Integration Testing in Ruby with RSpec's Story Automation Framework

David Chelimsky
articulatedman.com

Behaviour-Driven Development

BDD

- Dan North/Aslak Hellesøy 2004
  - Improve communication about Test Driven Development
  - JBehave

BDD

- Second generation "full stack" Agile methodology rooted in:
  - Extreme Programming
  - Acceptance Test Driven Planning
  - Test Driven Development

RSpec

RSpec

- Behaviour Driven Development Framework
  - Story Framework
    - Acceptance Test Driven Planning
  - Example Framework
    - Test Driven Development

RSpec Origins

- Inspired by a blog post by Dave Astels
- Authored by Steven Baker
    - Summer '05
- Maintained by me
    - Since Summer '06

http://daveastels.com/2005/07/05/a-new-look-at-test-driven-development/

BDD Process

Process

- Inject features discovered through analysis
  - Feature Injection - Chris Matts
- Extract stories from features
  - Focus on outputs
- Break stories down into scenarios
  - Acceptance Criteria

Feature Injection

Popping the “Why?” Stack

Popping the “Why?” Stack

- I want people to be able to register
  - Why?
- I want to know how many people are registered
  - Why?
- So I can measure progress towards registration goals
  - Why?

Popping the “Why?” Stack

- This is really annoying
  - I know. Why do you want to measure progress towards registration goals?

Popping the “Why?” Stack

- This is really annoying
  - I know. Why do you want to measure progress towards registration goals?
- SO THAT I CAN MANAGE COST!

Popping the "Why?" Stack

- If you keep asking "why?", you'll eventually land on one of:
  - Generate/protect revenue
  - Reduce/manage cost
- When you do, the answer to the previous "why?" is often a feature waiting to be discovered.

Focus on Outputs

- Business value lies in what you get out of the system, not what you put into it.
    - Reports
    - Messages

User Stories

- High level analysis and planning tool
- “Token for a conversation”

So what does this all have to do with Integration Testing?

Integration Testing

- Goals
  - Make sure the component parts play nice together
  - Document the expected behaviour of the system

BDD Stories

- The next step in the process
- Collections of automated scenarios

Conference Organizer (Example Application)

Example Story

Example Story

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

Title

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

Narrative

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

Narrative Format

“The Connextra Format”

As a **Role**

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

I want **Action**

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

So that **Goal**

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

Narrative Format

- Completely arbitrary, but ...
  - Look for a format that
    - Identifies the goal
    - Identifies the user/persona

Scenario Format

Given | When | Then

The other GWT

Given | When | Then

- A simple way of saying:
  - Pre-conditions, Event, Post-conditions
  - Context, Action, Outcome
  - Build, Operate, Check
    - Uncle Bob Martin

Given | When | Then

- Words that can be understood equally well by:
  - stakeholders
  - business analysts
  - developers
  - testers

Given

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

When

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

Then

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

Automation

Ruby

```ruby
Story "measure progress towards registration goals",%(
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals
), :type => RailsStory, :steps_for => :registrations do
  Scenario "one registration shows as 1%" do
    Given "a goal of 200 registrations"
    When "1 attendee registers"
    Then "the goal should be 1% achieved"
  end
  Scenario "one registration less than the goal shows as 99%" do
    Given "a goal of 200 registrations"
    When "199 attendees register"
    Then "the goal should be 99% achieved"
  end
end
```

Plain Text

```
with_steps_for :registrations do
  run "#{File.dirname(__FILE__)}/measure_progress.story",
      :type => RailsStory
end
```

Step Definitions

Direct Model Access

Given

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

Given

```
steps_for :registrations do
  Given "a goal of $goal registrations" do |goal|
    @conference = Conference.create!(
      :name => "BDD", :goal => goal
    )
  end
end
```

When

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

When

```
steps_for :registrations do
  ...
  When /(\d+) attendee(s?) register(s?)/ do |count,_,_|
    (1..(count.to_i)).each do |n|
      Registration.create!(
        :name => "Name #{n}",
        :email => "email#{n}@site.com",
        :conference => @conference
      )
    end
  end
end
```

Then

```
Story: measure progress towards registration goals
  As a conference organizer
  I want to see a report of registrations
  So that I can measure progress towards registration goals

  Scenario: one registration shows as 1%
    Given a goal of 200 registrations
    When 1 attendee registers
    Then the goal should be 1% achieved

  Scenario: one registration less than the goal shows as 99%
    Given a goal of 200 registrations
    When 199 attendees register
    Then the goal should be 99% achieved
```

Then

```
steps_for :registrations do
  ...
  Then "the goal should be $percentage% achieved" do
    |percentage|
    @conference.percentage_of_goal.should == percentage.to_i
  end
end
```

