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@fgjdxfj'
  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

Aug 2013 @ Socrates

www.webmasters-akademie.de

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

m.emrich@webmasters.de

http://twitter.com/marcoemrich

Real Code
webmasters.de **2005**

```
def test_validate_password
  @user.password = ''
  @user.password_confirmation = ''
  assert !@user.save
  assert_equal ["is too short (minimum is 5 characters)",
                "has invalid characters", "can't be blank"],
                @user.errors.on(:password)

  @user.password = 'testest'
  @user.password_confirmation = 'testtest'
  assert !@user.save
  assert_equal "doesn't match confirmation" , @user.errors.on(:password)

  @user.password = ''
  @user.password_confirmation = 'testtest'
  assert !@user.save
  assert_equal ["is too short (minimum is 5 characters)",
                "has invalid characters", "doesn't match confirmation",
                "can't be blank"], @user.errors.on(:password)

  @user.password = 'testtest'
  @user.password_confirmation = ''
  assert !@user.save
  assert_equal "doesn't match confirmation", @user.errors.on(:password)

  @user.password = 'test'
  @user.password_confirmation = 'test'
  assert !@user.save
  assert_equal "is too short (minimum is 5 characters)",
               @user.errors.on(:password)

  @user.password = 'testet'
  @user.password_confirmation = 'testet'
  assert @user.save
end
```

```
def test_validate_password
  @user.password = ''
  @user.password_confirmation = ''
  assert !@user.save
  assert_equal ["is too short (minimum is 5 characters)",
                "has invalid characters", "can't be blank"],
                @user.errors.on(:password)

  @user.password = 'testest'
  @user.password_confirmation = 'testtest'
  assert !@user.save
  assert_equal "doesn't match confirmation" , @user.errors.on(:password)

  @user.password = ''
  @user.password_confirmation = 'testtest'
  assert !@user.save
  assert_equal ["is too short (minimum is 5 characters)",
                "has invalid characters", "doesn't match confirmation",
                "can't be blank"], @user.errors.on(:password)

  @user.password = 'testtest'
  @user.password_confirmation = ''
  assert !@user.save
  assert_equal "doesn't match confirmation", @user.errors.on(:password)

  @user.password = 'test'
  @user.password_confirmation = 'test'
  assert !@user.save
  assert_equal "is too short (minimum is 5 characters)",
               @user.errors.on(:password)

  @user.password = 'testet'
  @user.password_confirmation = 'testet'
  assert @user.save
end
```


But there is still Hope

Test Refactoring

Example

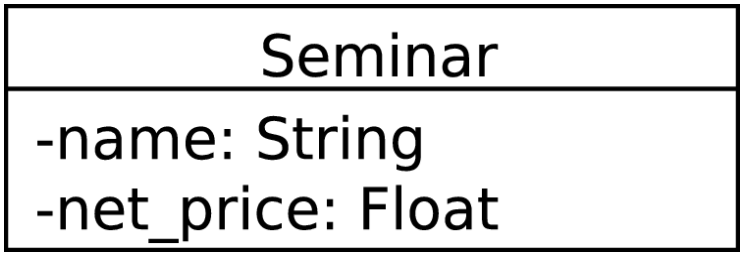
Seminar-Shop

Seminar-Shop


AMBULANCE

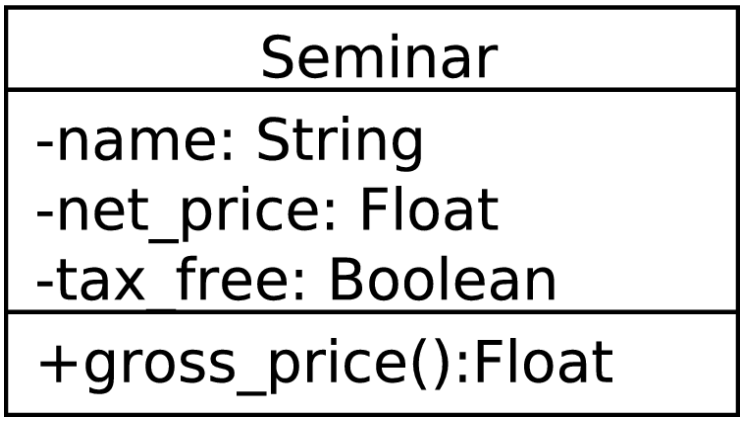
Specs

- a Seminar has a **name** and **net price**



Specs

- a Seminar has **name** and **net price**
- a Seminar calculates it's **gross price** except - it was freed from tax (VAT)

