RSpec and Cucumber Beyond the Basics

describe it with rspec

Cucumber

Behaviour Driven Development with elegance and joy

David Chelimsky

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

goals

focus on behaviour

executable documentation

readability/ expressiveness

optimize TDD process

pending

3 methods

not yet implemented

not yet implemented

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais"
  end
end
```

not yet implemented

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais"
  end
end
```

not yet implemented

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais"
  end
end
```

no block

not yet implemented

```
Account
  when first created
    has a balance of zero reais (PENDING: Not Yet Implemented)

Pending:

Account when first created has a balance of zero reais (Not Yet Implemented)
./pending.rb:6

Finished in 0.001638 seconds

1 example, 0 failures, 1 pending
```

not yet implemented

```
Account
  when first created
    has a balance of zero reais (PENDING: Not Yet Implemented)

Pending:

Account when first created has a balance of zero reais (Not Yet Implemented)
./pending.rb:6

Finished in 0.001638 seconds

1 example, 0 failures, 1 pending
```

short circuit

short circuit

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais" do
      pending "money needs to support more than :usd"
      account = Account.new
      account.balance.should == Money.new(0, :brl)
    end
  end
end
```

short circuit

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais" do
      pending "money needs to support more than :usd"
      account = Account.new
      account.balance.should == Money.new(0, :brl)
    end
  end
end
```

short circuit

```
Account
  when first created
    has a balance of zero reais (PENDING: money needs to support more than :usd)

Pending:

Account when first created has a balance of zero reais (money needs to support m
ore than :usd)
./pending.rb:12

Finished in 0.00152 seconds

1 example, 0 failures, 1 pending
```

short circuit

```
Account
  when first created
    has a balance of zero reais (PENDING: money needs to support more than :usd)

Pending:

Account when first created has a balance of zero reais (money needs to support m
ore than :usd)
./pending.rb:12

Finished in 0.00152 seconds

1 example, 0 failures, 1 pending
```

fail on success

fail on success

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais" do
      pending "money needs to support more than :usd" do
        account = Account.new
        account.balance.should == Money.new(0, :brl)
      end
    end
  end
end
```

fail on success

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais" do
      pending "money needs to support more than :usd" do
        account = Account.new
        account.balance.should == Money.new(0, :brl)
      end
    end
  end
end
```

fail on success

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais" do
      pending "money needs to support more than :usd" do
        account = Account.new
        account.balance.should == Money.new(0, :brl)
      end
    end
  end
end
```

with a block

while block raises

```
Account
  when first created
    has a balance of zero reais (PENDING: money needs to support more than :usd)

Pending:

Account when first created has a balance of zero reais (money needs to support m
ore than :usd)
./pending.rb:12

Finished in 0.00152 seconds

1 example, 0 failures, 1 pending
```

while block raises

```
Account
  when first created
    has a balance of zero reais (PENDING: money needs to support more than :usd)

Pending:

Account when first created has a balance of zero reais (money needs to support m
ore than :usd)
./pending.rb:12

Finished in 0.00152 seconds

1 example, 0 failures, 1 pending
```

when block does not raise

```
Account
  when first created
    has a balance of zero reais (FAILED - 1)

1)
'Account when first created has a balance of zero reais' FIXED
Expected pending 'money needs to support more than :usd' to fail. No Error was
raised.
./pending.rb:35:

Finished in 0.001907 seconds

1 example, 1 failure
```

when block does not raise

```
Account
  when first created
    has a balance of zero reais (FAILED - 1)

1)
'Account when first created has a balance of zero reais' FIXED
Expected pending 'money needs to support more than :usd' to fail. No Error was
raised.
./pending.rb:35:

Finished in 0.001907 seconds

1 example, 1 failure
```

nesting
describe/context

```ruby
describe AccountsController do
  describe "POST create" do
    context "with valid params" do
      it "creates an account"
      it "redirects to the index"
    end
  end
end
```

use describe for subject

```
describe AccountsController do
  describe "POST create" do
    context "with valid params" do
      it "creates an account"
      it "redirects to the index"
    end
  end
end
```

use describe for subject

```
describe AccountsController do
  describe "POST create" do
    context "with valid params" do
      it "creates an account"
      it "redirects to the index"
    end
  end
end
```