Direct Model Access

- Pros
  - More flexible
  - Less brittle
- Cons
  - Less thorough
    - No views/controllers

(Almost) Full Stack using Rails Integration Test and Webrat

Rails Integration Test

- Simulate HTTP requests
  - Goes through routing (unlike Rails functional tests)
- Simulate multiple sessions

RailsStory

- Wraps Rails Integration Test
    - Access to everything you get from Rails
    - Access to everything you get from RSpec

Given

```
steps_for :registrations_through_ui do
  Given "a goal of $goal registrations" do |goal|
    get "/conferences/new"
    response.should have_tag(
      "form[action=?]", conferences_path) do
      with_tag("input#conference_name")
      with_tag("input#conference_goal")
    end
    @conference_name =  "BDD #{Time.new.to_i}"
    post "/conferences", :conference => {
      :name => @conference_name, :goal => goal
    }
  end
end
```

Duplication

```
steps_for :registrations_through_ui do
  Given "a goal of $goal registrations" do |goal|
    get "/conferences/new"
    response.should have_tag(
      "form[action=?]", conferences_path) do
      with_tag("input#conference_name")
      with_tag("input#conference_goal")
    end
    @conference_name =  "BDD #{Time.new.to_i}"
    post "/conferences", :conference => {
      :name => @conference_name, :goal => goal
    }
  end
end
```

Webrat

- Ruby Gem written by Bryan Helmkamp
  - http://github.com/brynary/webrat
- stores DOM in memory
- manipulates DOM
- builds POST from DOM
  - logically binding the form to the POST

Given (with Webrat)

```
steps_for :registrations_through_ui do
  Given "a goal of $goal registrations" do |goal|
    @conference_name =  "BDD #{Time.new.to_i}"
    visits "/conferences/new"
    fills_in "Name", :with => @conference_name
    fills_in "Goal", :with => goal
    clicks_button "Goal"
  end
end
```

When

```
steps_for :registrations_through_ui do
  ...
  When /(\d+) attendee(s?) register(s?)/ do |count,_,_|
    (1..(count.to_i)).each do |n|
      visits "/registrations/new"
      fills_in "Name", :with => "Name #{n}"
      fills_in "E-Mail", :with => "email#{n}@site.com"
      selects @conference_name
      clicks_button
    end
  end
end
```

Then

```
steps_for :registrations_through_ui do
  ...
  Then "the goal should be $percentage% achieved" do
    |percentage|
    @conference = Conference.find_by_name(@conference_name)
    visits "/conferences/#{@conference.id}"
    response.should have_text(/#{percentage}%/)
  end
end
```

Full Stack (sans browser)

- Pros
  - Full stack
    - High level of coverage
    - Confidence

Full Stack (sans browser)

- Cons
  - Full Stack
    - Subject to changes from larger area
  - Requires known html elements/structure
    - Not too bad if you follow conventions
    - Webrat helps too

Full Stack using Selenium-RC

Given

```ruby
steps_for :registrations_through_browser do
  Given "a goal of $goal registrations" do |goal|
    $browser.open "http://localhost:3000/conferences/new"
    @conference_name =  "BDD #{Time.new.to_i}"
    $browser.type "conference_name", @conference_name
    $browser.type "conference_goal", goal
    $browser.submit "new_conference"
    $browser.wait_for_page_to_load(5000)
    @conference_url = $browser.get_location
  end
end
```

When

```
steps_for :registrations_through_browser do
  ...
  When /(\d+) attendee(s?) register(s?)/ do |count,_,_|
    (1..(count.to_i)).each do |n|
      $browser.open "http://localhost:3000/registrations/new"
      $browser.type "css=#registration_name", "Name #{n}"
      $browser.type "css=#registration_email",
                    "email#{n}@site.com"
      $browser.select "css=#registration_conference_id",
                      @conference_name
      $browser.submit "css=#new_registration"
    end
  end
end
```

Then

```
steps_for :registrations_through_browser do
  ...
  Then "the goal should be $percentage% achieved" do
    |percentage|
    $browser.open @conference_url
    $browser.get_text("css=#percentage_of_goal").
             should =~ /#{percentage}%/
  end
end
```

Full Stack (with browser)

- Pros
  - Test javascript/ajax
    - Test some aspects of HTML too
  - You can watch it!
    - High impact for customers

Full Stack (with browser)

- Cons
  - Brittle
    - Like in-memory, coupled to entire stack
    - Possibly even more subject to UI changes
  - S L O W

Full Stack (with browser)

- Recommendation
  - Use in-memory first
  - Use in-browser when
    - Testing javascript/ajax
    - Useful when necessary to increase customer confidence

