Agile Webentwicklung mit Ruby on Rails

Prof. Dr. Oliver Vornberger
Nils Haldenwang, B.Sc.

UNIVERSITÄT OSNABRÜCK

Institut für Informatik
AG Medieninformatik

Institut für Informatik
Prof. Dr. Oliver Vornberger
Nils Haldenwang, B.Sc.

Universität Osnabrück
http://www-lehre.inf.uos.de/~ror
03.05.2012

Agile Webentwicklung mit Ruby on Rails

Sommersemester 2012

Blatt 2

Aufgabe 1: Vector

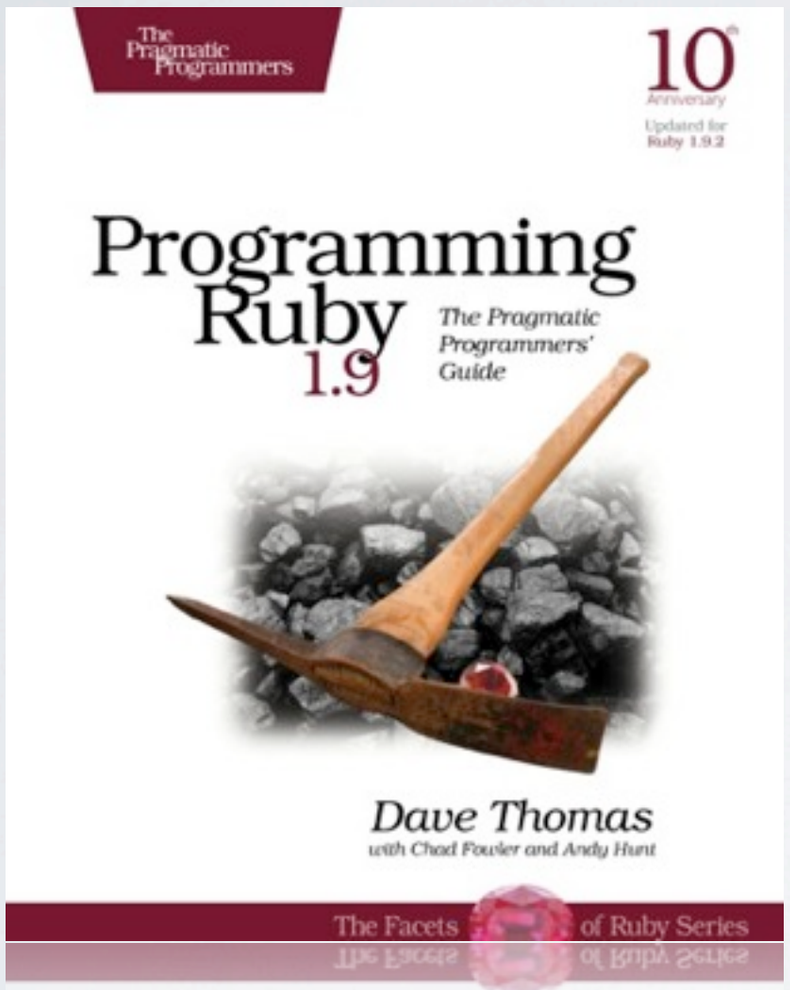
Schreiben Sie eine Klasse `Vector`, welche die folgenden Anforderungen erfüllt:

Der Initializer erwartet als Parameter ein Array mit den Elementen des Vektors. Die Dimension des Vektors soll daraus automatisch bestimmt werden und später auch abfragbar sein.

