SECON'2017

#SECONRU

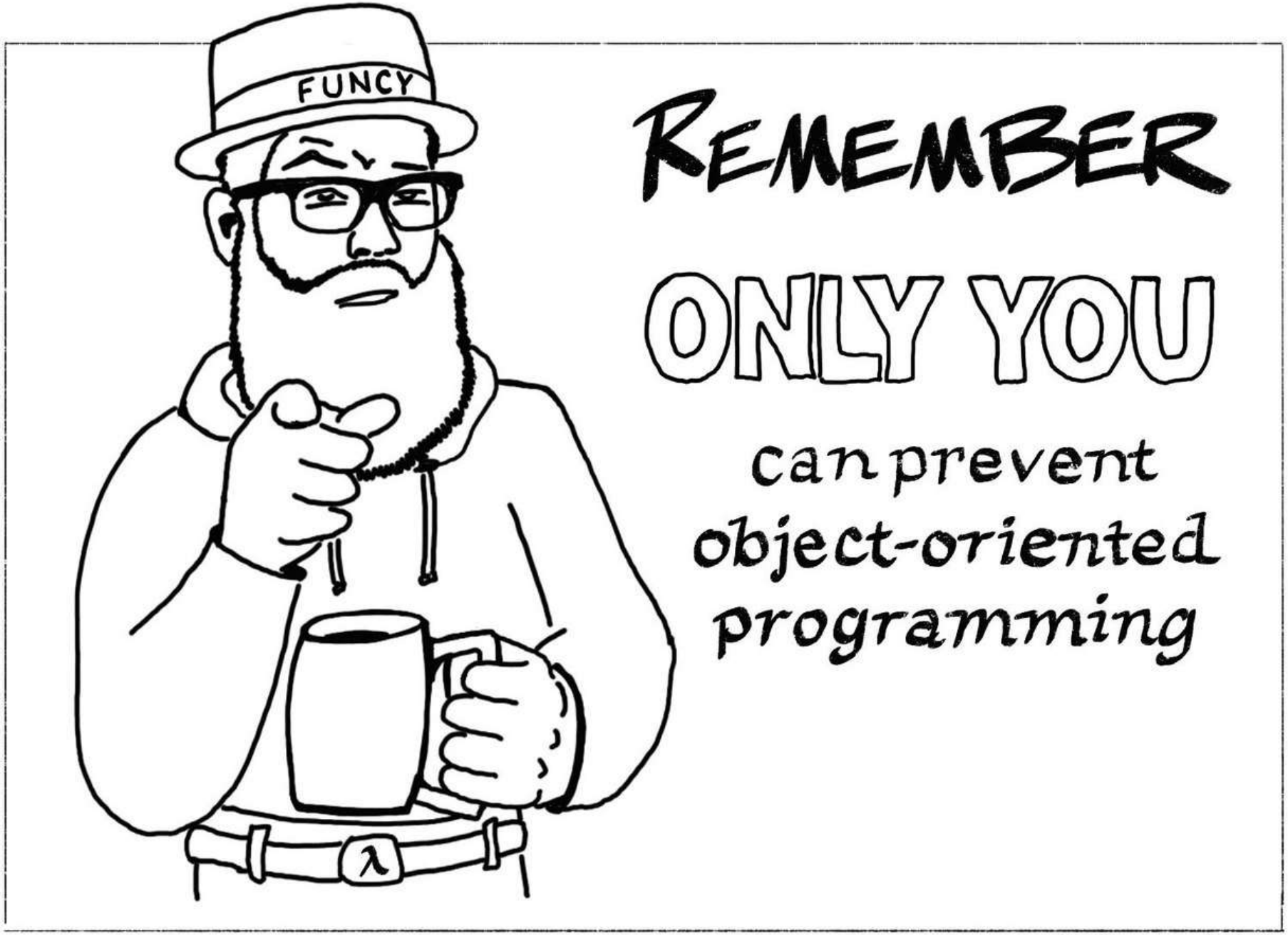
IX МЕЖРЕГИОНАЛЬНАЯ КОНФЕРЕНЦИЯ РАЗРАБОТЧИКОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Функциональный C#

Роман Неволин

Software Engineer, EPAM Systems

21-22 АПРЕЛЯ | ПЕНЗА


FUNCY
λ
REMEMBER
ONLY YOU
can prevent
object-oriented
programming
@Reverent Geek


λ
THE FUNCTIONAL WAY IS THE RIGHT WAY

Когда C# разработчики сталкиваются с функциональщиной?

