ПОЛИТЕХ
Санкт-Петербургский
политехнический университет
Петра Великого

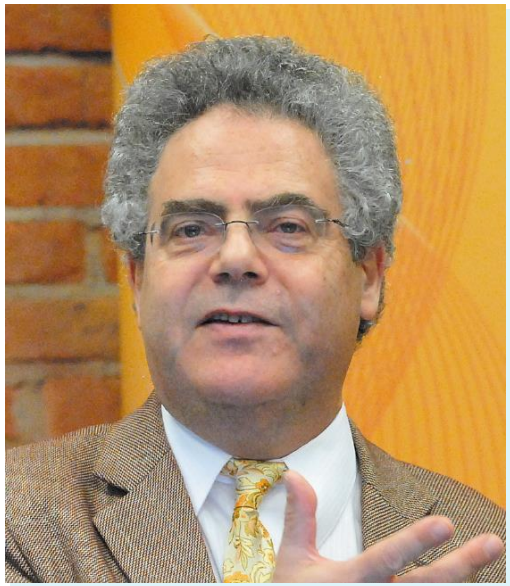
КСПТ
КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ПРОГРАММНЫЕ ТЕХНОЛОГИИ

Технологии разработки программного обеспечения

Контрактное программирование

Контрактное программирование

- Контрактное программирование (программирование по контракту, Design by Contracts, DbC) – подход к созданию программ высокого качества
- Автор подхода – проф. Бертран Мейер
- Впервые введено в языке программирования Eiffel
- 1985 год



Контрактное программирование

- Основная идея – объединить программный код и спецификации
- Спецификации (контракты) встраиваются в программу
- В основе лежит логика Хоара
- Тройка Хоара: {P}C{Q}
  - P и Q – утверждения
  - C – часть программы

Контрактное программирование

- В терминах контрактного программирования метод (или функция) обязуется выполнить контракт:
  - Если на вход поступили данные, **удовлетворяющие** входному условию контракта, то метод **гарантирует** соблюдение выходного условия
  - Если входные данные **не удовлетворяют** первому условию, то ничего **не гарантируется**
  - При этом **соблюдаются** некоторые обобщенные условия

Контрактное программирование

- Входное условие называется предусловием (precondition)
- Выходное условие – постусловием (postcondition)
- Дополнительно обеспечивается поддержка инвариантов.
- Инвариант (invariant) – условие, которое не должно нарушаться из-за выполнения метода, т.е. должно гарантироваться выполнение инварианта до и после выполнения метода
  - Во время выполнения инвариант может быть временно нарушен!

Контрактное программирование

Пример 1. Метод, вычисляющий корни квадратного уравнения $\mathbf{a \cdot x^2 + b \cdot x + c = 0}$

- **void** roots(**in float** a, b, c; **out float** x_1 , x_2)

- Предусловие:
  - $a \neq 0$
- Постусловие:
  - Вариант 1:
    - $a \cdot x_1^2 + b \cdot x_1 + c = 0$
    - $a \cdot x_2^2 + b \cdot x_2 + c = 0$
  - Вариант 2:
    - $x_1 + x_2 = -b/a$
    - $x_1 \cdot x_2 = c/a$

Почему так – плохо?

Контрактное программирование

Пример 2. Снятие денег в банкомате

- **int** money(**int** Cash)
  - Balance – сумма на счете клиента
  - Cash – сумма, которую клиент хочет снять
  - Amount – денежный запас банкомата
  - Результат – сумма, выданная клиенту
- Предусловия:
  - Cash > 0;
  - Cash ≤ 10000;
- Постусловия:
  - **new** Balance = **old** Balance – Cash;
  - **new** Amount = **old** Amount – Cash;
- Инвариант:
  - Amount ≥ 0;
  - Balance ≥ - 1000;

Контрактное программирование

Пример 3. Поиск максимума в массиве

- **int** max(**int** Array[n])
  - Array – массив элементов
  - n – фактический размер массива
  - Результат – номер максимального элемента
- Предусловия:
  - $n > 0$;
- Постусловия:
  - $Result \in [0..n-1]$
  - $\forall\ i \in [0..n-1]: Array[i] \leq Array[Result]$

Контрактное программирование

Пример 4. Сортировка массива

- **void** sort(**int**& Array[n])
  - Array – массив элементов
  - n – фактический размер массива
  - Результат – отсортированный массив по возрастанию
- Предусловия:
  - n > 0
- Постусловия:
  - Для n=1: true
  - Для n>1: $\forall$ i $\in$ [0..n-2]: Array[i] $\leq$ Array[i+1]