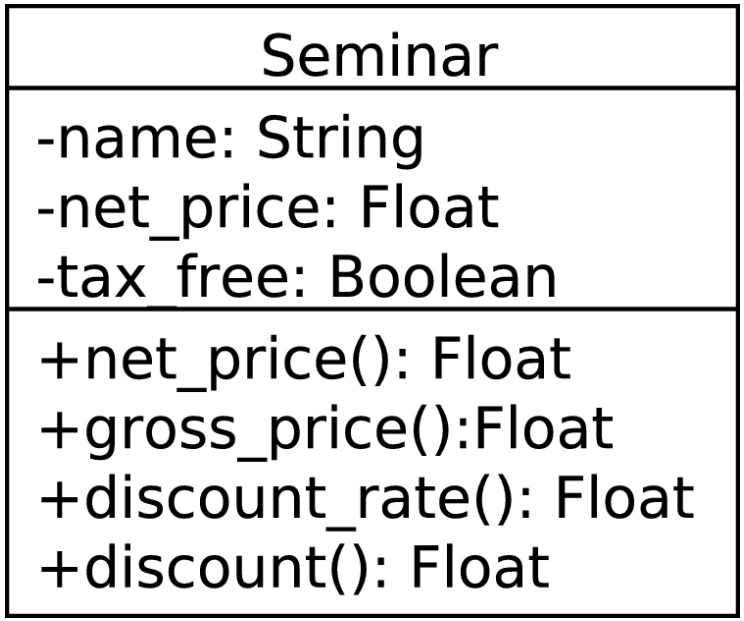


Crazy ideas from **Marketing Department**

3-Letter Discount

Specs

- a Seminar has **name** and **netprice**
- a Seminar calculates it's **gross price** except - it was freed from tax (VAT)
- there is a 5% **discount** on 3-Letter-Seminars



```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 minutes

Found any bugs?

2 bugs?

3 bugs?

Alternative Test-Suite: *9 test cases*

```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 minutes

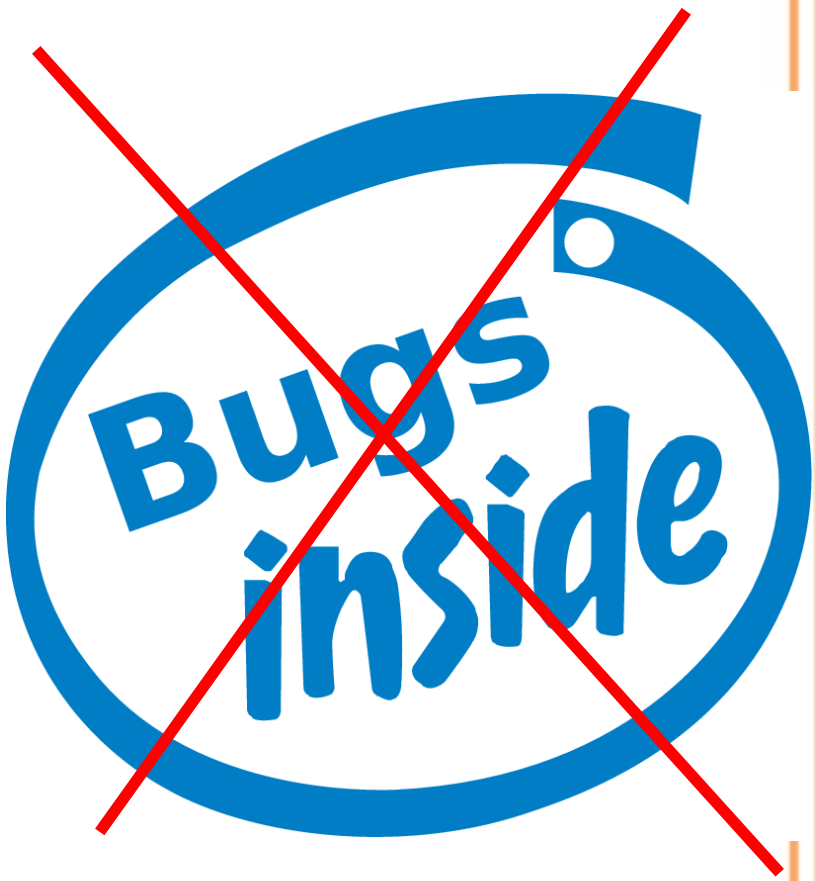
Found any bugs?

