webmasters akademie Nürnberg GmbH

```
def test_validate_e_mail
  @user.e_mail = ''
  assert !@user.save
  assert_equal ["is invalid", "can't be blank"],
                 @user.errors.on(:e_mail)

  @user.e_mail = 'dhjxfgjdxfj'
  assert !@user.save
  assert_equal "is invalid", @user.errors.on(:e_mail)

  @user.e_mail = 'dhjx@fgjdx'
  assert !@user.save
  assert_equal "is invalid", @user.errors.on(:e_mail)

  @user.e_mail = 'dhjx@fgjdxf.j'
  assert !@user.save
  assert_equal "is invalid", @user.errors.on(:e_mail)

  @user.e_mail = 'dhjx@fgjd.fj'
  assert @user.save

  @user.e_mail = 'dh.jx@fgjdx.fj'
  assert @user.save
```

Refactoring Unit Tests

Dipl.-Inf. (FH) Marco Emrich

Jan 2013

Beispiel: Seminar-Shop

Seminar-Shop


AMBULANCE

Specs

- Seminare haben **Name** und **Nettopreis**

Seminar
-name: String -net_price: Float

Ruby

Seminar
-name: String -netPrice: float

Java

Specs

- Seminare haben **Name** und **Nettopreis**
- Seminare berechnen Ihren **Bruttopreis** - außer, sie sind **steuerbefreit**

Seminar
-name: String -net_price: Float -tax_free: Boolean
+gross_price():Float

Ruby

Seminar
-name: String -netPrice: float -taxFree: boolean
+grossPrice():float

Java

Marketing-Abteilung

3-Buchstaben Rabatt

Specs

- Seminare haben **Name** und **Nettopreis**
- Seminare berechnen Ihren **Bruttopreis** - außer, sie sind **steuerbefreit**
- 3-Buchstaben-Seminare* bekommen 5% **Rabatt**

Seminar
-name: String -net_price: Float -tax_free: Boolean
+net_price(): Float +gross_price():Float +discount_rate(): Float +discount(): Float

Ruby

Seminar
-name: String -netPrice: float -taxFree: boolean
+netPrice(): float +grossPrice():float +discountRate(): float +discount(): float

Java

```ruby
class Seminar
  TAX_RATE = 1.19
  THREE_LETTER_DISCOUNT_RATE = 5

  attr_writer :net_price, :tax_free, :name

  def initialize(name, net_price, tax_free)
    @name, @net_price, @tax_free =
      name, net_price, tax_free
  end

  def gross_price
    net_price * tax_rate
  end

  def net_price
    @net_price - discount
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end

  def discount
    @net_price * discount_rate / 100
  end

  def discount_rate
    discount_granted? ?
      THREE_LETTER_DISCOUNT_RATE :
      0
  end

  def discount_granted?
    @name.size < 3
  end

end
```

```ruby
class Seminar
  TAX_RATE = 1.19
  THREE_LETTER_DISCOUNT_RATE = 5

  attr_writer :net_price, :tax_free, :name

  def initialize(name, net_price, tax_free)
    @name, @net_price, @tax_free =
      name, net_price, tax_free
  end

  def gross_price
    net_price * tax_rate
  end

  def net_price
    @net_price - discount
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end

  def discount
    @net_price * discount_rate / 100
  end

  def discount_rate
    discount_granted? ?
      THREE_LETTER_DISCOUNT_RATE :
      0
  end

  def discount_granted?
    @name.size < 3
  end

end
```

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_seminar_should_calculate_correct_gross_prices
    seminar = Seminar.new('OOP', 500, false)

    assert_equal 565.25, seminar.gross_price

    seminar.net_price = 300
    assert_equal 339.15, seminar.gross_price

    seminar.tax_free = true
    assert_equal 285, seminar.gross_price

    seminar.name = 'Objekt-Orientierte Programmierung'
    assert_equal 300, seminar.gross_price
  end

end
```

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_seminar_should_calculate_correct_gross_prices
    seminar = Seminar.new('OOP', 500, false)

    assert_equal 565.25, seminar.gross_price

    seminar.net_price = 300
    assert_equal 339.15, seminar.gross_price

    seminar.tax_free = true
    assert_equal 285, seminar.gross_price

    seminar.name = 'Objekt-Orientierte Programmierung'
    assert_equal 300, seminar.gross_price
  end

end
```

```
Failure:
<565.25> expected but was
<5.95>.
```

3 Minuten

Fehler gefunden?