Ruby

Methoden

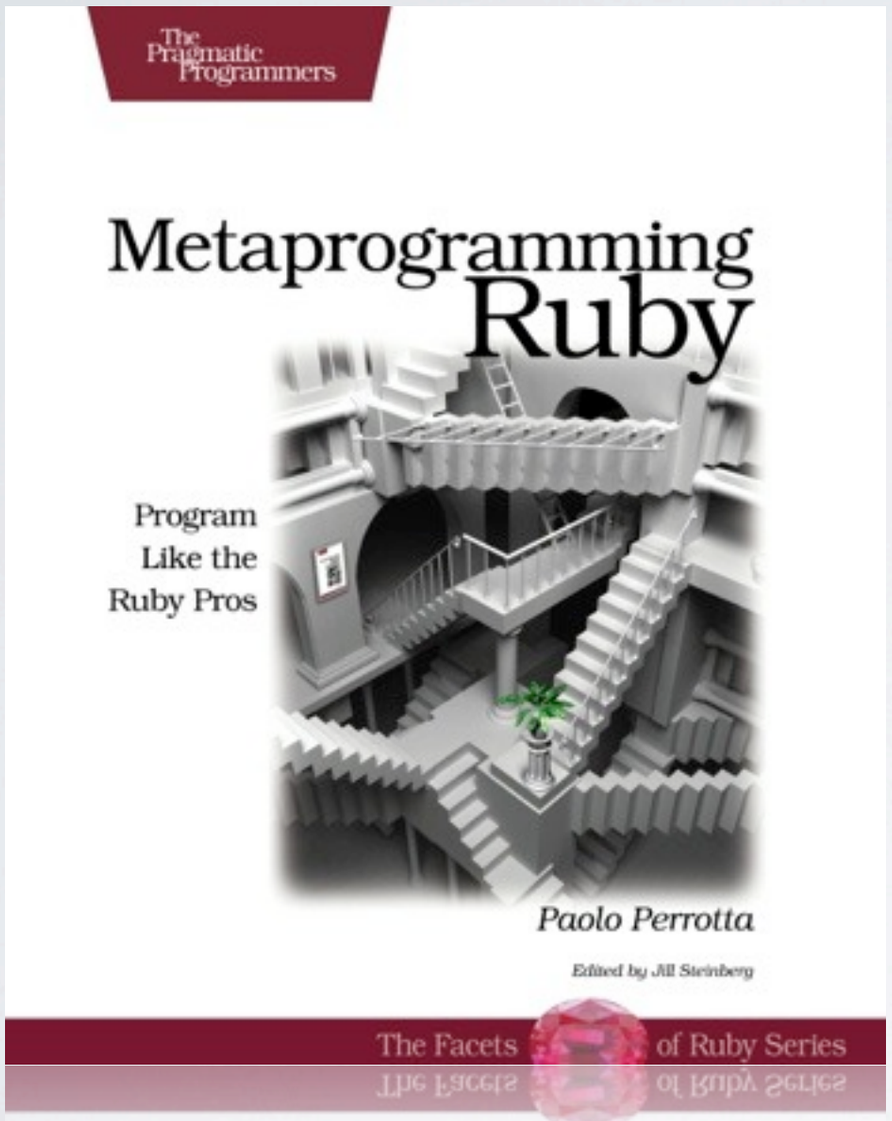
Quellen / Literaturempfehlungen



Dave Thomas,
Chad Fowler,
Andy Hunt,
Programming Ruby 1.9,
Pragmatic Bookshelf, 2009

S. 119-128.
S. 341 f.

Quellen / Literaturempfehlungen



Paolo Perrotta,
Metaprogramming Ruby,
Pragmatic Bookshelf, 2010

S. 38-68.

Parameterübergabe

Ruby

Methoden

Default-Werte

```ruby
def default_args(arg1=42)
  arg1
end

default_args     # => 42
default_args 73 # => 73

def more_default_args(arg1=42, arg2=73)
  "arg1: #{arg1}, arg2: #{arg2}"
end

more_default_args        # => arg1: 42, arg2: 73
more_default_args 23     # => arg1: 23, arg2: 73
more_default_args 23, 5 # => arg1: 23, arg2: 5
```

Variable Argumentlisten

Splat-Operator

```ruby
def varargs(arg1, *rest)
  "arg1: #{arg1}, rest: #{rest}"
end

varargs "one"
# => arg1: one, rest: []

varargs "one", "two"
# => arg1: one, rest: ["two"]

varargs "one", "two", "three"
# => arg1: one, rest: ["two", "three"]
```

Splat-Position beliebig

```ruby
def split_apart(first, *splat, last)
  "First: #{first}, splat: #{splat}, last: #{last}"
end

split_apart 1, 2
# => First: 1, splat: [], last: 2

split_apart 1, 2, 3
# => First: 1, splat: [2], last: 3

split_apart 1, 2, 3, 4
# => First: 1, splat: [2, 3], last: 4
```

Hash-Argumente

```ruby
def hash_params(normal_arg, hash_arg)
  puts "normal_arg: #{normal_arg}, hash_arg: #{hash_arg}"
end

hash_params( 23, { a: 42, b: 73 } )
# => normal_arg: 23, hash_arg: {:a=>42, :b=>73}

hash_params( 23, a: 42, b: 73 )
# => normal_arg: 23, hash_arg: {:a=>42, :b=>73}

songlist.search(
  :singles,
  genre: :jazz,
  duration_less_than: 270
)
```