2 bugs?

3 bugs?

```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
```



Compare the suites

Quality Criteria
for
test cases

Test Case Quality Criteria

- **readable**, easy to understand
  - Tests are difficult to test
  - Test documents the requirements
    => BDD: Specs

Test Case Quality Criteria

- readable
- **fast**
  - Early feedback
  - Run them often

Test Case Quality Criteria

- readable
- fast
- **maintainable**
  - No duplication/redundancies: DRY
  - No Test-Overlap (test isolation, orthogonality)
  - Reduce coupling to production code

Test Case Quality Criteria

- readable
- fast
- maintainable
- **localizing** – defect localization
  - Rainsberger: a test method checks exactly one interesting behavior

Test Case Quality Criteria

readable

fast

maintainable

localizing

Test Case Quality Criteria

sometimes
contradict each other

Test Case Quality Criteria

best compromise?

Kent Beck

TDD

BDD

Dan North

David Astels

TDD/BDD help to archive good balance

TDD/BDD is not perfect

Improvement is always possible

Legacy Test Suites

Test code needs to be maintained!

Test Refactoring

Example

```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 Redundant Tests
(same equivalence class)

```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: Example Factory

```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

  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

  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

```ruby
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
```

```ruby
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

```ruby
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
```

```ruby
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
```

Isolate

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

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

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
```

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

```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

def test_seminar_should_use_tax_rate_to_calculate_gross_price
  seminar = create_seminar(tax_free: false)