2 Fehler gefunden?

3 Fehler gefunden?

Alternative Test-Suite: *9 Testfälle*

```ruby
def test_seminar_should_have_the_german_mwst_tax_rate_if_it_is_not_tax_free
  seminar = create_seminar(tax_free: false)
  assert_equal Seminar::TAX_RATE, seminar.tax_rate
end
```

```
Failure:
<1.19> expected but was
<0.0119>.
```

```ruby
def test_discount_granted_should_return_true_if_name_consists_of_3_letters
  seminar = create_seminar(name: 'OOP') # 3 Buchstaben
  assert seminar.discount_granted?
end
```

```
Failure:
<false> is not true.
```

3 Minuten

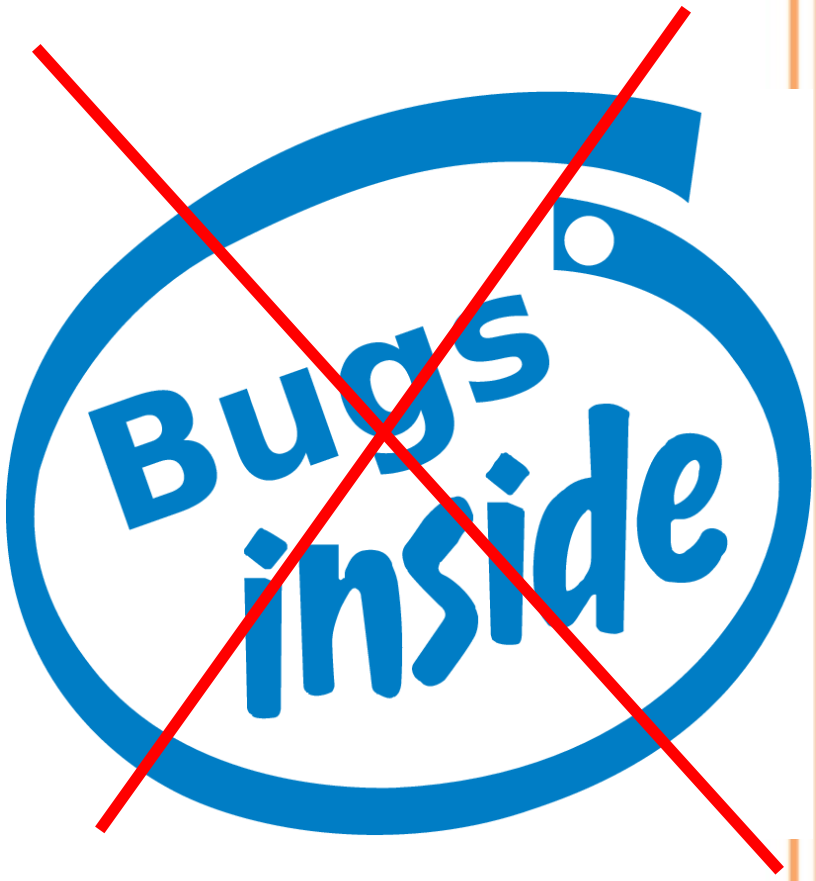
Fehler gefunden?

2 Fehler gefunden?

3 Fehler gefunden?

```ruby
class Seminar
  TAX_RATE = 1.19
  THREE_LETTER_DISCOUNT_RATE = 5

  attr_writer :net_price, :tax_free, :name

  def initialize(name, net_price, tax_free)
    @name, @net_price, @tax_free =
      name, net_price, tax_free
  end

  def gross_price
    net_price * tax_rate
  end

  def net_price
    @net_price - discount
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end

  def discount
    @net_price * discount_rate / 100
  end

  def discount_rate
    discount_granted? ?
      THREE_LETTER_DISCOUNT_RATE :
      0
  end

  def discount_granted?
    @name.size < 3
  end

end
```

```ruby
class Seminar
  TAX_RATE = 1.19
  THREE_LETTER_DISCOUNT_RATE = 5

  attr_writer :net_price, :tax_free, :name

  def initialize(name, net_price, tax_free)
    @name, @net_price, @tax_free =
      name, net_price, tax_free
  end

  def gross_price
    net_price * tax_rate
  end

  def net_price
    @net_price - discount
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE
  end

  def discount
    @net_price * discount_rate / 100
  end

  def discount_rate
    discount_granted? ?
      THREE_LETTER_DISCOUNT_RATE :
      0
  end

  def discount_granted?
    @name.size <= 3
  end

end
```