Block zu Proc konvertieren

```ruby
class TaxCalculator
  def initialize(name, &block)
    @name, @block = name, block
  end

  def tax(amount)
    "#{ @name } on #{ amount }: $#{ @block.call(amount)}"
  end
end

tc = TaxCalculator.new("Sales tax") { |amnt| amnt * 0.075 }

tc.tax(100) # => "Sales tax on 100: $7.5"
tc.tax(250) # => "Sales tax on 250: $18.75"
```

Proc zu Block konvertieren

```
amnt_calculator = lambda { |amnt| amnt * 0.075 }

tc = TaxCalculator.new("Sales tax", &amnt_calculator)

tc.tax(100) # => "Sales tax on 100: $7.5"
tc.tax(250) # => "Sales tax on 250: $18.75"
```

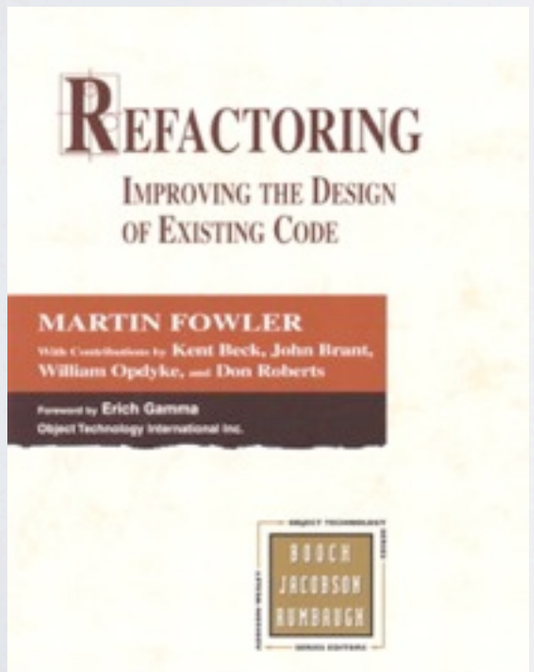
Exkurs: Refactoring

Ruby

Methoden

Refactoring

"Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs."



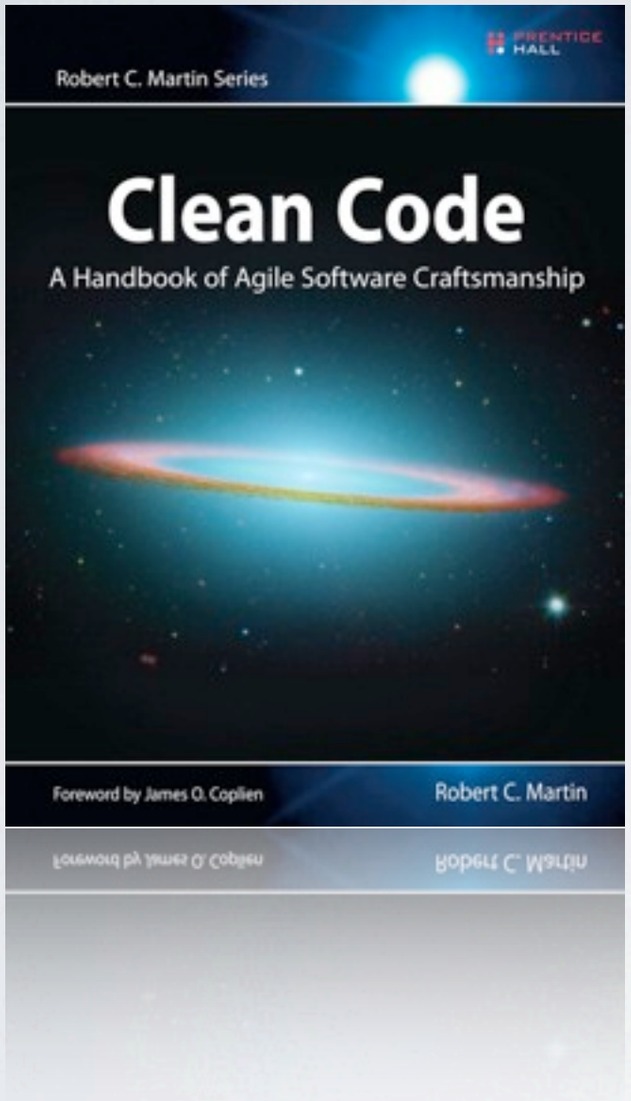
M. Fowler and K. Beck,
Refactoring: Improving the Design of Existing Code.
Addison-Wesley Professional, 1999.

