

You're Doing it Wrong.

Ruby on Rails Antipatterns and Refactoring.

- Chad Pytel : thoughtbot.com
- Tammer Saleh : tammersaleh.com

*...authors of **Rails Antipatterns***

Pain & Money

Tackling a Rescue Mission refactoring.



AdminController#users

```
def users
  per_page = Variable::default_pagination_value
  @users = User.find(:all)
  # First, check to see if there
  # was an operation performed
  if not params[:operation].nil? then
  # there's more...
```

```
if (params[:operation] == "reset_password") then
  user = User.find(params[:id])
  user.generate_password_reset_access_key
  user.password_confirmation = user.password
  user.email_confirmation = user.email
  user.save!
  flash[:notice] = user.first_name + " " +
    user.last_name + "'s password has been reset."
end
# ...and more...
```

```
if (params[:operation] == "delete_user") then
  user = User.find(params[:id])
  user.item_status = ItemStatus.find_deleted()
  user.password_confirmation = user.password
  user.email_confirmation = user.email
  user.save!
  flash[:notice] = user.first_name + " " +
    user.last_name + " has been deleted"
end
# ...and still more...
```

```
if (params[:operation] == "activate_user") then
  user = User.find(params[:id])
  user.item_status = ItemStatus.find_active()
  user.password_confirmation = user.password
  user.email_confirmation = user.email
  user.save!
  flash[:notice] = user.first_name + " " +
    user.last_name + " has been activated"
end
end
# ...not done yet...
```

```
if (params[:operation] == "show_user") then
  @user = User.find(params[:id])
  render :template => show_user
  return true
end
# ...getting the idea?...
```

```
user_order = 'username'
if not params[:user_sort_field].nil? then
  user_order = params[:user_sort_field]
  if !session[:user_sort_field].nil? &&
    user_order == session[:user_sort_field] then
    user_order += " DESC"
  end
  session[:user_sort_field] = user_order
end
# ...almost there...
```

```
@user_pages, @users = paginate(:users,  
  :order => user_order,  
  :conditions => ['item_status_id <> ?',  
                 ItemStatus.find_deleted().id],  
  :per_page => per_page)  
end  
# ...and that was just one action.
```

So Many Choices...

- No tests
- Finding all users twice
- Bad validations on user
- Monolithic admin controller
- Finders in the controller
- No error reporting (*save!*)
- In-memory, old-style pagination
- Custom code instead of plugins for deleting, activating, etc.
- ...

So Little Time.

- Focus on client value
- Technical debt
- Highest yield change first

Don't get distracted.

- Take notes of refactoring issues as you see them.
- Create Pivotal Tracker chores, Lighthouse tickes, etc.
- Make sure client sees everything.

Monolithic controller

- /admin/users?operation=reset_password?id=x
- /admin/users?operation=delete_user?id=x
- /admin/users?operation=activate_user?id=x
- /admin/users?operation=show_user?id=x
- /admin/users

Convert to REST

Talk with your client.

Turns out, they never really needed the user activation functionality, so skip it.

Create a feature branch

- Isolate the refactoring from master so you can still fight fires while refactoring.
- Also protects you from the whole thing going painfully wrong.

