

## Lezione 6: Testing, HTML forms

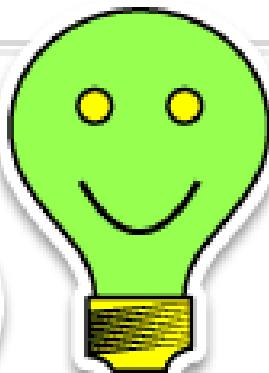
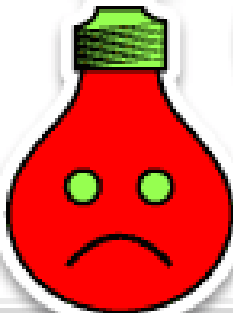
Matteo Vaccari

<http://matteo.vaccari.name/>  
matteo.vaccari@uninsubria.it



(cc) Matteo Vaccari. Published in Italy.  
Attribution – Non commercial – Share alike 2.5

**Debugging  
sucks.**



**Testing rocks.**

# Benvenuti nel XXI secolo

Testare è assolutamente essenziale

Debuggare fa tanto ventesimo secolo

Scrivere test è divertente

# Clean code that works.

Is out of reach of even the best programmers, some of the time,  
and out of reach of most programmers (like me) most of the time

— Kent Beck

## Come li vogliamo i test?

- ▶ Automatici
- ▶ Ripetibili
- ▶ Isolati
- ▶ Self-checking

## Tipi di test in Rails

- ▶ *unit* tests: modelli
- ▶ *functional* tests: controller (e indirettamente i modelli)
- ▶ *integration* tests: più di un modello
- ▶ *performance* tests

# Unit test

my\_test.rb

---

```
require "test/unit"
```

```
class MyTest < Test::Unit::TestCase
  def test_sum
    assert_equal 4, 2+2
  end
end
```

```
$ ruby my_test.rb
```

```
Started
```

```
.
```

```
Finished in 0.000366 seconds.
```

```
1 tests, 1 assertions, 0 failures, 0 errors
```

```
$
```

## Tre ambienti

A seconda della variabile RAILS\_ENV:

<b>environment</b>	<b>database</b>	<b>configuration</b>
DEVELOPMENT	quotes_development	config/environments/development.rb
TEST	quotes_test	config/environments/test.rb
PRODUCTION	quotes_production	config/environments/production.rb

```
$ rake test
```

- ▶ Copia la struttura del db di sviluppo nel db di test
- ▶ Esegue tutti i test

# Che cosa testare?

Fare test **positivi** e **negativi**

---

```
def test_find_by_author
  Quote.delete_all
  quote = Quote.create(:body => 'bla', :author => 'Pippo')

  # Test positivo: lo troviamo Pippo?
  assert_equal quote, Quote.find_by_author('Pippo')

  # Test negativo: che succede se cerchiamo un autore che non c'è?
  assert_equal nil, Quote.find_by_author('foobar')
end
```

## Assertion methods

```
assert(User.find_by_name("dave"), "user 'dave' is missing")
```

```
assert_equal(3, Product.count)
```

```
assert_not_equal(0, User.count, "no users in database")
```

```
assert_nil(User.find_by_name("willard"))
```

```
assert_not_nil(User.find_by_name("henry"))
```

```
assert_raise(ActiveRecord::RecordNotFound) do
```

```
  Product.find(bad_id)
```

```
end
```

```
assert_difference('Post.count', 1) do
```

```
  Quote.create(:title => "foo", :author => "bar")
```

```
end
```

## Testing of controllers

test/functional/quotes\_controller\_test.rb

---

```
require 'test_helper'
```

```
class QuotesControllerTest < ActionController::TestCase
  def test_index
    get :index
    assert_response :success
    assert_template 'list'
    assert_equal 10, assigns(:quotes).size
  end
end
```

app/controllers/quotes\_controller.rb

---

```
class QuotesController < ApplicationController
  def index
    @quotes = Quote.find :all, :limit => 10
    render :template => "list"
  end
end
```

## Testare le view

```
assert_select "title", "Pragprog Books Online Store"
assert_select "title", /Online/

assert_select "div#cart"
assert_select "div#cart table tr", :count => 3
assert_select "div#cart table tr.total-line td:last-of-type", "$57.70"

assert_select "div#cart" do
  assert_select "table" do
    assert_select "tr", :count => 3
    assert_select "tr.total-line td:last-of-type", "$57.70"
  end
end
```