Ruby

Ziel: Clean Code

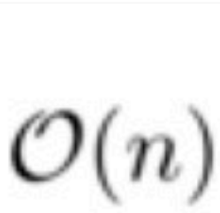
- verbesserte Wartbarkeit und Erweiterbarkeit
  - Fehler zu heben ist einfacher, wenn der Code verständlich und lesbar ist und die Intention des Autors klar kommuniziert.
  - Konventionen und erkennbare Design Patterns erleichtern den Einstieg und das Verständnis.

Clean Code



- Code Conventions
- Design Patterns
- Convention over Configuration
- Don't Repeat Yourself
- ...

R. C. Martin,
Clean Code: A Handbook of Agile Software Craftsmanship.
Pearson Education, 2009.



Computer Science
@CompSciFact

Follow

'No matter how slow you are writing clean code, you will always be slower if you make a mess.' - @UncleBobMartin

Beispielproblem: Workstation Reports

Ruby

Methoden

Legacy System

```
class DS
  def initialize # connect to data source

  def get_mouse_info(workstation_id)      # ...
  def get_mouse_price(workstation_id)     # ...

  def get_keyboard_info(workstation_id)   # ...
  def get_keyboard_price(workstation_id)  # ...

  def get_cpu_info(workstation_id)        # ...
  def get_cpu_price(workstation_id)       # ...

  def get_display_info(workstation_id)    # ...
  def get_display_price(workstation_id)   # ...
  # and so on
end
```

Legacy System

```
ds = DS.new

ds.get_cpu_info(42)     # => 2.16 Ghz
ds.get_cpu_price(42)    # => 150

ds.get_mouse_info(42)   # => Dual Optical
ds.get_mouse_price(42) # => 40
```

Ansatz: Computer-Objekte

```ruby
class Computer
  def initialize(computer_id, data_source)
    @id, @data_source = computer_id, data_source
  end

  def mouse
    info   = @data_source.get_mouse_info(@id)
    price  = @data_source.get_mouse_price(@id)
    result = "Mouse: #{info} ($#{price})"
    result = "* #{result}" if price >= 100
    result
  end

  # and so on
end
```

Dynamische Methodenaufrufe

Ruby

Methoden

Dynamischer Methodenaufruf

```ruby
class MyClass
  def my_method(my_arg)
    my_arg * 2
  end
end

obj = MyClass.new
obj.my_method(3) # => 6

obj.send(:my_method, 3) # => 6
```

Dynamische Aufrufe und Private

```
class MyClass
  private
  def private_method
    puts "private_method()"
  end
end

obj = MyClass.new

obj.send(:private_method)
#  => private_method()
```


Ruby

Dynamic Dispatch

```
class Computer
  # initializer left out

  def mouse
    component :mouse
  end

  # and so on

  private
  def component(name)
    info   = @data_source.send "get_#{name}_info", @id
    price  = @data_source.send "get_#{name}_price", @id
    result = "#{name.to_s.capitalize}: #{info} ($#{price})"
    result = "* #{result}" if price >= 100
    result
  end
end
```

Dynamische Methodendefinitionen

Ruby

Methoden

Dynamische Methodendefinition

```ruby
def MyClass
  define_method :my_method do |my_arg|
    my_arg * 3
  end
end

obj = MyClass.new

obj.my_method(2) # => 6
```

Class Macro

```ruby
class Computer
  # initialize left out

  def self.define_component(name)
    define_method(name) do
      info   = @data_source.send "get_#{name}_info", @id
      price  = @data_source.send "get_#{name}_price", @id
      result = "#{name.to_s.capitalize}: #{info} ($#{price})"
      result = "* #{result}" if price >= 100
      result
    end
  end

  define_component :mouse
  define_component :keyboard
  define_component :cpu
  # and so on
end
```

