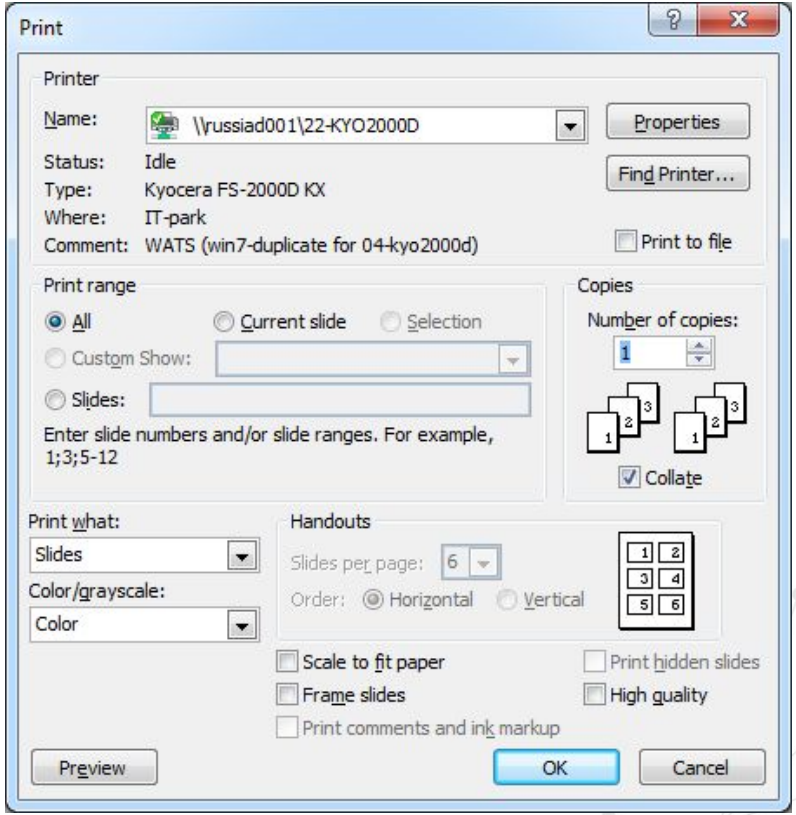
Equivalence classes Task

- Для полей или параметров, принимающих фиксированные перечни значений, выясните, какие из значений в них входят



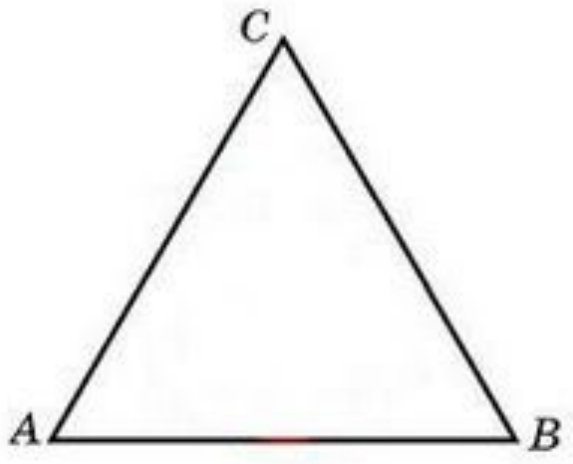
Equivalence classes Task

- Поищите классы значений, зависящих от времени



Equivalence classes Task

- Выявите группы переменных, совместно участвующих в определенных вычислениях, результат которых ограничивается конкретным набором или диапазоном значений



Boundary values

Почему?

- Давно замечено, что при разработке большое число проблем возникает на границах входных переменных
- Даже если эквивалентные классы найдены правильно, то граничные значения могут быть ошибочно отнесены к другому классу

Boundary values

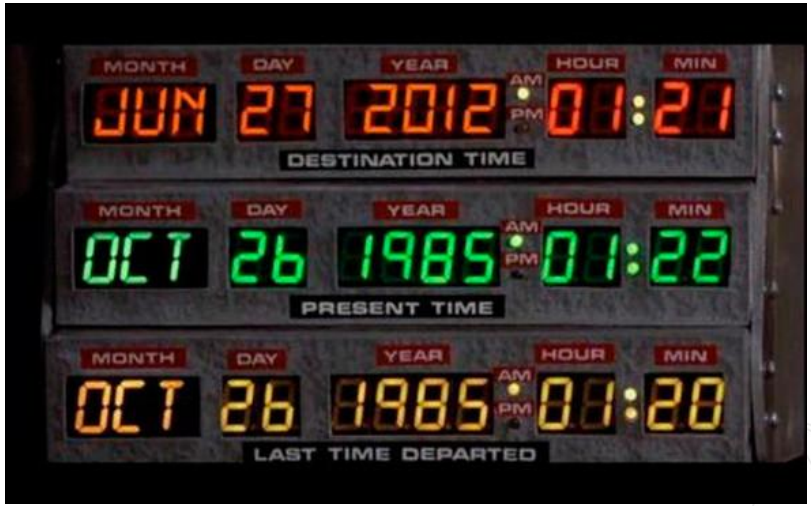
Типы границ:

- Физические (не могут существовать)
- Логические (не имеют смысла)
- Произвольные (так решили)
- Технологические (так устроено)

Boundary values

Физические

- Это такие границы, которые физически нельзя преодолеть
- Пример: строка отрицательной длины



Boundary values

Логические

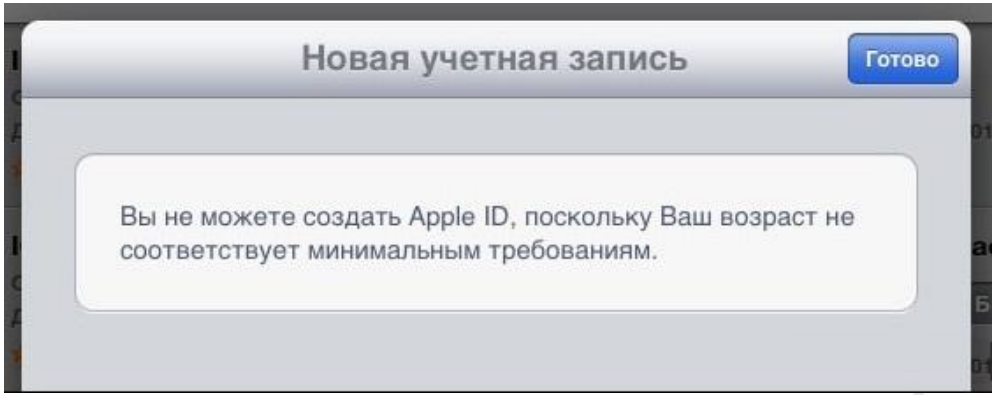
- Определяются здравым смыслом или законами природы. Это границы обусловленные логикой работы самого продукта

- Пример: длина строки – отрицательное число

Boundary values

Произвольные

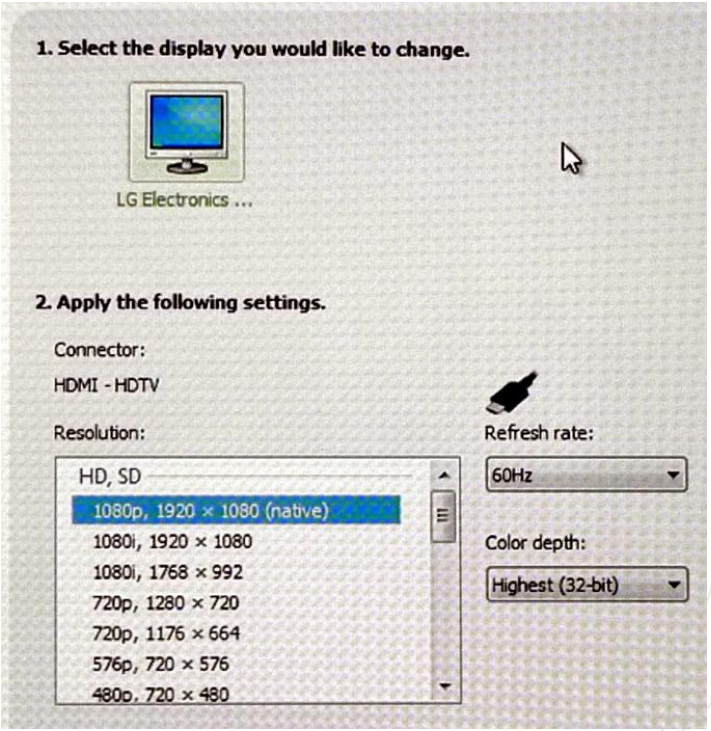
- Данный тип границ отличается от всех остальных тем, что они устанавливаются разработчиком в процессе разработки



Boundary values

Технологические

- Они опасны тем, что они были придуманы или, правильнее сказать, оговорены стандартами и средствами разработки, которые мы с вами используем