Когда C# разработчики сталкиваются с функциональщиной?

На самом деле постоянно. Но самый привычный пример – это LINQ

Функциональные принципы на примере LINQ

```
var adminNames = users
                 .Where(x => x.Group == Group.Admin)
                 .Select(x => x.Name);
```

Функциональные принципы на примере LINQ

```
var adminNames = users
                 .Where(x => x.Group == Group.Admin)
                 .Select(x => x.Name);
```

- **Иммутабельность** - в результате операции мы не изменяем существующий объект, а создаем новый.

Функциональные принципы на примере LINQ

```
var adminNames = users
                 .Where(x => x.Group == Group.Admin)
                 .Select(x => x.Name);
```

- **Иммутабельность** - в результате операции мы не изменяем существующий объект, а создаем новый.
- **Функции первого порядка** – LINQ-методы принимают в качестве параметров функции.

Функциональные принципы на примере LINQ

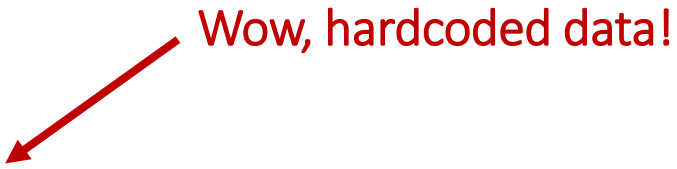
```
var adminNames = users
                 .Where(x => x.Group == Group.Admin)
                 .Select(x => x.Name);
```

- **Иммутабельность** - в результате операции мы не изменяем существующий объект, а создаем новый.
- **Функции первого порядка** – LINQ-методы принимают в качестве параметров функции.
- **Прозрачность** – при вызове методов с одинаковыми данными мы получаем одинаковый результат.

Функциональные принципы : параметризируем все

```
public static List<int> DoubleNumbers()
{
    var numbers = new List<int>() { 1, 2, 3, 4, 5 };
    var doubledNumbers = numbers.Select(x => x * 2).ToList();
    // some logic goes here...
    return doubledNumbers;
}
```

Функциональные принципы : параметризируем все




```
public static List<int> DoubleNumbers()
{
    var numbers = new List<int>() { 1, 2, 3, 4, 5 };
    var doubledNumbers = numbers.Select(x => x * 2).ToList();
    // some logic goes here...
    return doubledNumbers;
}
```

Функциональные принципы : параметризируем все

```
public static List<int> DoubleNumbers(List<int> numbers)
{
    var doubledNumbers = numbers.Select(x => x * 2).ToList();
    // some logic goes here...
    return doubledNumbers;
}
```

Функциональные принципы : параметризируем все

```
public static List<int> DoubleNumbers(List<int> numbers)
{
    var doubledNumbers = numbers.Select(x => x * 2).ToList();
    // some logic goes here...
    return doubledNumbers;
}
```

```
public static List<int> SquareNumbers(List<int> numbers)
{
    var squaredNumbers = numbers.Select(x => x * x).ToList();
    // same logic as in 1st function
    return squaredNumbers;
}
```

Функциональные принципы : параметризируем все

```
public static List<int> ProcessNumbers(List<int> numbers, Func<int, int> process)
{
    var processedNumbers = numbers.Select(process).ToList();
    // some logic goes here...
    return processedNumbers;
}
```

Функциональная «Стратегия»

```
public interface ICalculator
{
    1 reference
    int Calculate(int value);
}


public class StuffMaker
{
    public ICalculator Calculator;

    0 references
    public StuffMaker(ICalculator calculator)
    {
        Calculator = calculator;
    }

    0 references
    public int MakeStuff(int value)
    {
        return Calculator.Calculate(value);
    }
}
```

А как бы вы реализовали это в функциональном стиле?