Introspection

```
class Computer
  def initialize(computer_id, data_source)
    @id, @data_source = computer_id, data_source

    @data_source.methods.grep(/^get_(.*)_info$/) do
      Computer.define_component $1
    end
  end

  def self.define_component(name)
    define_method(name) do
      info   = @data_source.send "get_#{name}_info", @id
      price  = @data_source.send "get_#{name}_price", @id
      result = "#{name.capitalize}: #{info} ($#{price})"
      result = "* #{result}" if price >= 100
      result
    end
  end
```

Ghost Methods

Ruby

Methoden

Ergänzung: Method Lookup

```ruby
class Lawyer; end

nick = Lawyer.new
nick.talk_simple

# undefined method `talk_simple' for
# #<Lawyer:0x007f9b5987bb70> (NoMethodError)

nick.send :method_missing, :talk_simple

# undefined method `talk_simple' for
# #<Lawyer:0x007f9b5987bb70> (NoMethodError)
```

Überschreiben: method_missing

```ruby
class Lawyer
  def method_missing(method, *args)
    puts "You called: #{method} with args (#{args.join(", ")})"
    puts "(You also passed it a block)" if block_given?
  end
end

bob = Lawyer.new
bob.talk_simple(:a, :b) do
  # nothing to do here
end

# You called: talk_simple with args (a, b)
# (You also passed it a block)
```

Dynamic Proxy

```ruby
class Computer
  # initializer left out

  def method_missing(name, *args)
    super unless @data_source.respond_to?("get_#{name}_info")

    info   = @data_source.send "get_#{name}_info", @id
    price  = @data_source.send "get_#{name}_price", @id
    result = "#{name.capitalize}: #{info} ($#{price})"
    result = "* #{result}" if price >= 100
    result
  end

  def respond_to?(method)
    @data_source.respond_to?("get_#{method}_info") || super
  end
end
```

Problem: Vorhandene Methoden

```ruby
my_computer = Computer.new(42, DS.new)
my_computer.display #  => nil

Object.instance_methods.grep(/^d/)
#  => [:dup, :display, ...]
```

Lösung: Blank Slate

```
class Computer
  instance_methods.each do |m|
    pattern = /method_missing|respond_to?/
    undef_method m unless m.to_s =~ pattern
  end
  # ...
end

# blank_slate.rb:3: warning:
#   undefining `object_id' may cause serious problems
# blank_slate.rb:3: warning:
#   undefining `__send__' may cause serious problems

class Computer
  instance_methods.each do |m|
    pattern = /^__|object_id|method_missing|respond_to?/
    undef_method m unless m.to_s =~ pattern
  end
  # ...
end
```

Ghost Method Benchmark

```ruby
class String
  def method_missing(method, *args)
    method == :ghost_reverse ? reverse : super
  end
end

require 'benchmark'
Benchmark.bm do |b|
  b.report 'Normal method' do
    1_000_000.times { "abc".reverse }
  end
  b.report 'Ghost method' do
    1_000_000.times { "abc".ghost_reverse }
  end
end
#                 user     system      total        real
# Normal method  0.280000   0.000000   0.280000 (  0.276858)
# Ghost  method  0.580000   0.000000   0.580000 (  0.581901)
```

Ghost Definition Benchmark

```
class String
  def method_missing(method, *args)

    if method == :ghost_reverse
      String.send(:define_method, :ghost_reverse) { reverse }
    else
      super # args are handled implicit
    end

  end
end
#                      user     system      total        real
# Normal method    0.250000   0.000000   0.250000 (  0.259091)
# Ghost  method    0.580000   0.000000   0.580000 (  0.581901)
# Defined method   0.390000   0.000000   0.390000 (  0.383126)
```

Mehraufwand meist nicht relevant!

Zusammenfassung: Methoden

- Methoden können dynamisch anhand ihres Namens aufgerufen werden
- Zur Laufzeit können Methoden dynamisch definiert und entfernt werden
- Introspection ermöglicht die automatische Definition von Wrappern und erleichtert die Erstellung von Proxy-Klassen
- Mittels *method_missing* lassen sich *Dynamic Proxies* realisieren
- Interferenz mit vorhandenen Methoden kann man durch Einsatz eines *Blank Slate* erreichen
- **Einbußen bei der Performance spielen fast keine Rolle!**

Ruby

Blocks, Procs und Lambdas

Closures

Ruby

Blocks, Procs und Lambdas

Blöcke sind Closures

Binding: Lokale Variablen, Self, Instanzvariablen

```
def my_method
  x = "Goodbye"
  yield "cruel"
end

x = "Hello"

my_method do |y|
  "#{x}, #{y} world."
end
# => Hello, cruel world.
```