Vergleich der Testsuites

Qualitätskriterien
für
Testfälle

Qualitätskriterien guter Testfälle

- **hohe Lesbarkeit**, leicht zu verstehen
  - Tests sind schwierig zu testen
  - als Dokumentation der Anforderungen verwendbar => BDD: Specs

Qualitätskriterien guter Testfälle

- hohe Lesbarkeit, leicht zu verstehen
- **hohe Ausführungsgeschwindigkeit**
  - Häufige Ausführung

Qualitätskriterien guter Testfälle

- hohe Lesbarkeit, leicht zu verstehen
- hohe Ausführungsgeschwindigkeit
- **geringer Wartungsaufwand**
  - möglichst Redundanzfrei: DRY
  - geringer Test-Overlap
  - geringe Kopplung an den Produktivcode

Qualitätskriterien guter Testfälle

- hohe Lesbarkeit, leicht zu verstehen
- hohe Ausführungsgeschwindigkeit
- geringer Wartungsaufwand
- **gute Defekt-Lokalisierung**
  - Rainsberger: Jede Testmethode prüft genau eine interessantes Verhaltensweise

Qualitätskriterien guter Testfälle

Lesbar

Schnell

Wartbar

gute Defekt-Lokalisierung

Qualitätskriterien guter Testfälle

manchmal
Widersprüchliche Anforderungen

Qualitätskriterien guter Testfälle

Optimaler Kompromiss?

Testcode muss gewartet werden

Test Refactoring

Test Refactoring

Kosten/Nutzen beachten
für
Jedes Refactoring!

Beispiel

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_seminar_should_calculate_correct_gross_prices
    seminar = Seminar.new('OOP', 500, false)

    assert_equal 565.25, seminar.gross_price

    seminar.net_price = 300
    assert_equal 339.15, seminar.gross_price

    seminar.tax_free = true
    assert_equal 285, seminar.gross_price

    seminar.name = 'Objekt-Orientierte Programmierung'
    assert_equal 300, seminar.gross_price
  end

end
```

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_seminar_should_calculate_correct_gross_prices
    seminar = Seminar.new('OOP', 500, false)

    assert_equal 565.25, seminar.gross_price

    seminar.net_price = 300
    assert_equal 339.15, seminar.gross_price

    seminar.tax_free = true
    assert_equal 285, seminar.gross_price

    seminar.name = 'Objekt-Orientierte Programmierung'
    assert_equal 300, seminar.gross_price
  end

end
```

Remove Redudant Tests
(gleiche Äquivalenzklasse)

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_seminar_should_calculate_correct_gross_prices
    seminar = Seminar.new('OOP', 500, false)

    assert_equal 565.25, seminar.gross_price

    seminar.net_price = 300
    assert_equal 339.15, seminar.gross_price

    seminar.tax_free = true
    assert_equal 475, seminar.gross_price

    seminar.name = 'Objekt-Orientierte Programmierung'
    assert_equal 500, seminar.gross_price
  end

end
```

Remove Redudant Tests

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_seminar_should_calculate_correct_gross_prices
    seminar = Seminar.new('OOP', 500, false)

    assert_equal 565.25, seminar.gross_price

    seminar.tax_free = true
    assert_equal 475, seminar.gross_price

    seminar.name = 'Objekt-Orientierte Programmierung'
    assert_equal 500, seminar.gross_price
  end

end
```