Функциональная «Стратегия»

```
public static int MakeStuff(int value, Func<int, int> calculator)
{
    return calculator(value);
}
```

А существуют какие-нибудь паттерны функционального программирования?

OO pattern/principle

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

FP pattern/principle

OO pattern/principle

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

FP pattern/principle

- Functions

OO pattern/principle

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

FP pattern/principle

- Functions
- Functions

OO pattern/principle

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

FP pattern/principle

- Functions
- Functions
- Functions, also

OO pattern/principle	**FP pattern/principle**
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	
• Strategy pattern	
• Decorator pattern	
• Visitor pattern	

OO pattern/principle	FP pattern/principle
Single Responsibility Principle	Functions
Open/Closed principle	Functions
Dependency Inversion Principle	Functions, also
Interface Segregation Principle	Functions
Factory pattern	Yes, functions
Strategy pattern	
Decorator pattern	
Visitor pattern	

OO pattern/principle	FP pattern/principle
Single Responsibility Principle	Functions
Open/Closed principle	Functions
Dependency Inversion Principle	Functions, also
Interface Segregation Principle	Functions
Factory pattern	Yes, functions
Strategy pattern	Oh my, functions again!
Decorator pattern	
Visitor pattern	

OO pattern/principle	**FP pattern/principle**
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	

OO pattern/principle	**FP pattern/principle**
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions ☺

OO pattern/principle	**FP pattern/principle**
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions ☺

Seriously, FP patterns are different

Полнота функций

Полнота функций

```
public static List<User> ListUsers()
```

Вопрос : что возвращает эта функция?

Полнота функций

```
public static List<User> ListUsers()
```

Вопрос : что возвращает эта функция?

Ответ : список пользователей, конечно!

Полнота функций

```
public static List<User> ListUsers()
```

Вопрос : что возвращает эта функция?

Ответ : список пользователей, конечно!
или null...

Полнота функций

```
public static List<User> ListUsers()
```

Вопрос : что возвращает эта функция?

Ответ : список пользователей, конечно!
или null…
а может упасть исключение…

Полнота функций : класс Result

```
public class Result<T> {
    0 references
    public T Value { get; set; }
    1 reference
    public ErrorType Error { get; set; }
    0 references
    public bool IsSuccess => Error == ErrorType.None;
}
```

Полнота функций : класс Result

```
var result = ListUsers();

if (result.IsSuccess)
{
    var users = result.Value;
    // process users
}
else if (result.Error == ErrorType.DatabaseError)
{
    // log error
}
```