use context for ... context

```
AccountsController
  GET index
    assigns all accounts as @accounts
  GET show
    assigns the requested account as @account
  GET new
    assigns a new account as @account
  GET edit
    assigns the requested account as @account
  POST create
    with valid params
      assigns a newly created account as @account
      redirects to the created account
    with invalid params
      assigns a newly created but unsaved account as @account
      re-renders the 'new' template
```

```
AccountsController
  GET index
    assigns all accounts as @accounts
  GET show
    assigns the requested account as @account
  GET new
    assigns a new account as @account
  GET edit
    assigns the requested account as @account
  POST create
    with valid params
      assigns a newly created account as @account
      redirects to the created account
    with invalid params
      assigns a newly created but unsaved account as @account
      re-renders the 'new' template
```

subject

subject

```
AccountsController
  GET index
    assigns all accounts as @accounts
  GET show
    assigns the requested account as @account
  GET new
    assigns a new account as @account
  GET edit
    assigns the requested account as @account
  POST create
    with valid params
      assigns a newly created account as @account
      redirects to the created account
    with invalid params
      assigns a newly created but unsaved account as @account
      re-renders the 'new' template
```

context

context

good for documentation

bad for building up state

Author
with one article
and 37 followers
is popular
and 36 followers
is not popular

incremental state

```ruby
describe Author do
  before(:each) do
    @author = Author.new
  end

  context "with one article" do
    before(:each) do
      @author.articles << Article.new
    end

    context "and 37 followers" do
      before(:each) do
        37.times { @author.followers << Follower.new }
      end

      it "is popular" do
        @author.should be_popular
      end
    end
  end
end
```

```ruby
describe Author do
  before(:each) do
    @author = Author.new
  end

  context "with one article" do
    before(:each) do
      @author.articles << Article.new
    end

    context "and 37 followers" do
      before(:each) do
        37.times { @author.followers << Follower.new }
      end

      it "is popular" do
        @author.should be_popular
      end
    end
  end
end
```

```ruby
describe Author do
  before(:each) do
    @author = Author.new
  end

  context "with one article" do
    before(:each) do
      @author.articles << Article.new
    end

    context "and 37 followers" do
      before(:each) do
        37.times { @author.followers << Follower.new }
      end

      it "is popular" do
        @author.should be_popular
      end
    end
  end
end
```

```ruby
describe Author do
  before(:each) do
    @author = Author.new
  end

  context "with one article" do
    before(:each) do
      @author.articles << Article.new
    end

    context "and 37 followers" do
      before(:each) do
        37.times { @author.followers << Follower.new }
      end

      it "is popular" do
        @author.should be_popular
      end
    end
  end
end
```

```ruby
describe Author do
  before(:each) do
    @author = Author.new
  end

  context "with one article" do
    before(:each) do
      @author.articles << Article.new
    end

    context "and 37 followers" do
      before(:each) do
        37.times { @author.followers << Follower.new }
      end

      it "is popular" do
        @author.should be_popular
      end
    end
  end
end
```

alternatives

all in one place?

```ruby
describe Author do
  context "with one article" do
    context "and 37 followers" do
      it "is popular" do
        author = Author.new
        author.articles << Article.new
        37.times { author.followers << Follower.new }
        author.should be_popular
      end
    end


    context "and 36 followers" do
      it "is not popular" do
        author = Author.new
        author.articles << Article.new
        36.times { author.followers << Follower.new }
        author.should_not be_popular
      end
    end
  end
end
```

```ruby
describe Author do
  context "with one article" do
    context "and 37 followers" do
      it "is popular" do
        author = Author.new
        author.articles << Article.new
        37.times { author.followers << Follower.new }
        author.should be_popular
      end
    end


    context "and 36 followers" do
      it "is not popular" do
        author = Author.new
        author.articles << Article.new
        36.times { author.followers << Follower.new }
        author.should_not be_popular
      end
    end
  end
end
```

clarity

clarity

```ruby
describe Author do
  context "with one article" do
    context "and 37 followers" do
      it "is popular" do
        author = Author.new
        author.articles << Article.new
        37.times { author.followers << Follower.new }
        author.should be_popular
      end
    end
```

clarity

duplication

```ruby
    context "and 36 followers" do
      it "is not popular" do
        author = Author.new
        author.articles << Article.new
        36.times { author.followers << Follower.new }
        author.should_not be_popular
      end
    end
  end
end
```