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_seminar_should_calculate_correct_gross_prices
    seminar = Seminar.new('Objekt-Orientierte Programmierung', 500, true)

    assert_equal 500, seminar.gross_price

    seminar.name = 'OOP'
    assert_equal 475, seminar.gross_price

    seminar.tax_free = false
    assert_equal 565.25, seminar.gross_price
  end

end
```

Use **neutral fixture**
(Build up!)

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_a_tax_free_seminar_should_return_a_gross_price_without_tax
    seminar = Seminar.new('Objekt-Orientierte Programmierung', 500, true)
    assert_equal 500, seminar.gross_price
  end

  def test_a_not_tax_free_seminar_should_return_gross_price_with_correct_tax
    seminar = Seminar.new('Objekt-Orientierte Programmierung', 500, false)
    assert_equal 595, seminar.gross_price
  end

  def test_a_3letter_seminar_should_return_a_gross_price_with_discount
    seminar = Seminar.new('OOP', 500, true)
    assert_equal 475, seminar.gross_price
  end

end
```

Split test methods

Fresh fixture

Arrange Act Assert: **AAA**-Pattern

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_a_tax_free_seminar_should_return_a_gross_price_without_tax
    seminar = Seminar.new('Objekt-Orientierte Programmierung', 500, true)
    assert_equal 500, seminar.gross_price
  end

  def test_a_not_tax_free_seminar_should_return_gross_price_with_correct_tax
    seminar = Seminar.new('Objekt-Orientierte Programmierung', 500, false)
    assert_equal 595, seminar.gross_price
  end

  def test_a_3letter_seminar_should_return_a_gross_price_with_discount
    seminar = Seminar.new('OOP', 500, true)
    assert_equal 475, seminar.gross_price
  end

end
```

Use test data factories

Test data factories

- **Ruby-Frameworks**
  - Factory-Girl
  - Machinist
- **Java-Frameworks**
  - Usurper
  - PojoBuilder
- **Patterns**
  - Object Mothers
  - Test Data Builders
  - Test Data Factories (Ruby, JavaScript...)

```ruby
class SeminarTest < Test::Unit::TestCase

  def create_seminar(args = {})
    new_args = {
        :name => 'Object Oriented Programming',
        :net_price => 500,
        :tax_free => true
    }.merge(args)

    Seminar.new(new_args[:name], new_args[:net_price], new_args[:tax_free])
  end

end
```

Defaults
(neutral Fixture)

Use factories

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_a_tax_free_seminar_should_return_a_gross_price_without_tax
    seminar = Seminar.new('Objekt-Orientierte Programmierung', 500, true)
    seminar = create_seminar(tax_free: true)
    assert_equal 500, seminar.gross_price
  end

  def test_a_not_tax_free_seminar_should_return_gross_price_with_correct_tax
    seminar = Seminar.new('Objekt-Orientierte Programmierung', 500, false)
    seminar = create_seminar(tax_free: false)
    assert_equal 595, seminar.gross_price
  end

  def test_a_3letter_seminar_should_return_a_gross_price_with_discount
    seminar = Seminar.new('OOP', 500, true)
    seminar = create_seminar(name: 'OOP')
    assert_equal 475, seminar.gross_price
  end

end
```

Use factories

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_a_tax_free_seminar_should_return_a_gross_price_without_tax
    seminar = create_seminar(tax_free: true)
    assert_equal 500, seminar.gross_price
  end

  def test_a_not_tax_free_seminar_should_return_gross_price_with_correct_tax
    seminar = create_seminar(tax_free: false)
    assert_equal 595, seminar.gross_price
  end

  def test_a_3letter_seminar_should_return_a_gross_price_with_discount
    seminar = create_seminar(name: 'OOP')
    assert_equal 475, seminar.gross_price
  end

end
```

Use factories

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_a_tax_free_seminar_should_return_a_gross_price_without_tax
    seminar = create_seminar(tax_free: true)
    assert_equal seminar.net_price, seminar.gross_price
  end
```

500

```ruby
  def test_a_not_tax_free_seminar_should_return_gross_price_with_correct_tax
    seminar = create_seminar(tax_free: false)
    assert_equal seminar.net_price * Seminar::TAX_RATE, seminar.gross_price
  end
```

500

```ruby
  def test_a_3letter_seminar_should_return_a_gross_price_with_discount
    seminar = create_seminar(name: 'OOP', net_price: 500)
    assert_equal 500 * 0.95, seminar.gross_price
  end

end
```