Binding bei Aufruf irrelevant

Binding bei Definition relevant

Scopes

Ruby

Blocks, Procs und Lambdas

Scopes

```
v1 = 1

class MyClass
  v2 = 2
  local_variables #  => [:v2]

  def my_method
    v3 = 3
    local_variables
  end

  local_variables # => [:v2]
end

obj = MyClass.new
obj.my_method   # => [:v3]
local_variables # => [:v1, :obj]
```

Scope Gate

Scope Gate

Scope Gate

Scope Gate

Flattening the Scope

```ruby
my_var = "Success"

class MyClass
  # We want to
  # print my_var
  # here...

  def my_method
    # ..and here
  end
end
```

Scope Gate

Scope Gate

Flattening the Scope

```ruby
my_var = "Success"

MyClass = Class.new do
  puts "#{my_var} in the class definition!"

  define_method :my_method do
    puts "#{my_var} in the method!"
  end
end

# => Success in the class definition!

MyClass.new.my_method
# => Success in the the method!
```

Manipulation von Self

Ruby

Blocks, Procs und Lambdas

Blöcke und Self

```
class MyClass
  def my_method
    yield
  end
end

MyClass.new.my_method do
  puts self
end

# => main
```

Self aus Binding des Blocks

Binding und Self

```
class MyClass
  def initialize
    @v = 1
  end
end

obj = MyClass.new

obj.instance_eval do
  self #  => #<MyClass:0x3340dc @v=1>
  @v   #  => 1
end

v = 2
obj.instance_eval { @v = v}
obj.instance_eval { @v } #  => 2
```

Receiver wird *self*

restliches Binding
bleibt unangetastet

Beispiel: Clean Room

```ruby
class CleanRoom
  def complex_calculation
    # ...
  end

  def do_something
    # ...
  end
end

clean_room = CleanRoom.new
clean_room.instance_eval do
  if complex_calculation > 10
    do_something
  end
end
```

Bereitstellen von Helfermethoden

Auswertung des Blocks in kontrollierbarer Umgebung

Callable Objects

Ruby

Blocks, Procs und Lambdas

Callable Objects

impliziter Aufruf des current block

Lambda

Proc

konvertierter Block

```ruby
def my_method
  yield #  => call current block
end
my_method { puts "Hello!" } # => Hello!

l = lambda { puts "Hello, Lambda!" }
l.call # => Hello, Lambda!
l.class # => Proc

p = Proc.new { puts "Hello, Proc!" }
p.call #  => Hello, Proc!
p.class #  => Proc

def other_method(&block)
  block.class
end
#  => Proc
```

Procs vs. Lambdas

```ruby
l = lambda { puts "Hello" }
l.lambda? # => true

p = Proc.new { puts "Hello" }
p.lambda? # => false

def my_method(&block)
  block.lambda?
end

my_method { puts "Hello" }
# => false
```

Procs, Lambdas und Return

```ruby
def lambda_double
  l = lambda { return 10 }
  l.call * 2
end

lambda_double #  => 20

def proc_double
  p = Proc.new { return 10 }
  p.call * 2
end

proc_double #  => 10
```

bei Lambda
nur return
aus dem Block

bei Proc
return
aus der Methode

Ruby

Procs, Lambdas und Arity

```ruby
p = Proc.new { |a,b| [a, b] }
p.arity # => 2

p.call(1, 2)      # => [1, 2]

p.call(1, 2, 3) # => [1, 2]

p.call(1)         # => [1, nil]

l = lambda { |a,b| [a, b] }
l.arity # => 2

l.call(1, 2) # => [1, 2]

l.call(1)
# =>[...] wrong number of arguments
#             (1 for 2) (ArgumentError)
```

Procs behandeln Argumente entsprechend der Erwartung

Lambdas prüfen strikt, ob Argumentliste korrekt

Zusammenfassung: Procs/Lambdas

- Blöcke werden mit dem Binding ausgewertet, in dem sie definiert werden
- Klassen-, Modul- und Methodendefinitionen sind *Scope Gates*
- Das aktuelle Objekt (self) des Bindings eines Blocks lässt sich mit *instance_eval* manipulieren
- Lambdas und Procs verhalten sich unterschiedlich im Bezug auf *return* und *arity*

Ruby