Процедурное
Объектно-Ориентированное
Функциональное

Зачем?

Стариченко Никита

LinkedIn: www.linkedin.com/in/nikita-starichenko/
Telegram: @nikita_starichenko

6+ Years of Experience:

STO solutions, San Francisco USA **01/2020 - now**

- Designing and Developing SPA application for healthcare risk adjustment automation
- Designed architecture of the entire system from scratch

Dodo Pizza, Oxford USA / Moscow Russia **03/2018 – 04/2019**

- Developing and maintaining DODO IS
- Reduced release time by 40% by fixing more than 50 UI tests that led to a decrease of manual testing
- Piloted first microservice on .Net Core and GRPC that is composing by Docker that uses a full CI/CD including integration tests and run in Kubernetes
- Piloted integration React to Angular.js and add ability to step by step rewrite frontend from Angular to React that speeded up front development by 2 times
- Got rid of the need to restart the system by eliminating the daily memory leak of 100mb by finding that leak in .Net Core application using memory snapshot tools on Linux in runtime

Основа

Декларативное:

1. Функциональное

Примеры

1. SQL
2. Regex
3. Haskell
4. F#

Императивное:

1. Процедурное
2. ООП

Примеры

1. C++
2. Java
3. C#
4. Javascript

Популярное

Процедурное:

1. Процедура
2. Запись
3. Массив

Примеры

1. Fortran
2. Pascal
3. C

ООП:

1. Класс
2. Объект
3. Принципы

Примеры

1. C++
2. C#
3. Java
4. Swift

Функциональное:

1. Порядок
2. Чистота
3. Рекурсия
4. Списки

Примеры

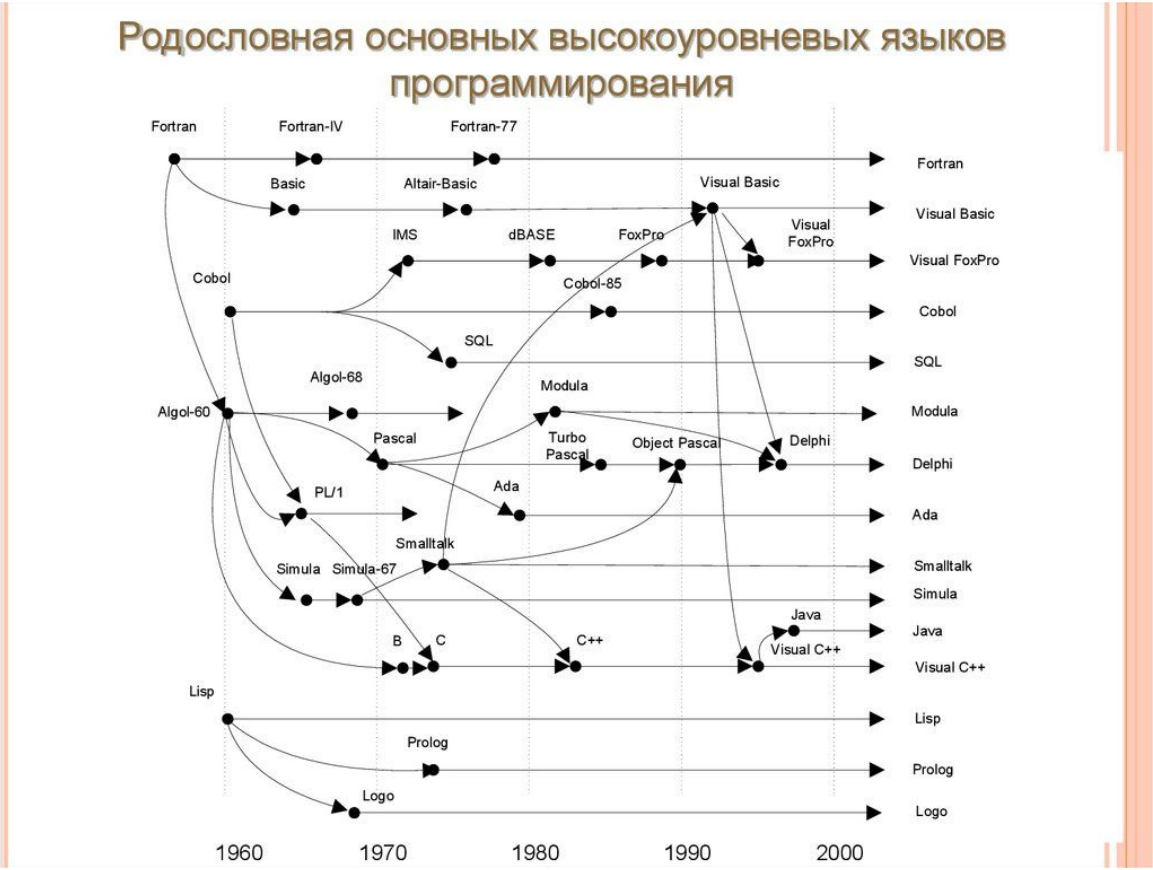
1. Lisp
2. Haskell
3. F#

Зачем так много?