# Strategie per iniziare a testare

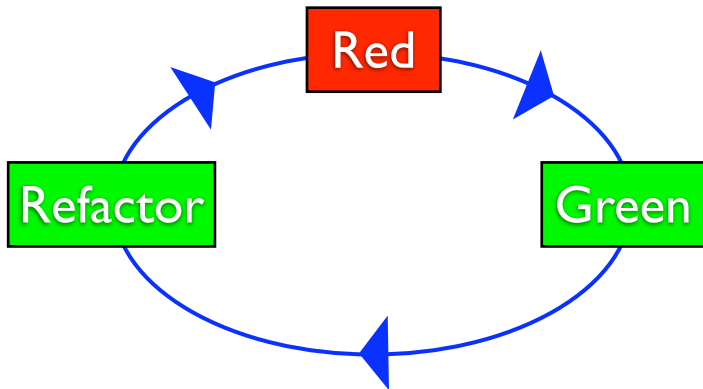
- ▶ test di integrazione
- ▶ test-driven development

## Test di integrazione

```
def assert_xml(contents)
  REXML::Document.new(contents)
end
```

```
def test_index
  get '/quotes'
  assert_response :success
  assert_xml @response.body
end
```

# Test-Driven Development



## Esercizio

Testare che il servizio realizzato nella scorsa lezione funzioni correttamente

1. `http://example.com/`  
ridirige su `http://example.com/quotes?random`
2. `http://example.com/quotes?random`  
produce una citazione a caso
3. `http://example.com/quotes`  
produce le prime 10 citazioni
4. `http://example.com/quotes/show/123`  
produce la citazione numero 123
5. `http://example.com/quotes?count=4`  
produce le prime 4 citazioni
6. `http://example.com/quotes?start=12`  
produce 10 citazioni a partire dalla dodicesima

# Html Forms

Html form: markup, content + **controls**

```
<form action='login' method='post'>  
  Login:   <input type='text'      name='login'      value='' />  
  <br />  
  Password: <input type='password' name='password' value='' />  
  <br />  
           <input type='submit'    value='Dite amici ed entrate' />  
</form>
```



Login: amici

Password: \*\*\*

Dite amici ed entrate

## Un esempio di controllo

```
<input type='text' name='login' value='' />
```

`type` vari tipi di controlli: text, password, hidden...

`name` identifica il controllo

`value` il valore *iniziale*

Ogni controllo ha un valore *iniziale* e uno *corrente*

## Control types: buttons

### Tre tipi di bottoni

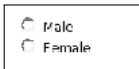
- ▶ submit buttons: submits a form
  - ▶ viene invocata la *action* della form
  - ▶ si carica una nuova pagina
- ▶ reset buttons: resets all controls to their initial values.
- ▶ push buttons: no default behavior; => Client-side scripts

```
<input type='submit' value='Dite amici ed entrate' />  
<input type='reset' value='Ricomincia da capo' />  
<button value='cliccami' onclick='alert("zot!")' />
```

## Checkboxes, radio buttons

### Radio buttons: alternative

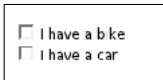
```
<form>  
<input type='radio' name='sex' value='male' /> Male <br />  
<input type='radio' name='sex' value='female' /> Female  
</form>
```



A rectangular box containing two radio button options. The first option is a radio button followed by the text "Male". The second option is a radio button followed by the text "Female".

### Check boxes: opzioni

```
<form>  
<input type='checkbox' name='bike' /> I have a bike <br />  
<input type='checkbox' name='car' /> I have a car  
</form>
```



A rectangular box containing two checkbox options. The first option is a checkbox followed by the text "I have a bike". The second option is a checkbox followed by the text "I have a car".

## Menu a discesa

```
<select name='cars'>  
  <option value='volvo' checked='checked'>Volvo</option>  
  <option value='saab'>Saab</option>  
  <option value='fiat'>Fiat</option>  
  <option value='audi'>Audi</option>  
</select>
```

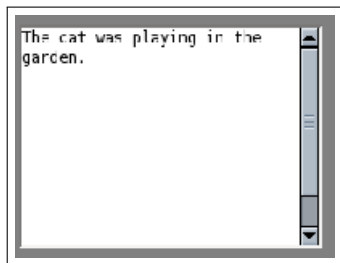


## Editor di testo

Singola riga: `<input type='text' ... />`

Più righe:

```
<textarea rows='10' cols='30'>  
The cat was playing in the garden.  
</textarea>
```



## File select

Permette di passare file all'applicazione

```
<input type='file' name='f' />
```