clarity

middle ground?

```ruby
describe Author do
  context "with one article" do
    before(:each) do
      @author = Author.new
      @author.articles << Article.new
    end

    context "and 37 followers" do
      it "is popular" do
        37.times { @author.followers << Follower.new }
        @author.should be_popular
      end
    end
  end
end
```

```ruby
describe Author do
  context "with one article" do
    before(:each) do
      @author = Author.new
      @author.articles << Article.new
    end

    context "and 37 followers" do
      it "is popular" do
        37.times { @author.followers << Follower.new }
        @author.should be_popular
      end
    end
  end
end
```

balancing DRY and clarity is an art

clear over DRY

while we value the
items on the right

clear over DRY

while we value the
items on the right

clear over DRY

we value the items
on the left more

implicit subject

implicit subject

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais" do
      subject.balance.should == Money.new(0, :BLR)
    end
  end
end
```

implicit subject

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais" do
      subject.balance.should == Money.new(0, :BLR)
    end
  end
end
```

implicit subject

```
describe Account do
  context "when first created" do
    it "has a balance of zero reais" do
      subject.balance.should == Money.new(0, :BLR)
    end
  end
end
```

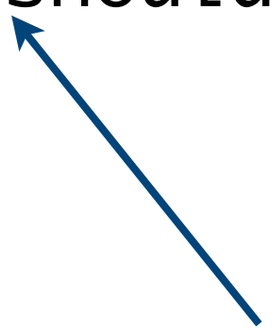
Account.new

implicit subject

```
describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
  end
end
```

implicit subject

```
describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
  end
end
```



who should
(who is the receiver)?

implicit subject

```
describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
  end
end
```

delegates to implicit subject

(Account.new)

explicit subject

explicit subject

```
describe Account do
  context "with 100 reais" do
    subject { Account.new(100, :BLR) }
    it "has a balance of 100 reais" do
      subject.balance.should == Money.new(100, :BLR)
    end
  end
end
```

explicit subject

```
describe Account do
  context "with 100 reais" do
    subject { Account.new(100, :BLR) }
    it "has a balance of 100 reais" do
      subject.balance.should == Money.new(100, :BLR)
    end
  end
end
```

explicit subject

```
describe Account do
  context "with 100 reais" do
    subject { Account.new(100, :BLR) }
    it { should have_a_balance_of(Money.new(100, :BLR))}
  end
end
```

explicit subject

```
describe Account do
  context "with 100 reais" do
    subject { Account.new(100, :BLR) }
    it { should have_a_balance_of(Money.new(100, :BLR))}
  end
end
```

delegates to explicit subject

subject is memoized

memoized

```
describe Account do
  context "with a zero balance" do
    it "accepts deposits" do
      subject.deposit(12, :USD)
      subject.should have_a_balance_of(12, :USD)
    end
  end
end
```

memoized

```
describe Account do
  context "with a zero balance" do
    it "accepts deposits" do
      subject.deposit(12, :USD)
      subject.should have_a_balance_of(12, :USD)
    end
  end
end
```

both reference the same object (Account.new)

let

```ruby
describe Account do
  context "with 10 BLR" do
    it "denies a withdrawal of 11 BLR" do
      account = Account.new(10, :BRL)
      expect { account.withdraw(11, :BLR) }.
        to raise_error(InsufficientFunds)
    end
  end
end
```

```ruby
describe Account do
  context "with 10 BLR" do
    it "denies a withdrawal of 11 BLR" do
      account = Account.new(10, :BRL)
      expect { account.withdraw(11, :BLR) }.
        to raise_error(InsufficientFunds)
    end

    it "allows a withdrawal of 10 BLR" do
      account = Account.new(10, :BRL)
      expect { account.withdraw(10, :BLR) }.
        to_not raise_error
    end
  end
end
```

```ruby
describe Account do
  context "with 10 BLR" do
    it "denies a withdrawal of 11 BLR" do
      account = Account.new(10, :BRL)
      expect { account.withdraw(11, :BLR) }.
        to raise_error(InsufficientFunds)
    end
```