1. Преобразование данных
2. Управление преобразованием
3. Надежность преобразования

1. Прикладное
2. Академическое

История

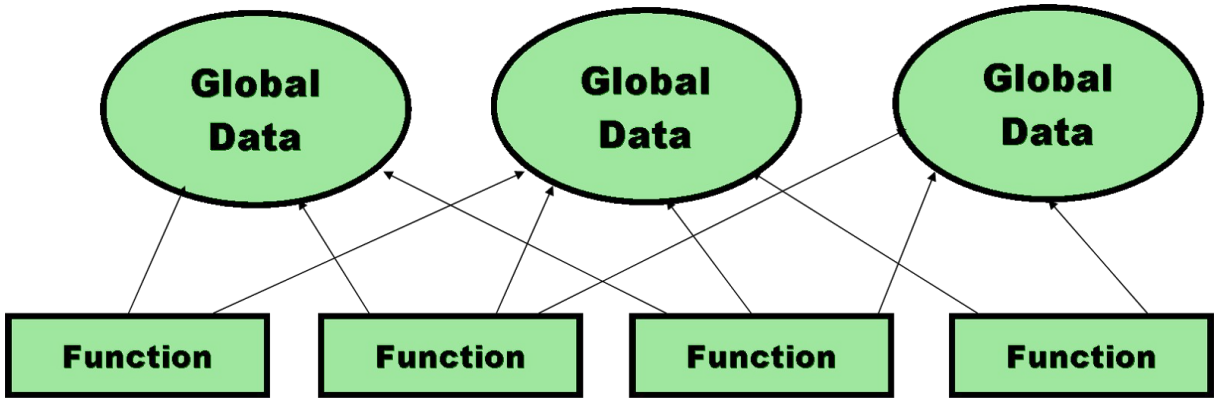

Родословная основных высокоуровневых языков программирования
Fortran
Fortran-IV
Fortran-77
Basic
Altair-Basic
Visual Basic
IMS
dBASE
FoxPro
Visual FoxPro
Cobol
Cobol-85
SQL
Algol-68
Modula
Algol-60
Pascal
Turbo Pascal
Object Pascal
Delphi
PL/1
Ada
Smalltalk
Simula
Simula-67
Java
B
C
C++
Visual C++
Lisp
Prolog
Logo
1960
1970
1980
1990
2000

Процедурное

1. Процедура
2. Запись
3. Массивы

Примеры

1. Fortran
2. Pascal
3. C



The Procedural Paradigm

Примеры

Структурный подход

```
const numbers1 = [1, 2, 3, 4, 5]
let sumOfNumbers1 = 0
for (let index = 0;index < numbers1.length; index++) {
    sumOfNumbers1 += numbers1[index]
}
console.log(sumOfNumbers1)
```

Процедурный подход

```
const numbers1 = [1, 2, 3, 4, 5]
console.log(sumOfNumbers(numbers1))
const numbers2 = [6, 7, 8, 9, 10]
console.log(sumOfNumbers(numbers2))


function sumOfNumbers(numbers) {
    let sumOfNumbers = 0
    for (let index = 0;index < numbers.length; index++) {
        sumOfNumbers += numbers[index]
    }
    return sumOfNumbers
}
```

Плюсы и минусы

Плюсы:

1. Память/CPU

Минусы:

1. Моделирование
2. Как > Что
3. Расширяемость
4. Поддерживаемость

Обзор ООП

1. Класс
2. Объект
3. Абстракция
   Инкапсуляция
   Наследование
   Полиморфизм
4. Паттерны,
   Принципы,
   Подходы

Примеры

1. Smalltalk
2. C++
3. C#
4. Java
5. Swift

Примеры

```
class Numbers {
    constructor(numbers) {
        this.numbers = numbers
    }
    sumOf() {
        let sumOfNumbers = 0
        for (let index = 0;index < this.numbers.length; index++) {
            sumOfNumbers += this.numbers[index]
        }
        return sumOfNumbers
    }
    productOf() {
        let productOfNumbers = 1
        for (let index = 0;index < this.numbers.length; index++) {
            productOfNumbers *= this.numbers[index]
        }
        return productOfNumbers
    }
}
```

```
const numbers1 = new Numbers([1, 2, 3, 4, 5])
console.log(numbers1.sumOf())
console.log(numbers1.productOf())
const numbers2 = new Numbers([6, 7, 8, 9, 10])
console.log(numbers2.sumOf())
console.log(numbers2.productOf())
```