Цепочка продолжений
Во славу Сатаны

Цепочка продолжений
Во славу Сатаны

Цепочка продолжений

Во славу Са[illegible]ны

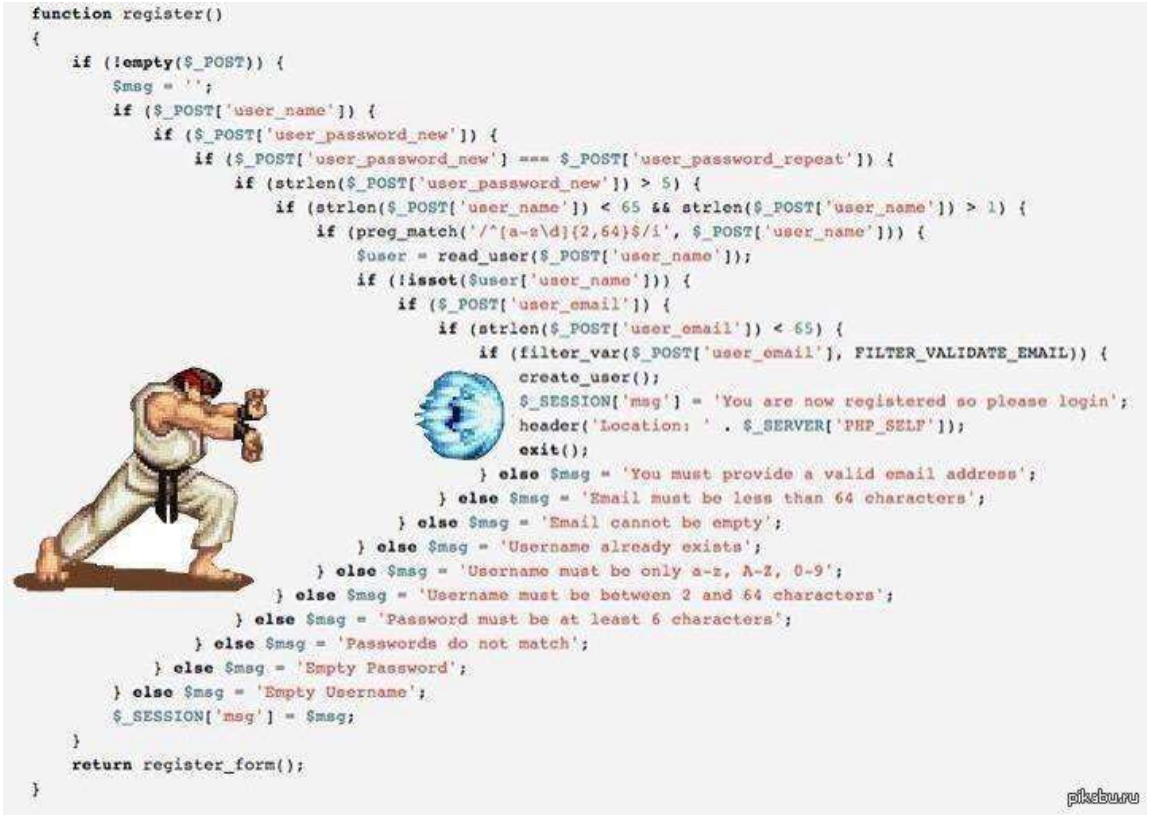
добра

Цепочка продолжений

```
public User Sample(string input)
{
    var a = MakeStuff(input);
    if (a != null)
    {
        var b = MakeOtherStuff(a);
        if (b != null)
        {
            return MakeMoreStuff(b);
        }
        else return null;
    }
    else return null;
}
```