duplication

```ruby
    it "allows a withdrawal of 10 BLR" do
      account = Account.new(10, :BRL)
      expect { account.withdraw(10, :BLR) }.
        to_not raise_error
    end
  end
end
```

```ruby
describe Account do
  context "with 10 BLR" do
    before(:each) { @account = Account.new(10, :BRL) }

    it "denies a withdrawal of 11 BLR" do
      expect { @account.withdraw(11, :BLR) }.
        to raise_error(InsufficientFunds)
    end

    it "allows a withdrawal of 10 BLR" do
      expect { @account.withdraw(10, :BLR) }.
        to_not raise_error
    end
  end
end
```

```ruby
describe Account do
  context "with 10 BLR" do
    before(:each) { @account = Account.new(10, :BRL) }

    it "denies a withdrawal of 11 BLR" do
      expect { @account.withdraw(11, :BLR) }.
        to raise_error(InsufficientFunds)
    end

    it "allows a withdrawal of 10 BLR" do
      expect { @account.withdraw(10, :BLR) }.
        to_not raise_error
    end
  end
end
```

```ruby
describe Account do
  context "with 10 BLR" do
    before(:each) { @account = Account.new(10, :BRL) }

    it "denies a withdrawal of 11 BLR" do
      expect { @account.withdraw(11, :BLR) }.
        to raise_error(InsufficientFunds)
    end

    it "allows a withdrawal of 10 BLR" do
      expect { @account.withdraw(10, :BLR) }.
        to_not raise_error
    end
  end
end
```

extract to before(:each)

1 addition
2 deletions
2 changes

```ruby
describe Account do
  context "with 10 BLR" do
    let(:account) { Account.new(10, :BRL) }
    it "denies a withdrawal of 11 BLR" do
      expect { account.withdraw(11, :BLR) }.
        to raise_error(InsufficientFunds)
    end

    it "allows a withdrawal of 10 BLR" do
      expect { account.withdraw(10, :BLR) }.
        to_not raise_error
    end
  end
end
```

```ruby
describe Account do
  context "with 10 BLR" do
    let(:account) { Account.new(10, :BRL) }
    it "denies a withdrawal of 11 BLR" do
      expect { account.withdraw(11, :BLR) }.
        to raise_error(InsufficientFunds)
    end

    it "allows a withdrawal of 10 BLR" do
      expect { account.withdraw(10, :BLR) }.
        to_not raise_error
    end
  end
end
```

extract to let()

1 addition
2 deletions
0 changes

custom matchers

matchers

```
describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
  end
end
```

matchers

```
describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
  end
end
```

matchers

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end


describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
  end
end
```

matchers

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end
```

match block returns boolean
true => pass
false => fail

```
describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
  end
end
```

should or should_not

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
    it { should_not have_a_balance_of(
      Money.new(1_000_0000, :BLR)
    )}
  end
end
```

should or should_not

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
    it { should_not have_a_balance_of(
      Money.new(1_000_0000, :BLR)
    )}
  end
end
```

should or should_not

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
    it { should_not have_a_balance_of(
      Money.new(1_000_0000, :BLR)
    )}
  end
end
```

should or should_not

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

describe Account do
  context "when first created" do
    it { should have_a_balance_of(Money.new(0, :BLR))}
    it { should_not have_a_balance_of(
      Money.new(1_000_0000, :BLR)
    )}
  end
end
```

```
with should
  true => pass
  false => fail
with should_not
  true => fail
  false => pass
```

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end
```

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should have_a_balance_of(Money.new(10, :BLR))
```

default failure messages

```ruby
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should have_a_balance_of(Money.new(10, :BLR))
```

```
expected Account (0 BLR) to have a balance of 10 BLR
```

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should have_a_balance_of(Money.new(10, :BLR))
```

```
expected Account (0 BLR) to have a balance of 10 BLR
```

account.inspect

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should have_a_balance_of(Money.new(10, :BLR))
```

```
expected Account (0 BLR) to have a balance of 10 BLR
```

account.inspect

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should have_a_balance_of(Money.new(10, :BLR))
```

```
expected Account (0 BLR) to have a balance of 10 BLR
```

account.inspect

money.inspect

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end
```

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should_not have_a_balance_of(Money.new(10, :BLR))
```

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should_not have_a_balance_of(Money.new(10, :BLR))
```

```
expected Account (10 BLR) not to have a balance of 10 BLR
```