Use factories

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_a_tax_free_seminar_should_return_a_gross_price_without_tax
    seminar = create_seminar(tax_free: true)
    assert_equal seminar.net_price, seminar.gross_price
  end

  def test_a_not_tax_free_seminar_should_return_gross_price_with_correct_tax
    seminar = create_seminar(tax_free: false)
    assert_equal seminar.net_price * Seminar::TAX_RATE, seminar.gross_price
  end

  def test_a_3letter_seminar_should_return_a_gross_price_with_discount
    seminar = create_seminar(name: 'OOP', net_price: 500)
    assert_equal 500 * 0.95, seminar.gross_price
  end

end
```

add missing test

```ruby
class SeminarTest < Test::Unit::TestCase

  def test_a_tax_free_seminar_should_return_a_gross_price_without_tax
    seminar = create_seminar(tax_free: true)
    assert_equal seminar.net_price, seminar.gross_price
  end

  def test_a_not_tax_free_seminar_should_return_gross_price_with_correct_tax
    seminar = create_seminar(tax_free: false)
    assert_equal seminar.net_price * Seminar::TAX_RATE, seminar.gross_price
  end

  def test_a_3letter_seminar_should_return_a_gross_price_with_discount
    seminar = create_seminar(name: 'OOP', net_price: 500)
    assert_equal 500 * 0.95, seminar.gross_price
  end

  def test_a_more_letters_seminar_should_return_a_net_price_without_discount
    seminar = create_seminar(name: 'Object O. Programming', net_price: 500)
    assert_equal 500, seminar.gross_price
  end

end
```

add missing test

Discount

```
def test_a_3letter_seminar_should_return_a_gross_price_with_discount
  seminar = create_seminar(name: 'OOP')
  assert_equal 500 * 0.95, seminar.gross_price
end

def test_a_more_letters_seminar_should_return_a_net_price_without_discount
  seminar = create_seminar(name: 'Object Oriented Programming')
  assert_equal 500, seminar.gross_price
end
```

```
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def net_price
    @net_price - discount
  end

  def discount
    @net_price * discount_rate / 100
  end
end
```

Isolate

Discount

```
def test_a_3letter_seminar_should_return_a_gross_price_with_discount
  seminar = create_seminar(name: 'OOP')
  assert_equal 500 * 0.95, seminar.net_price .gross_price
end

def test_a_more_letters_seminar_should_return_a_net_price_without_discount
  seminar = create_seminar(name: 'Object Oriented Programming')
  assert_equal 500, seminar.net_price .gross_price
end
```

```
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def net_price
    @net_price - discount
  end

  def discount
    @net_price * discount_rate / 100
  end
end
```

Isolate

Tax

```ruby
def test_a_tax_free_seminar_should_return_a_gross_price_without_tax
  seminar = create_seminar(tax_free: true)
  assert_equal seminar.net_price, seminar.gross_price
end

def test_a_not_tax_free_seminar_should_return_gross_price_with_correct_tax
  seminar = create_seminar(tax_free: false)
  assert_equal seminar.net_price * Seminar::TAX_RATE, seminar.gross_price
end
```

```ruby
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end
end
```

Isolate

Tax

```
def test_a_tax_free_seminar_should_have_a_tax_rate_of_1
  seminar = create_seminar(tax_free: true)
  assert_equal seminar.net_price, seminar.gross_price
  assert_equal 1, seminar.tax_rate
end

def test_a_not_tax_free_seminar_should_have_the_correct_tax_rate
  seminar = create_seminar(tax_free: false)
  assert_equal seminar.net_price * Seminar::TAX_RATE, seminar.gross_price
  assert_equal Seminar::TAX_RATE, seminar.tax_rate
end
```

```
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end
end
```

Isolate

Tax

```ruby
def test_a_tax_free_seminar_should_have_a_tax_rate_of_1
  seminar = create_seminar(tax_free: true)
  assert_equal 1, seminar.tax_rate
end

def test_a_not_tax_free_seminar_should_have_the_correct_tax_rate
  seminar = create_seminar(tax_free: false)
  assert_equal Seminar::TAX_RATE, seminar.tax_rate
end
```

```ruby
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end
end
```

Isolate

Tax