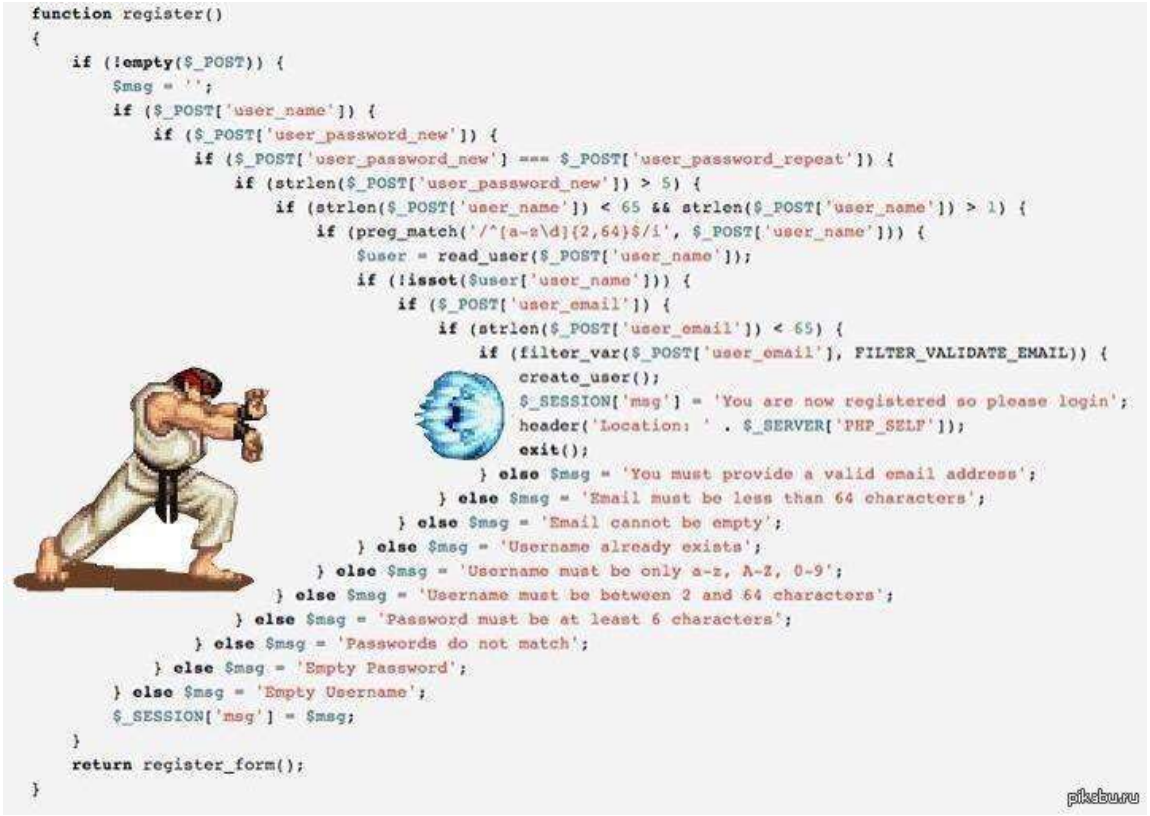
Цепочка продолжений

```
public User Sample(string input)
{
    var a = MakeStuff(input);
    if (a != null)
    {
        var b = MakeOtherStuff(a);
        if (b != null)
        {
            return MakeMoreStuff(b);
        }
        else return null;
    }
    else return null;
}
```



Цепочка продолжений

```
public User Sample(string input)
{
    var a = MakeStuff(input);
    if (a != null)
    {
        var b = MakeOtherStuff(a);
        if (b != null)
        {
            return MakeMoreStuff(b);
        }
        else return null;
    }
    else return null;
}
```



Цепочка продолжений

```
if (input != null) {
    // process data
}
else {
    // return none
}
```

```
public R Bind<T,R>(Func<T,R> nextFunction, T input)
{
    if (input != null)
    {
        return nextFunction(input);
    }
    else return null;
}
```

Цепочка продолжений

```
if (input != null) {
    // process data
}
else {
    // return none
}
```

```
public R Bind<T,R>(Func<T,R> nextFunction, T input)
{
    if (input != null)
    {
        return nextFunction(input);
    }
    else return null;
}
```

Цепочка продолжений

```
public User Sample(string input)
{
    return input
        .Bind(MakeStuff)
        .Bind(MakeOtherStuff)
        .Bind(MakeMoreStuff);
}
```

Цепочка продолжений

```
public User Sample(string input)
{
    return input
        .Bind(MakeStuff)
        .Bind(MakeOtherStuff)
        .Bind(MakeMoreStuff);
}
```

Цепочка продолжений : снова Result

```
public static Result<R> Bind<T,R>(Func<Result<T>,Result<R>> nextFunction,
    Result<T> input) where R : class
{
    if (input.IsSuccess)
    {
        return nextFunction(input);
    }
    else return Result<R>.Fail(input.Error);
}
```

Цепочка продолжений : снова Result

```
public static Result<R> Bind<T,R>(Func<Result<T>,Result<R>> nextFunction,
    Result<T> input) where R : class
{
    if (input.IsSuccess)
    {
        return nextFunction(input);
    }
    else return Result<R>.Fail(input.Error);
}
```