  assert_equal ?, 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

  def net_price
    @net_price - discount
  end
```

Isolate
Use **Stubs**

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

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
```

```ruby
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**

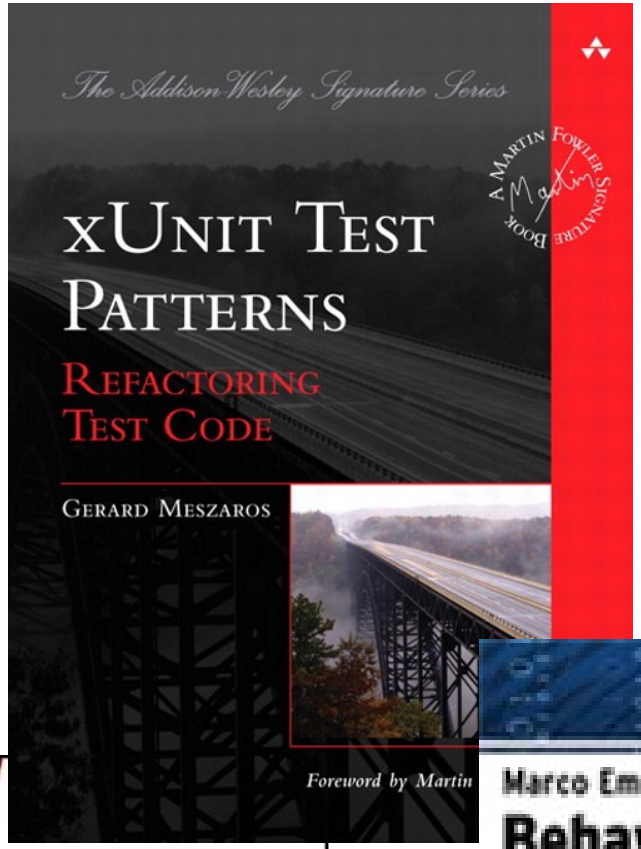
Mocks & Stubs

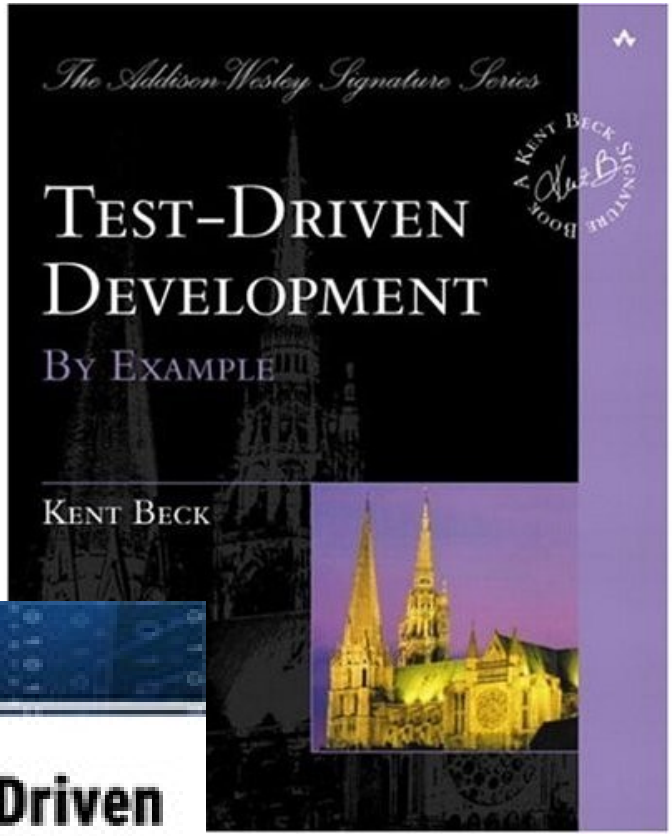
- **Ruby**
  - Rspec-Mocks,
  - Mocha,
  - FlexMock
- **Java**
  - EasyMock
  - Mockito
  - Jmockit
- **JavaScript**
  - Included in Jasmine

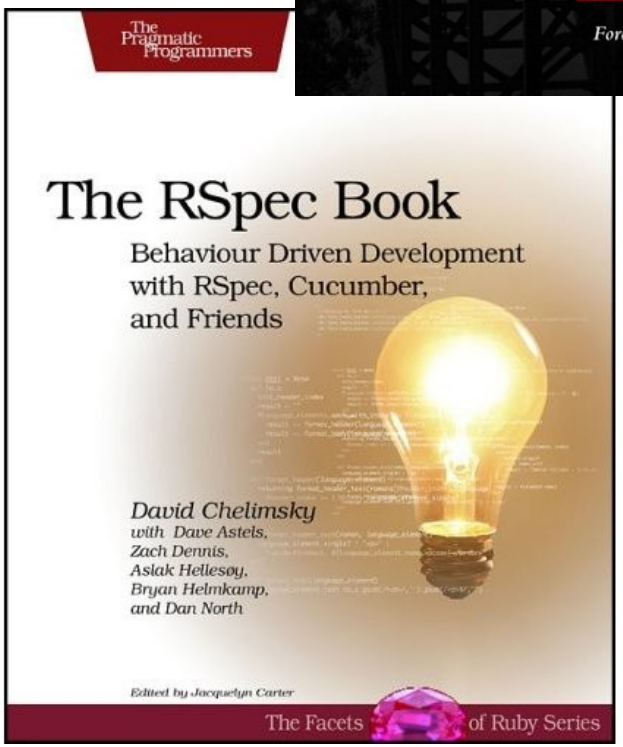
BDD-Frameworks

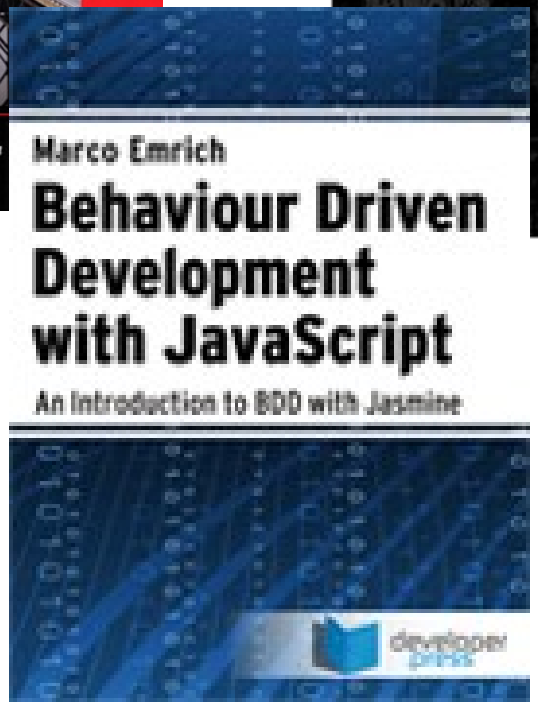
- **Ruby**
  - RSpec
  - Shoulda
- **Java**
  - Jdave
  - Jnario
  - ScalaTest
- **JavaScript**
  - Jasmine

more readable!


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


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


Marco Emrich
Behaviour Driven Development with JavaScript
An Introduction to BDD with Jasmine