Примеры

```
class SameNumbers extends Numbers {
    constructor(value, times) {
        super(Array(times).fill(value));
        this.value = value
        this.times = times
    }
    sumOf() {
        return this.value * this.times
    }
    productOf() {
        return Math.pow(this.value, this.times)
    }
}
```

```
const numbers1 = new Numbers([1, 2, 3, 4, 5])
console.log(numbers1.sumOf())
console.log(numbers1.productOf())
const numbers2 = new Numbers([6, 7, 8, 9, 10])
console.log(numbers2.sumOf())
console.log(numbers2.productOf())
const numbers3 = new SameNumbers(2, 10)
console.log(numbers3.sumOf())
console.log(numbers3.productOf())
```

Плюсы и минусы

Плюсы:

1. Расширяемость
2. Изоляция
3. Моделирование
4. Что > Как
5. Поддерживаемость

Минусы:

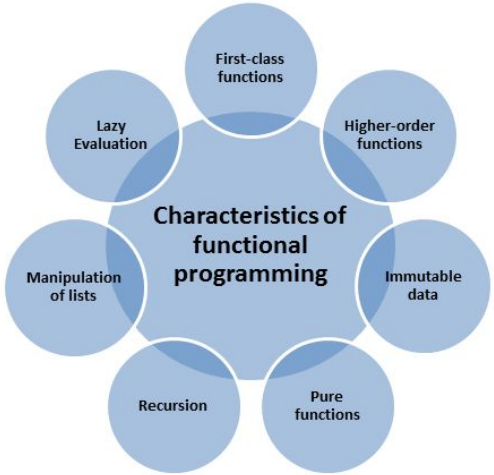
1. Сложность
2. Память/CPU

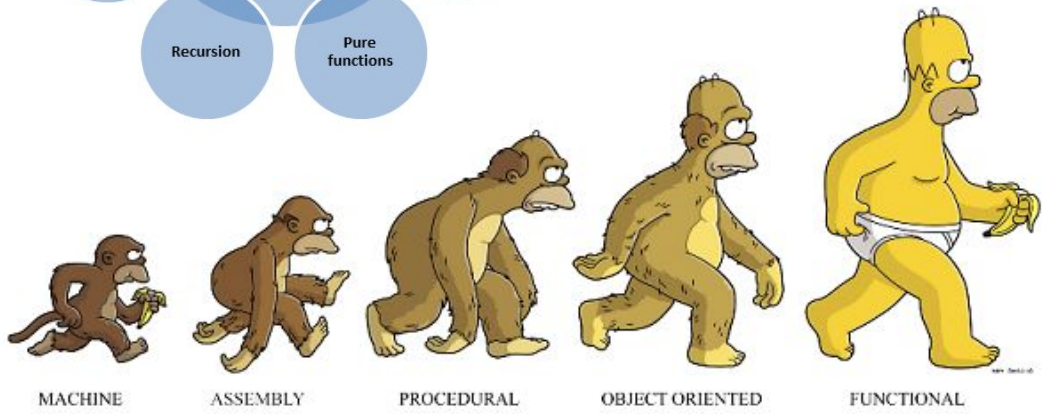
Функциональное

1. Высшие функции
2. Чистые функции
3. Рекурсия
4. Порядок вычисления

Примеры

1. Prolog
2. Lisp
3. Erlang
4. F#
5. Scala
6. Haskell





Примеры

```
class Numbers {
    constructor(numbers) {
        this.numbers = numbers
    }

    map(operator) {
        let result = null
        for (let i = 0;i < this.numbers.length; i++) {
            result = operator(result, this.numbers[i])
        }
        return result
    }
}
```

```
function max(result, current) {
    if (result === null) result = Number.MIN_SAFE_INTEGER
    if (result > current) return result
    else return current
}

const arrayOfNumbers = [
    new Numbers([1, 2, 3, 4, 5]),
    new Numbers([6, 7, 8, 9, 10]),
    new SameNumbers(2, 10),
]

for (let index = 0;index < arrayOfNumbers.length; index++) {
    console.log(arrayOfNumbers[index].map(max))
}
```

Примеры

```
function sum(result, current) {
    if (result === null) result = 0
    return result + current
}

function product(result, current) {
    if (result === null) result = 1
    return result * current
}
```

```
const arrayOfNumbers = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    Array(10).fill(2)
]

const mapOfFunctions = [
    sum, product
]

arrayOfNumbers
    .map(it => mapOfFunctions
        .map(functionItem =>
            it.reduce(functionItem)))
    .flatMap(it => it)
    .forEach(it => console.log(it))
```