```
def test_a_tax_free_seminar_should_have_a_tax_rate_of_1
  seminar = create_seminar(tax_free: true)
  assert_equal 1, seminar.tax_rate
end

def test_a_not_tax_free_seminar_should_have_the_correct_tax_rate
  seminar = create_seminar(tax_free: false)
  assert_equal Seminar::TAX_RATE, seminar.tax_rate
end
```

```
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end
end
```

Lost Coverage

Tax

```
def test_a_tax_free_seminar_should_have_a_tax_rate_of_1
  seminar = create_seminar(tax_free: true)
  assert_equal 1, seminar.tax_rate
end

def test_a_not_tax_free_seminar_should_have_the_correct_tax_rate
  seminar = create_seminar(tax_free: false)
  assert_equal Seminar::TAX_RATE, seminar.tax_rate
end

def test_seminar_should_use_tax_rate_to_calculate_gross_price



end
```

```
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end

  def net_price
    @net_price - discount
  end
```

Lost Coverage

Tax

```
def test_a_tax_free_seminar_should_have_a_tax_rate_of_1
  seminar = create_seminar(tax_free: true)
  assert_equal 1, seminar.tax_rate
end

def test_a_not_tax_free_seminar_should_have_the_correct_tax_rate
  seminar = create_seminar(tax_free: false)
  assert_equal Seminar::TAX_RATE, seminar.tax_rate
end

def test_seminar_should_use_tax_rate_to_calculate_gross_price
  seminar = create_seminar(tax_free: false)


  assert_equal ?, seminar.gross_price
end
```

```
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end

  def net_price
    @net_price - discount
  end
```

Isolate
Use **Stubs**

Tax

```
def test_a_tax_free_seminar_should_have_a_tax_rate_of_1
  seminar = create_seminar(tax_free: true)
  assert_equal 1, seminar.tax_rate
end

def test_a_not_tax_free_seminar_should_have_the_correct_tax_rate
  seminar = create_seminar(tax_free: false)
  assert_equal Seminar::TAX_RATE, seminar.tax_rate
end

def test_seminar_should_use_tax_rate_to_calculate_gross_price
  seminar = create_seminar(tax_free: false)
  seminar.stubs(net_price: 100)
  seminar.stubs(tax_rate: 1.5)
  assert_equal 150, seminar.gross_price
end
```