Продолжения во благо валидации

```
public string UpdateUser(Request request)
{
    ValidateRequest(request);
    FormatPhoneNumber(request);
    db.updateDbFromRequest(request);
    smsService.sendMessage(request.Message);

    return "OK";
}
```

Продолжения во благо валидации

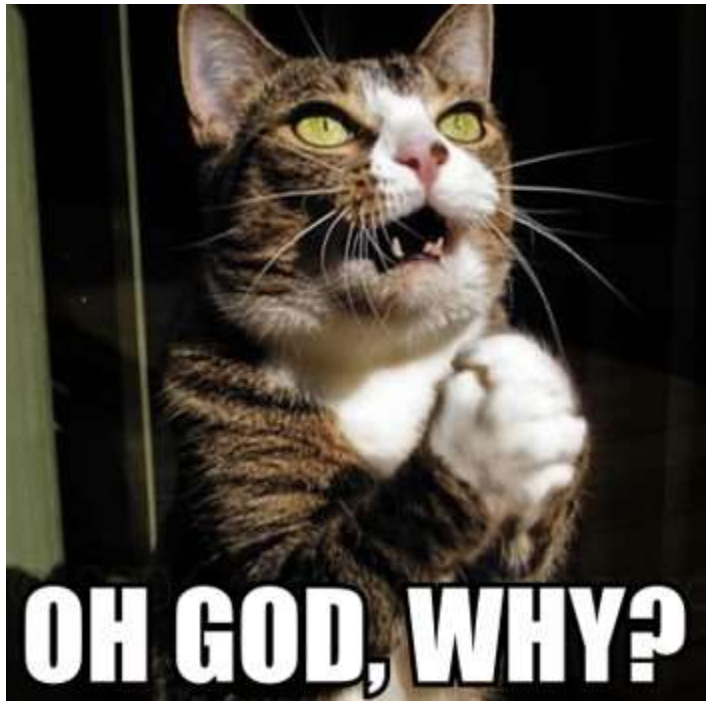
```
public string UpdateUser(Request request)
{
    var validationResult = ValidateRequest(request);
    if (!validationResult)
    {
        return "BAD";
    }
    FormatPhoneNumber(request);
    db.updateDbFromRequest(request);
    smsService.sendMessage(request.Message);

    return "OK";
}
```

Продолжения во благо валидации

```
public string UpdateUser(Request request)
{
    var validationResult = ValidateRequest(request);
    if (!validationResult)
    {
        return "Validation isn't passed";
    }
    FormatPhoneNumber(request);
    var updateResult = db.updateDbFromRequest(request);
    if (!updateResult.Success)
    {
        return "Record can't be updated";
    }
    smsService.sendMessage(request.Message);

    return "OK";
}
```