Контракты и ООП

- В ООП языках программирования контракты могут являться частью объектной модели
- При наследовании:
  - Предусловия у наследников могут быть ослаблены
  - Постусловия у наследников могут быть усилены
  - Инварианты у наследников могут быть усилены

Контракты и ООП

- Класс **Автомобиль**
  - Метод **Стартовать**
    - Предусловия:
      - Автомобиль стоит
      - Запас топлива > 0
      - Имеется ключ зажигания
    - Постусловия
      - Скорость через 1 секунду > 1км/ч
  - Инвариант
    - Октановое число топлива не ниже 95
    - Скорость < 431 км/ч

Контракты и ООП

- Класс **Мерседес**(**Автомобиль**)
  - Метод **Стартовать**
    - Предусловия:
      - Автомобиль стоит
      - Запас топлива > 0
    - Постусловия
      - Скорость через 1 секунду > 5км/ч
      - Все двери заблокированы
  - Инвариант
    - Октановое число топлива не ниже 98
    - Скорость < 250 км/ч

Использование контрактов

- Проверка всех условий во время исполнения
  - В случае нарушения – останов
  - Возможность отключения проверок в релизах
- Статические проверки
  - Вывод с помощью дедуктивных методов
  - Проверка контрактов с помощью методов статического анализа

Использование контрактов

- Документирование
  - Предусловия + постусловия + интерфейс класса – документирование методов
  - Инварианты + интерфейс класса – документирование классов
- Тестирование
  - Контракты могут служить основой для автоматизированного тестирования:
    - Предусловия и инварианты – ограничения на генерируемые тесты
    - Постусловия и инварианты – основа для построения тестовых оракулов.

Мощность контрактов

- Контракты задаются с помощью выражений логики первого порядка с использованием:
  - целочисленной арифметики
  - вещественной арифметики
  - И т.п.
- Контракты – сугубо декларативное описание требований

Языки программирования, поддерживающие контракты

- Языки со встроенной поддержкой
  - Eiffel
  - SPEC#
  - Fortress
  - D
  - …
- Языки с добавленной поддержкой
  - C# (Code Contracts)
  - Java (JML, Modern Jass, CoFoJa и т.п.)
  - ADA
  - C/C++ (DbC средствами препроцессора)
  - …

Задание контрактов в языке Eiffel

```
class ACCOUNT
create make
feature {NONE} -- Инициализация
	make -- Инициализировать
		do
			create all_deposits
		end
feature -- Доступ
	balance: INTEGER -- Текущий баланс
	deposit_count: INTEGER
		-- Количество сделанных взносов с момента открытия
	do
		Result := all_deposits.count
	end
```

Задание контрактов в языке Eiffel

```
feature -- Изменение элемента
        deposit (sum: INTEGER) -- Добавить `sum' на счет.
        require
                non_negative: sum >= 0
        do
                all_deposits.extend (sum)
                balance := balance + sum
                ensure
                        one_more_dep: deposit_count = old deposit_count + 1
                        updated: balance = old balance + sum
        end
feature {NONE} -- Реализация
        all_deposits: DEPOSIT_LIST -- Список взносов с момента открытия счета
invariant
        consistent_balance: balance = all_deposits.total
        zero_if_no_deposits: all_deposits.is_empty implies (balance = 0)
end -- класс ACCOUNT
```

Задание контрактов с помощью аннотирующих комментариев (JML)

```
public class SqrtExample {
public final static double eps = 0.0001;
//@ requires x >= 0.0;
/*@ ensures JMLDouble.approximatelyEqualTo
  @ (x, \result * \result, eps);
  @*/
public static double sqrt(double x) {
// ...
}

/*@ requires a != null
  @          && (\forall int i; 0 < i && i < a.length;
  @          a[i-1] <= a[i]);
  @*/
int binarySearch(int[] a, int x) {
// ...
}
```

Задание контрактов с помощью аннотаций (CoFoJa)

```
@Invariant("size() >= 0")
interface Stack<T> {
        public int size();

        @Requires("size() >= 1")
        public T peek();

        @Requires("size() >= 1")
        @Ensures({
                "size() == old(size()) - 1",
                "result == old(peek())"
        })
        public T pop();

        @Ensures({
                "size() == old(size()) + 1",
                "peek() == old(obj)"
        })
        public void push(T obj);
}
```