Boundary values

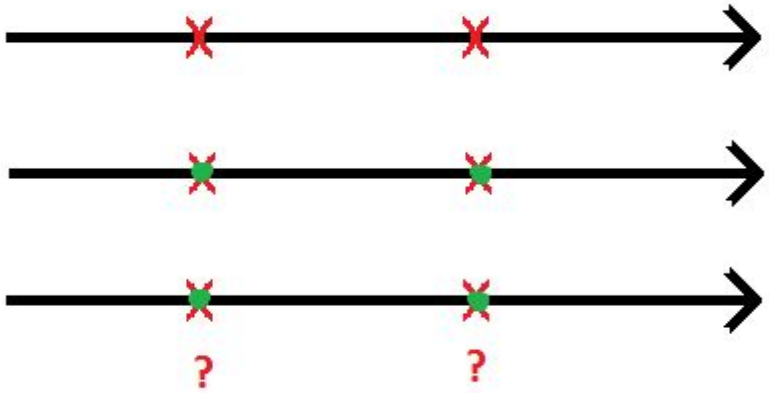
Цель:

- Найти ошибки, связанные с граничными значениями

$$m_x = \frac{\sigma}{\sqrt{n}}$$

Boundary values Algorithm

1. Выделить классы эквивалентности
2. Определить граничные значения классов
3. Понять к какому классу относится граница
4. Провести тесты: на границе, чуть левее, чуть правее



Boundary values Example

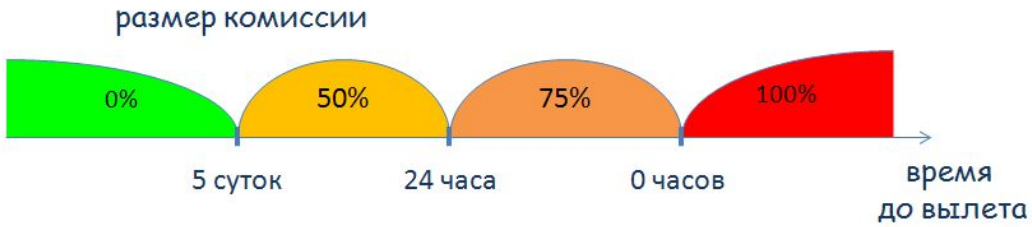
Комиссия при отмене бронирования авиабилета:

- За 5 суток – 0%
- Меньше 5 суток, но больше 24 часов – 50%
- Меньше 24 часов, но до вылета – 75%
- После вылета – 100%

Boundary values Example

1. Выделим классы эквивалентности:

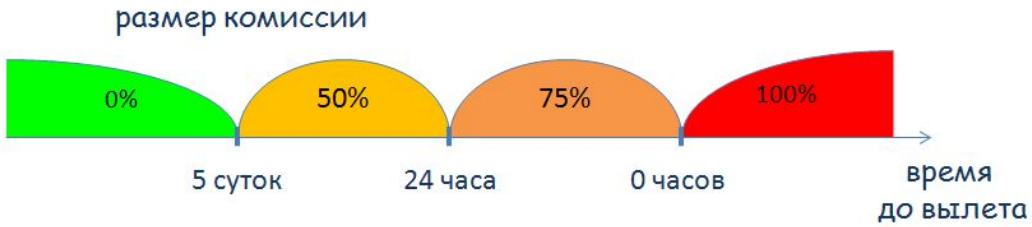
1) t > 5 суток
2) 24 часа < t < 5 суток
3) 0 часов < t < 24 часа
4) t < 0 (вылет состоялся)



Boundary values Example

2. Определим границы:

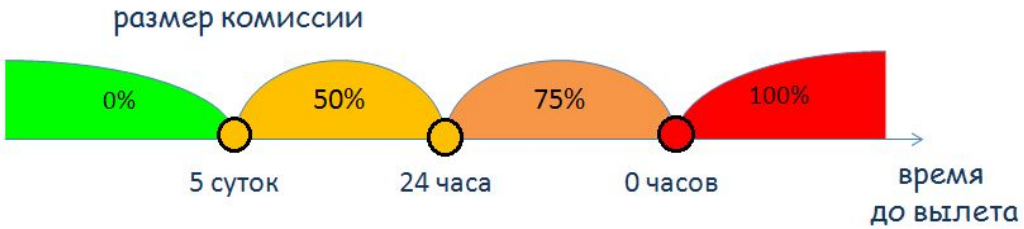
1) 5 суток
2) 24 часа
3) 0 часов



Boundary values Example

3. Определим, к какому классу относятся границы:

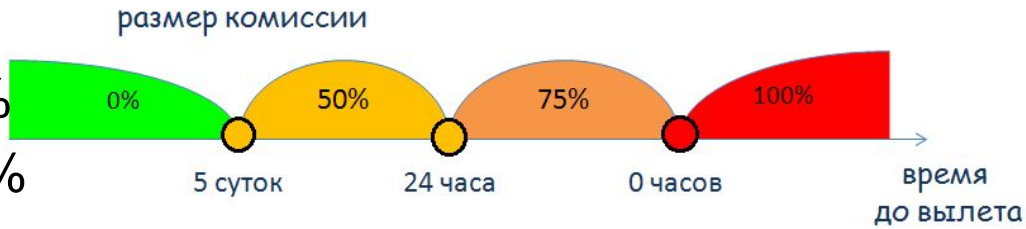
1) 5 суток – 2 класс
2) 24 часа – 2 класс
3) 0 часов – 4 класс



Boundary values Example

3. Тестируем на границе, чуть левее и чуть правее:

1) 5 суток + 1 секунда – 0%
2) 5 суток – 50%
3) 5 суток – 1 секунда – 50%
4) 24 часа + 1 секунда – 50%
5) 24 часа – 50%
6) 24 часа – 1 секунда – 75%
7) 1 секунда до вылета – 75%



8) Время вылета – 100%
9) Через 1 секунду после вылета – 100%

Summary

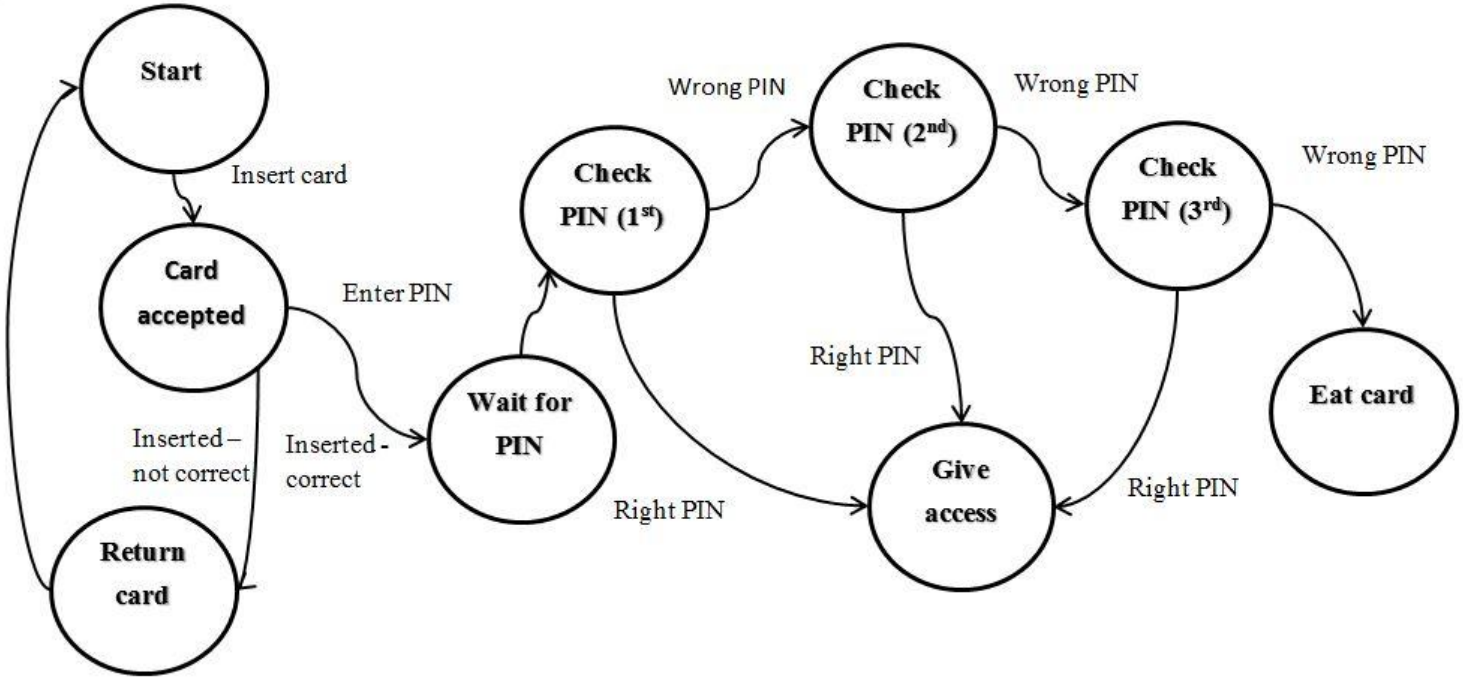
4 + 9 = 13 тестов вместо 950400

State transition testing Definition

- Тестирование состояний переходов – тестирование методом черного ящика, в котором сценарии тестирования строятся на основе выполнения корректных и некорректных переходов состояний