```
class Seminar
  def gross_price
    net_price * tax_rate
  end

  def tax_rate
    @tax_free ? 1 : TAX_RATE / 100
  end

  def net_price
    @net_price - discount
  end
```

Isolate
Use **Stubs**

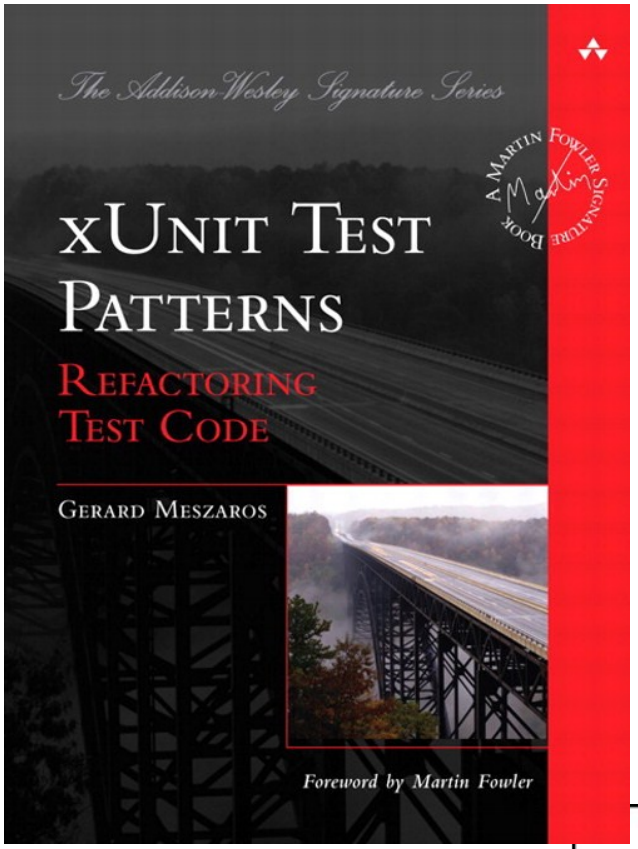
Mocks & Stubs

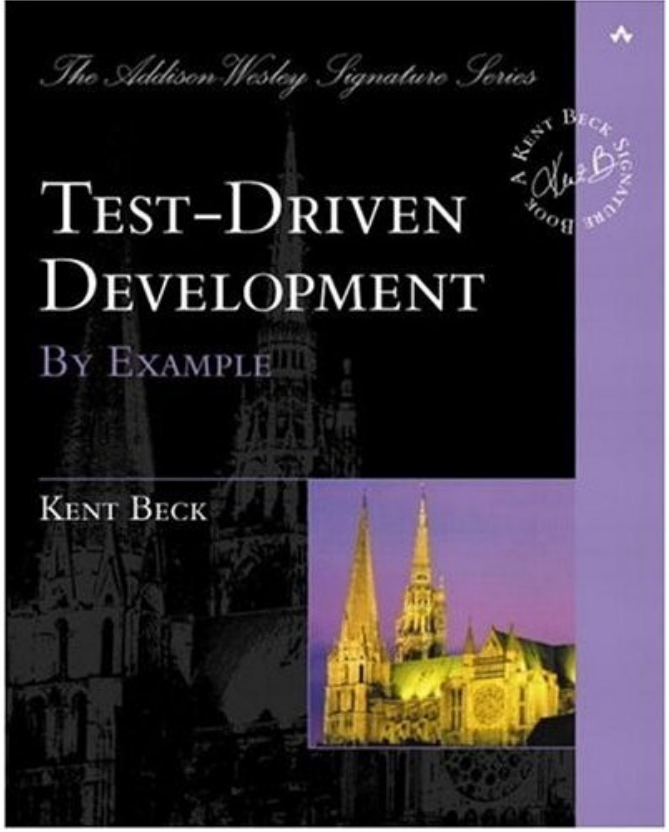
- **Ruby**
  - Rspec-Mocks,
  - Mocha,
  - FlexMock
- **Java**
  - EasyMock
  - Mockito
  - JMockit

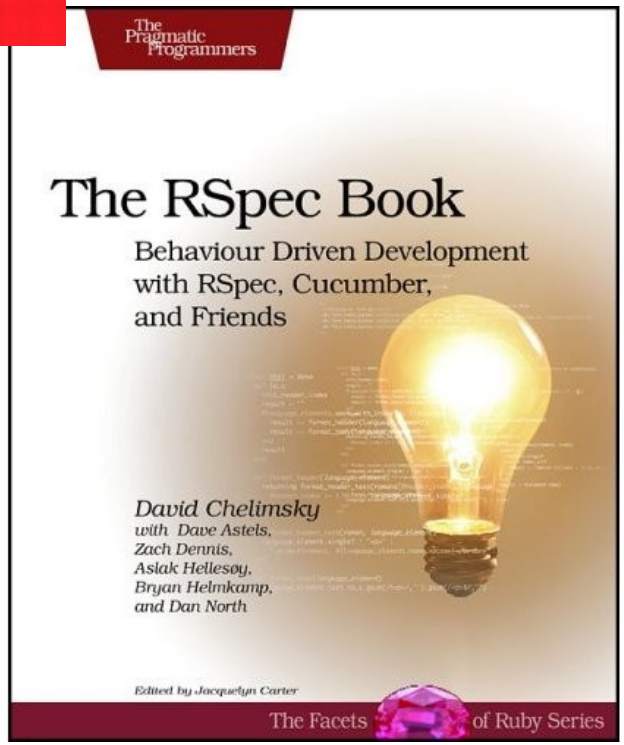
BDD-Frameworks

- **Ruby**
  - RSpec
  - Shoulda
- **Java**
  - JDave
  - ScalaTest
- **JavaScript**
  - Jasmine

Lesbarkeit


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck


The Pragmatic Programmers
The RSpec Book
Behaviour Driven Development with RSpec, Cucumber, and Friends
David Chelimsky
with Dave Astels, Zach Dennis, Aslak Hellesøy, Bryan Helmkamp, and Dan North
Edited by Jacquelyn Carter
The Facets of Ruby Series

m.emrich@webmasters.de

https://github.com/marcoemrich/

https://www.xing.com/profile/Marco_Emrich3

http://twitter.com/marcoemrich