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should_not have_a_balance_of(Money.new(10, :BLR))
```

```
expected Account (10 BLR) not to have a balance of 10 BLR
```

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should_not have_a_balance_of(Money.new(10, :BLR))
```

```
expected Account (10 BLR) not to have a balance of 10 BLR
```

default failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end
end

account = Account.new
account.should_not have_a_balance_of(Money.new(10, :BLR))
```

```
expected Account (10 BLR) not to have a balance of 10 BLR
```

override failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end

  failure_message_for_should do
    "expected Account to have #{money.inspect}"
  end

  failure_message_for_should_not do
    "expected Account not to have #{money.inspect}"
  end
end
```

override failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end

  failure_message_for_should do
    "expected Account to have #{money.inspect}"
  end

  failure_message_for_should_not do
    "expected Account not to have #{money.inspect}"
  end
end
```

override failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end

  failure_message_for_should do
    "expected Account to have #{money.inspect}"
  end

  failure_message_for_should_not do
    "expected Account not to have #{money.inspect}"
  end
end
```

override failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end

  failure_message_for_should do
    "expected Account to have #{money.inspect}"
  end

  failure_message_for_should_not do
    "expected Account not to have #{money.inspect}"
  end
end
```

```
expected Account to have a balance of 0 BLR
```

override failure messages

```
Spec::Matchers.define :have_a_balance_of do |money|
  match do |account|
    account.balance == money
  end

  failure_message_for_should do
    "expected Account to have #{money.inspect}"
  end

  failure_message_for_should_not do
    "expected Account not to have #{money.inspect}"
  end
end
```

```
expected Account to have a balance of 0 BLR

expected Account not to have a balance of 1000000 BLR
```

fluent matchers

```
describe Card do
  describe "Ace of spades" do
    it "is the Ace of spades" do
      card = Card.new('Ace', :spades)
      card.should be_the('Ace').of(:spades)
    end
  end
end
```

fluent matchers

```
describe Card do
  describe "Ace of spades" do
    it "is the Ace of spades" do
      card = Card.new('Ace', :spades)
      card.should be_the('Ace').of(:spades)
    end
  end
end
```

fluent matchers

```
Spec::Matchers.define :be_the do |rank|
  match do |card|
    card.rank == rank && card.suit == @suit
  end

  def of(suit)
    @suit = suit
    self
  end
end
```

fluent matchers

```
Spec::Matchers.define :be_the do |rank|
  match do |card|
    card.rank == rank && card.suit == @suit
  end

  def of(suit)
    @suit = suit
    self
  end
end
```

fluent matchers

```
Spec::Matchers.define :be_the do |rank|
  match do |card|
    card.rank == rank && card.suit == @suit
  end

  def of(suit)
    @suit = suit
    self
  end
end
```

be sure to return self

mock argument matchers

```
describe "glass" do
  context "when empty" do
  end
end
```

mock argument matchers

```
describe "glass" do
  context "when empty" do
    it "should be filled with cachaça" do
      glass = Glass.new
      glass.should_receive(:fill).with(cachaça)
      glass.fill(Cachaça.new)
    end
  end
end
```

mock argument matchers

```
describe "glass" do
  context "when empty" do
    it "should be filled with cachaça" do
      glass = Glass.new
      glass.should_receive(:fill).with(cachaça)
      glass.fill(Cachaça.new)
    end
  end
end
```

mock argument matchers

```
Spec::Matchers.define :cachaça do
  match do |actual|
    Cachaça === actual
  end
end

describe "glass" do
  context "when empty" do
    it "should be filled with cachaça" do
      glass = Glass.new
      glass.should_receive(:fill).with(cachaça)
      glass.fill(Cachaça.new)
    end
  end
end
```

wrapped assertions
(currently in dev)

```
describe AccountsHelper do
  describe "#link_to_account" do
    it "renders a link to the account" do
      account = mock_model(Account, :id => "37", :name => "David")
      assert_dom_equal %Q{<a href="/accounts/37">David's account</a>},
        helper.link_to_account(account)
    end
  end
end
```

wrapped assertions
(currently in dev)

```
describe AccountsHelper do
  describe "#link_to_account" do
    it "renders a link to the account" do
      account = mock_model(Account, :id => "37", :name => "David")
      assert_dom_equal %Q{<a href="/accounts/37">David's account</a>},
        helper.link_to_account(account)
    end
  end
end
```

wrapped assertions
(currently in dev)