## Campi invisibili

```
<input name='invisibile'  
      type='hidden'  
      value='un valore nascosto!' />
```

## Cosa succede quando premo il bottone?

```
<form action='autentica' method='post'>  
  ...  
  <input type='submit' value='Dite amici ed entrate'>  
</form>
```

input type submit: crea un bottone che manda i dati... dove?

action: url che indica dove verranno spediti i dati (default: la url della form stessa)

method: il metodo HTTP con cui li spediremo (default: GET)

## Due *metodi*: “get” e “post”

Con “**get**”:

- ▶ i dati sono visibili sulla url  
/pippo/autentica?login=amici&password=xyz
- ▶ non posso mandare grandi quantità di dati

Con “**post**”:

- ▶ i dati non sono visibili sulla url  
(ma sono lo stesso visibili con packet sniffing!)
- ▶ non c'è limite alla quantità di dati
- ▶ non posso fare un bookmark del risultato

## “get” e “post”, cont.

Semanticamente:

GET: operazione *idempotente*

(cerco un libro in un catalogo, ottengo sempre la stessa risposta)

POST: operazione *NON idempotente*

(ordino un libro; se eseguo la query 10 volte mando 10 ordini)

## Aggiungiamo una form

- ▶ prima in html puro
- ▶ poi con gli **helpers**
- ▶ aggiungiamo un **action** per gestire l'aggiunta di una citazione

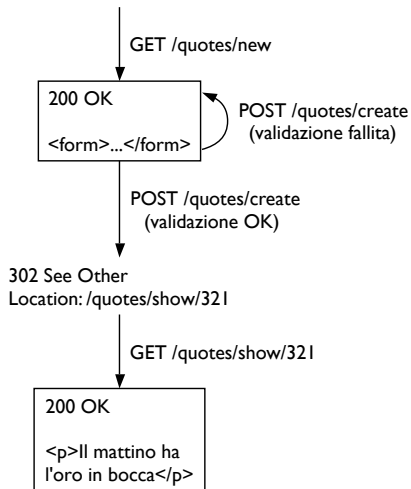
## Schema get-post-redirect

`/quotes/new`  $\xrightarrow{\text{GET}}$  200 OK  
<html>...</html>

`/quotes/create`  $\xrightarrow{\text{POST}}$  302 See Other  
Location: `/quotes/show/321`

`/quotes/show/321`  $\xrightarrow{\text{GET}}$  200 OK  
<html>...</html>

# Form editing



## app/controllers/quote\_controller.rb

---

```
class QuotesController < ApplicationController
  ...
  def create
    quote = Quote.new(:author => params[:author],
                     :body   => params[:body])

    quote.save
    redirect_to :action => :show, :id => quote.id
  end
end
```

## app/views/quote/new.html.erb

---

```
<form action='/quotes/create' method='post'>
  Autore: <input type='text' name='author' value='' /><br />
  Testo: <br />
  <textarea name='body' rows='8' cols='40'></textarea><br />
  <input type='submit' value='Inserisci' />
</form>
```

## L'azione "show"

app/controllers/quote\_controller.rb

---

```
class QuotesController < ApplicationController
  def show
    @quote = Quote.find(params[:id])
  end
end
```

app/views/quote/show.html.erb

---

```
<h2>Il motto con id <%= @quote.id %></h2>
<%= render :partial => 'quote' %>
```

## Form helpers

Si può scrivere la form in HTML

---

```
<form action='/quote/create' method='post'>
  Autore: <br />
  <input type='text' name='author' value='' /><br />
  Testo: <br />
  <textarea name='body' rows='8' cols='40'></textarea><br />
  <input type='submit' value='Inserisci' />
</form>
```

Ma è meglio usare gli helper

---

```
<% form_tag :action => :new do %>
Autore: <br />
<%= text_field_tag :quote, :author %><br />
Testo: <br />
<%= text_area_tag :quote, :body %><br />
<%= submit_tag 'Inserisci' %>
<% end %>
```

## app/views/quote/new.html.erb

---

```
<% form_tag :action => :new do %>
Autore: <br />
<%= text_field_tag :quote, :author %><br />
Testo: <br />
<%= text_area_tag :quote, :body %><br />
<%= submit_tag 'Inserisci' %>
<% end %>
```

produce      ↓

```
<form action="/quote/create" method="post">
Autore: <br />
<input id="quote_author" name="quote[author]" size="30" type="text" /><br />
Testo: <br />
<textarea cols="40" id="quote_body" name="quote[body]" rows="20"></textarea>
<input name="commit" type="submit" value="Inserisci" />
</form>
```