- Используется в случае, если систему можно классифицировать как «машину с конечными состояниями», т.е. система может находиться в конечном количестве разных состояний. Переходы между состояниями определены некоторыми правилами

State transition testing Example



State transition testing Example

Текущее состояние	Действие	Следующее состояние
Start	Insert card	Card accepted
Card accepted	Inserted correctly	Wait for PIN
Card accepted	Inserted incorrectly	Return card
Return card		Start
Wait for PIN	Enter PIN	Check PIN (1st)
Check PIN (1st)	Right PIN	Give access
Check PIN (1st)	Wrong PIN	Check PIN (2nd)
Check PIN (2nd)	Right PIN	Give access
Check PIN (2nd)	Wrong PIN	Check PIN (3rd)
Check PIN (3rd)	Right PIN	Give access
Check PIN (3rd)	Wrong PIN	Eat card

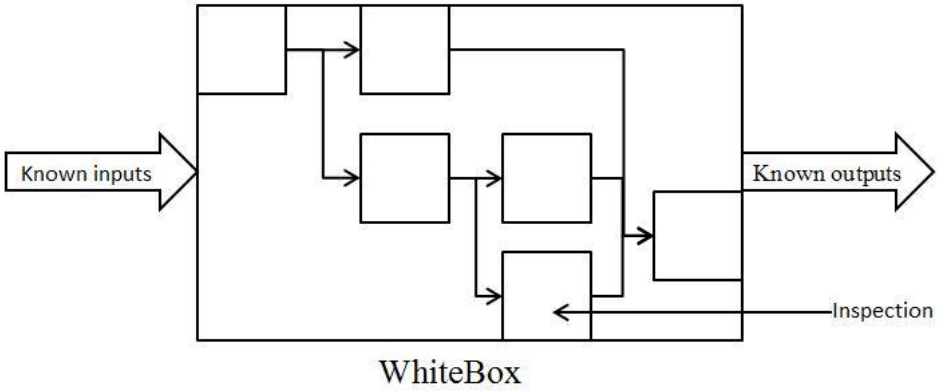
State transition testing

- Протестируйте все наиболее вероятные последовательности действий пользователей
- Если возможно предположить, что действия пользователей в одном режиме могут воздействовать на представление данных или набор представляемых программой возможностей в другом режиме, протестируйте эту зависимость
- Кроме проведения самых необходимых тестов стоит поработать с программой в произвольном режиме, случайным образом выбирая путь ее выполнения

White box test design techniques

Методы:

1. Покрытие операторов
2. Покрытие решений
3. Покрытие условий
4. Комбинаторное покрытие условий



White box test design techniques

- Критерий покрытия операторов подразумевает выполнение каждого оператора программы по крайней мере один раз

```
void func(int a, int b, float x) {
    if ((a > 1) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

a = 2 b = 0 x = 3

White box test design techniques

- Покрытие решений. В соответствии с этим критерием необходимо составить такое число тестов, при которых каждое условие в программе примет как истинное так и ложное значение

```
void func(int a, int b, float x) {
    if ((a > 1) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

a=3, b=0, x=3
a=2, b=1, x=1

White box test design techniques

- Покрытие условий. Записывается число тестов достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены по крайней мере один раз

```
void func(int a, int b, float x) {
    if ((a > 1) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

a=2, b=0, x=4
a=1, b=1, x=1

White box test design techniques

- Комбинаторное покрытие условий. Такое число тестов, чтобы в каждом решении все точки входа выполнялись по крайней мере один раз

```
void func(int a, int b, float x) {
    if ((a > 1) && (b == 0)) x = x/a;
    if (a == 2 || x > 1) x++;
}
```

a=2, b=0, x=4
a=2, b=1, x=1
a=1, b=0, x=2
a=1, b=1, x=1

Testing Methods Practice

- Сколько нужно провести тестов, чтобы получить 100% statement coverage?

```
READ A
READ B
IF A>B
  THEN C = 0
ENDIF
```

1 тест. Например A=10, B=5

Testing Methods Practice

- Сколько нужно провести тестов, чтобы получить 100% branch coverage?

```
READ A
READ B
C = A - 2 *B
IF C <0 THEN
   PRINT "C negative"
END IF
```

2 теста. Например A=11, B=5 (C>=0)
A=10, B=15, (C<0)

Testing Methods Practice

Определите классы эквивалентности и граничные условия.

- Возраст – не менее 18

1) Возраст <18 – недопустимый класс
2) Возраст >=18 – допустимый класс
3) Некорректные символы – недопустимый класс

- Протестируем:

1) 17, 18, 19
2) 1000, abc, $, 0, -1, @, 1$, …