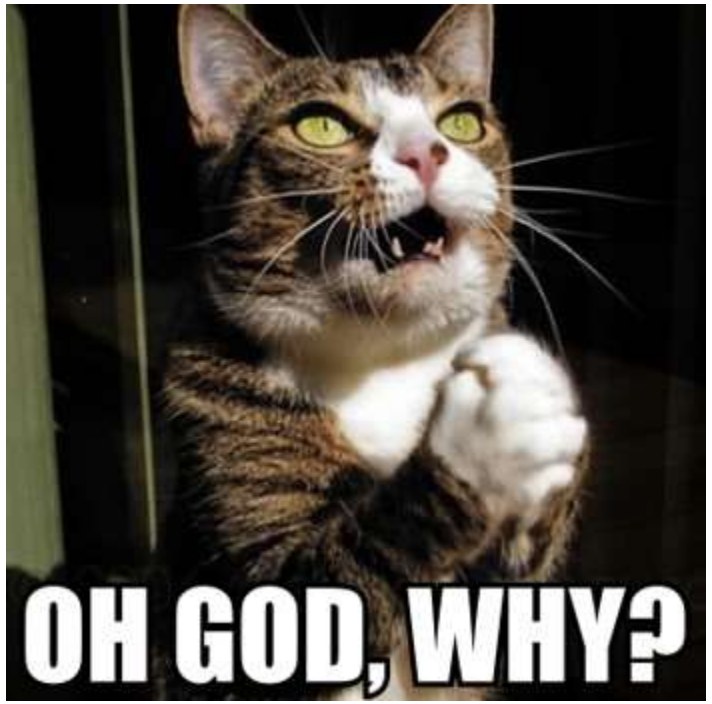
Продолжения во благо валидации

```
public string UpdateUser(Request request)
{
    var validationResult = ValidateRequest(request);
    if (!validationResult)
    {
        return "Validation isn't passed";
    }
    FormatPhoneNumber(request);
    try
    {
        var updateResult = db.updateDbFromRequest(request);
        if (!updateResult.Success)
        {
            return "Record can't be updated";
        }
    }
    catch (DatabaseUpdateException e)
    {
        return e.Message;
    }
    if (!smsService.sendMessage(request.Message))
    {
        logger.LogError("Message was not send");
    }

    return "OK";
}
```



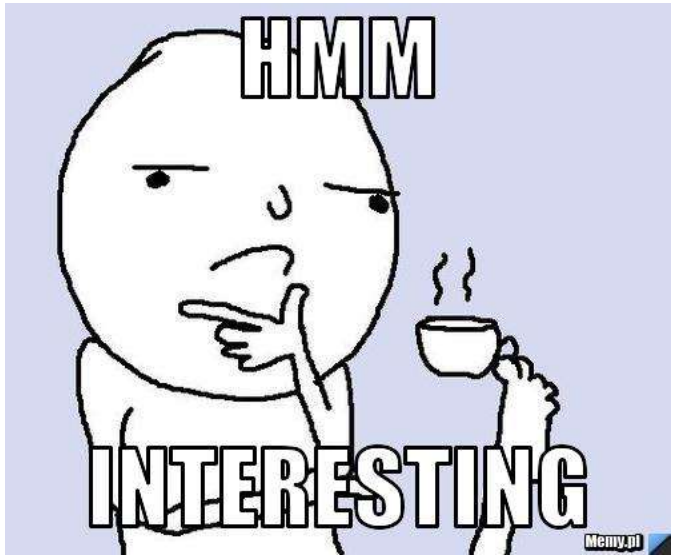
Продолжения во благо валидации

```
public string UpdateUser(Request request)
{
    var validationResult = ValidateRequest(request);
    if (!validationResult)
    {
        return "Validation isn't passed";
    }
    FormatPhoneNumber(request);
    try
    {
        var updateResult = db.updateDbFromRequest(request);
        if (!updateResult.Success)
        {
            return "Record can't be updated";
        }
    }
    catch (DatabaseUpdateException e)
    {
        return e.Message;
    }
    if (!smsService.sendMessage(request.Message))
    {
        logger.LogError("Message was not send");
    }

    return "OK";
}
```



```
public Result<User> UpdateUser(Request request)
{
    return ValidateRequest(request)
        .Bind(FormatPhoneNumber)
        .Bind(UpdateDbFromRequest)
        .Bind(SendMessage)
        .Bind(ReturnMessage);
}
```

```
public Result<User> UpdateUser(Request request)
{
    return ValidateRequest(request)
        .Bind(FormatPhoneNumber)
        .Bind(UpdateDbFromRequest)
        .Bind(SendMessage)
        .Bind(ReturnMessage);
}
```



Демо : рефакторим enterprise код с использованием функциональных принципов

SECON'2017

IX МЕЖРЕГИОНАЛЬНАЯ КОНФЕРЕНЦИЯ РАЗРАБОТЧИКОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Роман Неволин

Software Engineer, EPAM Systems

nevoroman@gmail.com

8 961 800-87-95