```
describe AccountsHelper do
  describe "#link_to_account" do
    it "renders a link to the account" do
      account = mock_model(Account, :id => "37", :name => "David")
      helper.link_to_account(account).
        should match_dom(%Q{<a href="/accounts/37">David's account</a>})
    end
  end
end
```

wrapped assertions (currently in dev)

```
Spec::Matchers.define :match_dom do |expected|
  extend Test::Unit::Assertions
  extend ActionController::TestCase::Assertions
  match do |actual|
    wrapped_assertion do
      assert_dom_equal expected, actual
    end
  end
end
```

wrapped assertions
(currently in dev)

```
Spec::Matchers.define :match_dom do |expected|
  extend Test::Unit::Assertions
  extend ActionController::TestCase::Assertions
  match do |actual|
    wrapped_assertion do
      assert_dom_equal expected, actual
    end
  end
end
```

extend the matcher to make the assertion available

wrapped assertions
(currently in dev)

```
Spec::Matchers.define :match_dom do |expected|
  extend Test::Unit::Assertions
  extend ActionController::TestCase::Assertions
  match do |actual|
    wrapped_assertion do
      assert_dom_equal expected, actual
    end
  end
end
```

extend the matcher to make the assertion available

returns false if wrapped_assertion block raises, otherwise returns true

wrapped assertions
(currently in dev)

```
describe AccountsHelper do
  describe "#link_to_account" do
    it "renders a link to the account" do
      account = mock_model(Account, :id => "37", :name => "David")
      helper.link_to_account(account).
        should_not match_dom(%Q{<a href="/accounts/37">David's account</a>})
    end
  end
end
```

wrapped assertions
(currently in dev)

```
describe AccountsHelper do
  describe "#link_to_account" do
    it "renders a link to the account" do
      account = mock_model(Account, :id => "37", :name => "David")
      helper.link_to_account(account).
        should_not match_dom(%Q{<a href="/accounts/37">David's account</a>})
    end
  end
end
```

wrapped assertions provide positive and negative matcher semantics from one matcher + assertion

spork + autotest

autotest monitors changes to files and re-runs your tests

spork loads the infrastructure once and forks a new process for each test run

```
$ rails my_project
$ cd my_project
$ gem install rspec-rails
$ gem install ZenTest
$ gem install spork
$ script/generate rspec
$ echo --drb >> spec/spec.opts
$ spork --bootstrap
# follow directions in spec/spec_helper.rb
```

```ruby
require 'rubygems'
require 'spork'

Spork.prefork do
  ENV["RAILS_ENV"] ||= 'test'
  require File.expand_path('../../config/environment', __FILE__)
  require 'spec/autorun'
  require 'spec/rails'

  require 'webrat/integrations/rspec-rails'

  Dir[File.expand_path('../support/**/*.rb', __FILE__)].each {|f| require f}

  Spec::Runner.configure do |config|
    config.use_transactional_fixtures = true
    config.use_instantiated_fixtures  = false
    config.fixture_path = RAILS_ROOT + '/spec/fixtures/'
  end
end

Spork.each_run do
end
```

```ruby
require 'rubygems'
require 'spork'

Spork.prefork do
  ENV["RAILS_ENV"] ||= 'test'
  require File.expand_path('../../config/environment', __FILE__)
  require 'spec/autorun'
  require 'spec/rails'

  require 'webrat/integrations/rspec-rails'

  Dir[File.expand_path('../support/**/*.rb', __FILE__)].each {|f| require f}

  Spec::Runner.configure do |config|
    config.use_transactional_fixtures = true
    config.use_instantiated_fixtures  = false
    config.fixture_path = RAILS_ROOT + '/spec/fixtures/'
  end
end

Spork.each_run do
end
```

wraps code that should run once

```ruby
require 'rubygems'
require 'spork'

Spork.prefork do
  ENV["RAILS_ENV"] ||= 'test'
  require File.expand_path('../../config/environment', __FILE__)
  require 'spec/autorun'
  require 'spec/rails'

  require 'webrat/integrations/rspec-rails'