Плюсы и минусы

Плюсы:

1. Без состояния
2. Без изменения
3. Тестирование
4. Параллелизм
5. Изоляция

Минусы:

1. Память
2. Как > Что
3. Сложность

Сравнение

Процедурное:

1. Процедура
2. Запись
3. Присваивание

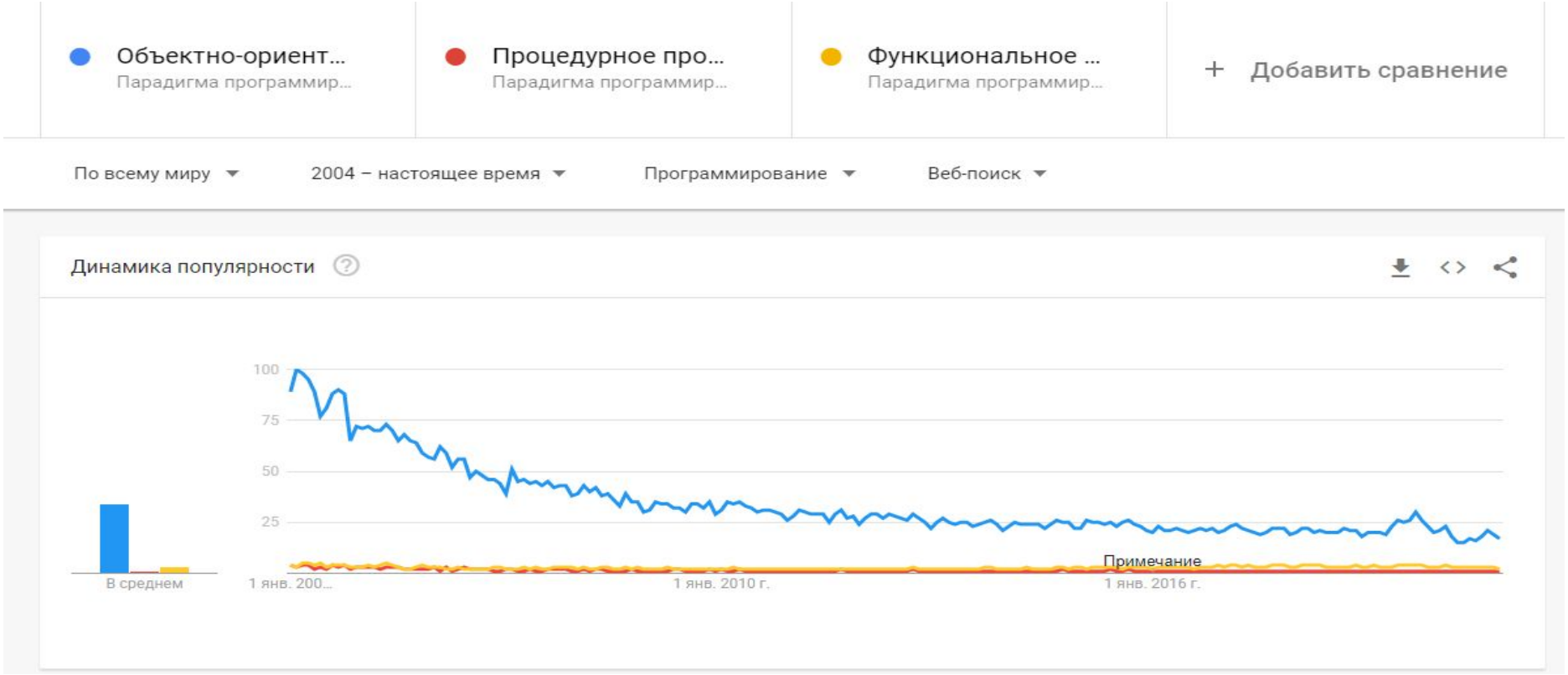
ООП:

1. Класс
2. Объект
3. Изоляция
4. Столпы

Функциональное:

1. Функция
2. Без состояния
3. Без изменения

Популярность



Что использовать?

ООП:

1. Работа с данными
2. Данных становится больше

Функциональное:

1. Работа с операциями
2. Операций становится больше

1. Javascript
2. Python
3. C#
4. Java

Примеры

ООП

Функциональное

Итог:

1. Процедурное -> ООП | Функциональное
2. ООП для данных
3. Функциональное для операций
4. Можно скрещивать подходы
5. Выбирайте головой

LinkedIn: www.linkedin.com/in/nikita-starichenko/ Telegram: @nikita_starichenko