Detailed Scenarios

Detailed Scenarios

```
Story: attendee registers
  As a potential attendee
  I want to register for a conference
  So that I may attend and learn great stuff

  Scenario: successful registration
    Given I am viewing the registration form
    When I enter Name: Joe Smith
    And I enter E-Mail: jsmith@site.com
    And I check Tutorials
    And I click Register
    Then I should see the Registration Confirmation
    And it should show Name: Joe Smith
    And it should show E-Mail: jsmith@site.com
    And it should show Tutorials: Yes
```

Detailed Scenarios

```
...

Scenario: missing email address
  Given I am viewing the registration form
  When I enter Name: Joe Smith
  And I do not enter E-Mail
  And I click Register
  Then I should see the Registration Form
  And it should show Email is required
```

Givens

```
steps_for :registration do
  Given "I am viewing the registration form" do
    visits new_registration_path
  end

  Given "a conference named $name" do |name|
    visits new_conference_path
    fills_in "Name", :with => name
    fills_in "Goal", :with => 200
    clicks_button
  end
end
```

Whens

```
steps_for :registration do
  When "I enter $label: $value" do |label, value|
    fills_in label, :with => value
  end

  When "I do not enter $label" do |label|
    # no-op - doc purposes only
  end
end
```

Whens

```
steps_for :registration do
  When "I select $label" do |label|
    selects label
  end

  When "I check $label" do |label|
    checks label
  end

  When "I click $button" do |button|
    clicks_button button
  end
end
```

Thens

```
steps_for :registration do
  Then /I should see the Registration (Form|Confirmation)/ do
    |form_or_confirmation|
    case form_or_confirmation
    when "Form"
      response.should render_template("registrations/new")
    when "Confirmation"
      r = Registration.find(:all, :order => 'id').last
      response.should render_template("registrations/show")
    end
  end

  Then "it should show $text" do |text|
    response.should include_text(text)
  end
end
```

Detailed Scenarios

- More "design up front" feel vs "story as token for conversation"
- Scenarios are *more* subject to changes from customer
- Steps are more granular
    - Easier to write
    - Easier to change

Multiple Sessions

Multiple Sessions

```
Story "can not view other group's calendar", %(
  As a member of one group with calendars
  I do not want members of other groups to see my calendars
  So that my data is kept private
), :type => RailsStory, :steps_for => [
   :users_and_groups, :calendars, :navigation
] do

...
end
```

Multiple Sessions

```
Story "can not view other group's calendar", %(
  ...
  Scenario "two users, two groups, two calendars" do
    Given "a group named Group1"
    And "a group named Group2"
    And "Group1 has a calendar named Calendar1"
    And "Group2 has a calendar named Calendar2"
    And "a Group1 member named Person1 is logged in"
    And "a Group2 member named Person2 is logged in"

    When "Person1 visits the calendar list"
    Then "he should see Calendar1"
    And "he should not see Calendar2"

    When "Person2 visits the calendar list"
    Then "she should see Calendar2"
    And "she should not see Calendar1"
  end
end
```

Given

```
Given "a group named $group" do |group|
  set_ivar :group, group, Group.create!(:name => group)
end

Given "a $group member named $name is logged in" do |group, name|
  create_user_named(name) do |user|
    user.activate
    user.groups << get_ivar(:group, group)
  end
  login_as(name)
end
```

When

```
When "$person visits the $page" do |person, page|
  page_map = {
    "calendar list" => "/calendars"
  }
  @current_session = get_ivar(:session, person)
  @current_session.visits page_map[page]
end
```

Then

```
Then /(he|she) (should|should not) see (.*)/ do
  |_, yes_or_no, calendar|

  if yes_or_no == 'should'
    @current_session.response.should include_text(calendar)
  else
    @current_session.response.should_not include_text(calendar)
  end
end
```

Multiple Sessions

```ruby
def create_user_named(login, password=login)
  User.find_by_login(login).destroy rescue nil
  user = User.create!(
    :login => login,
    :password => password,
    :password_confirmation => password,
    :email => "#{login}@company.com"
  )
  yield user if block_given?
  user
end

def login_as(login, password=login)
  set_ivar :session, login, open_session { |user|
    user.visits "/sessions/new"
    user.fills_in :login, :with => login
    user.fills_in :password, :with => password
    user.clicks_button
  }
end
```

Multiple Sessions

```
module InstanceVariableHelpers
  def set_ivar(type, name, obj)
    instance_variable_set ivar_name(type, name), obj
  end

  def get_ivar(type, name)
    returning instance_variable_get(ivar_name(type, name)) do |obj|
      yield obj if block_given?
    end
  end

  private
    def ivar_name(type, name)
      "@#{type}_#{name.gsub(/[ -]/,'_').gsub('&','and')}"
    end
end
```

Thank You

- rspec.info
- blog.davidchelimsky.net
- articulatedman.com