  Dir[File.expand_path('../support/**/*.rb', __FILE__)].each {|f| require f}

  Spec::Runner.configure do |config|
    config.use_transactional_fixtures = true
    config.use_instantiated_fixtures  = false
    config.fixture_path = RAILS_ROOT + '/spec/fixtures/'
  end
end

Spork.each_run do
end
```

wraps code that should run once

wraps code that should run each time

```
$ cd my_project
$ spork
```

two separate shells

```
$ cd my_project
$ autospec
```

cucumber

transforms

```ruby
When /^I increase the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end

When /^I decrease the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move down"
  end
end
```

```
When /^I increase the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end

When /^I decrease the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move down"
  end
end
```

```ruby
When /^I increase the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end

When /^I decrease the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move down"
  end
end
```

```ruby
When /^I increase the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end

When /^I decrease the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move down"
  end
end
```

```ruby
When /^I increase the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end

When /^I decrease the priority of the "([^\"]*)" card$/ do
    |title|
  visit iterations_path
  card = Card.find_by_title(title)
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move down"
  end
end
```

duplication

```ruby
Transform /^the "([^\"]*)" card$/ do |title|
  Card.find_by_title(title)
end

When /^I increase the priority of (the .* card)$/ do
    |card|
  visit iterations_path
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end
```

```ruby
Transform /^the "([^\"]*)" card$/ do |title|
  Card.find_by_title(title)
end

When /^I increase the priority of (the .* card)$/ do
    |card|
  visit iterations_path
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end
```

```ruby
Transform /^the "([^\"]*)" card$/ do |title|
  Card.find_by_title(title)
end

When /^I increase the priority of (the .* card)$/ do
    |card|
  visit iterations_path
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end
```

```ruby
Transform /^the "([^\"]*)" card$/ do |title|
  Card.find_by_title(title)
end

When /^I increase the priority of (the .* card)$/ do
    |card|
  visit iterations_path
  within("#card_#{card.id}") do |scope|
    scope.click_button "Move up"
  end
end
```

```ruby
Given /^the "([^\"]*)" card has higher priority than the "([^\"]*)" card$/ do
    |first_title, second_title|
  first_card = Card.find_by_title(first_title)
  second_card = Card.find_by_title(second_title)
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

```ruby
Given /^the "([^\"]*)" card has higher priority than the "([^\"]*)" card$/ do
    |first_title, second_title|
  first_card = Card.find_by_title(first_title)
  second_card = Card.find_by_title(second_title)
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

```ruby
Transform /^the "([^\"]*)" card$/ do |title|
  Card.find_by_title(title)
end

Given /^(the .* card) has higher priority than (the .* card)$/ do
    |first_card, second_card|
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

```ruby
Given /^the "([^\"]*)" card has higher priority than the "([^\"]*)" card$/ do
    |first_title, second_title|
  first_card = Card.find_by_title(first_title)
  second_card = Card.find_by_title(second_title)
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

```ruby
Transform /^the "([^\"]*)" card$/ do |title|
  Card.find_by_title(title)
end

Given /^(the .* card) has higher priority than (the .* card)$/ do
    |first_card, second_card|
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

```ruby
Given /^the "([^\"]*)" card has higher priority than the "([^\"]*)" card$/ do
    |first_title, second_title|
  first_card = Card.find_by_title(first_title)
  second_card = Card.find_by_title(second_title)
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

```ruby
Transform /^the "([^\"]*)" card$/ do |title|
  Card.find_by_title(title)
end

Given /^(the .* card) has higher priority than (the .* card)$/ do
    |first_card, second_card|
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

```ruby
Given /^the "([^\"]*)" card has higher priority than the "([^\"]*)" card$/ do
    |first_title, second_title|
  first_card = Card.find_by_title(first_title)
  second_card = Card.find_by_title(second_title)
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

```ruby
Transform /^the "([^\"]*)" card$/ do |title|
  Card.find_by_title(title)
end

Given /^(the .* card) has higher priority than (the .* card)$/ do
    |first_card, second_card|
  first_card.update_attributes!(:priority => 1) unless first_card.priority
  second_card.update_attributes!(:priority => first_card.priority + 1)
end
```

perguntas?

http://blog.davidchelimsky.net/

http://rspec.info/

http://cukes.info/

http://www.pragprog.com/titles/achbd/the-rspec-book

describe it with rspec

Cucumber

Behaviour Driven Development
with elegance and joy