- `git_remote_branch`

Write integration tests for existing behavior

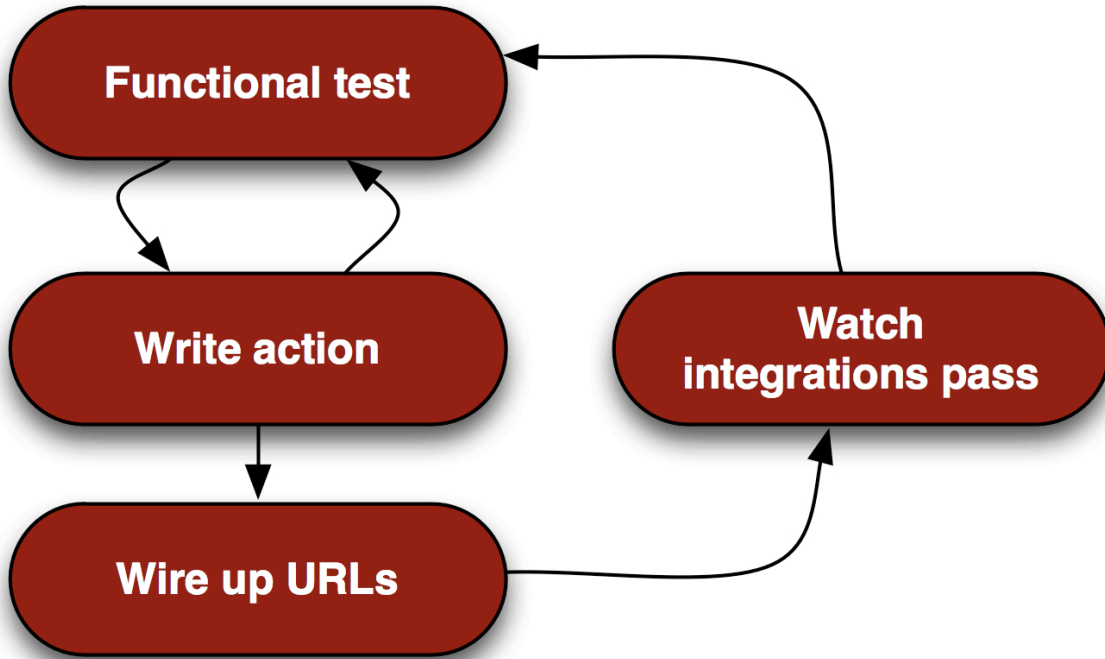
Write functional/unit tests for added or modified behavior

Start with integration tests for entire refactoring

- Write functional test - watch it fail

- Write action - watch the test pass
- Wire into urls

Repeat until integration tests pass



More about integration tests

- Cucumber stories or Rails integration tests
- Webrat is key

```
class AdminUsersTest < ActionController::IntegrationTest
  context "given some users" do
    setup { Factory(:user); Factory(:user) }

    a_logged_in_administrator do
      context "when viewing users" do
        setup { click_link 'admin'; click_link "users" }

        should "see all users" do
          User.each {|u| assert_contain u.first_name }
        end
      end
    end
  end
end
```

```
context "and drilling down to a single user" do
  setup do
    @user = User.first
    click_link @user.full_name
  end

  should "see that user" do
    assert_contain @user.first_name
    assert_contain @user.last_name
    assert_contain @user.email
  end
end
```

Add functional tests for new controller, one action at a time.

```
class Admin::UsersControllerTest < ActionController::TestCase
  should_route :get,
               "/admin/users",
               :controller => "admin/users",
               :action => "index"
```

```
when_logged_in_as_admin do
  context "given some users" do
    setup { Factory(:user); Factory(:user) }

    context "on GET to /admin/users" do
      setup { get :index }

      should_render_template :index
      should "assign users to @users" do
        assert assigns(:users)
        assert_equal User.count, assigns(:users).size
      end
    end
  end
end
```

```
class Admin::UsersController < Admin::BaseController
  def index
    per_page = Variable::default_pagination_value
    @users = User.find(:all)

    user_order = 'username'
    # ... same as before ...
    @user_pages, @users = paginate(:users,
      :order => user_order,
      :conditions => ['item_status_id <> ?',
        ItemStatus::DELETED],
      :per_page => per_page)
  end
end
```

```
def destroy
  @user = User.find(params[:id])
  @user.item_status = ItemStatus.find_deleted
  @user.save!
  flash[:notice] = "#{@user} has been removed."
  redirect_to admin_users_url
end
```

URL redirections?

- Most don't need it.
- Apache redirects or rewrites
- Rails redirect
- RSS readers don't redirect. Feedburner or rewrites.

Takeaways:

- Rescue missions are as much about process as about code.
- Client communication is key.
- Focus on client value.
- Use integration tests to verify large swaths of functionality.

General best practices for agile development

Rake Tasks

Are you testing them?

What makes them hard to test?

- By nature, scripts that live outside app
- Often have network and file access
- Often have output

Example Task

```
namespace :twitter do
  task :search => :environment do
    puts "Searching twitter."
    Twitter.search("@cpytel").each do |result|
      puts "Processing #{result.inspect}."
      alert = Alert.create(:body => result)
      alert.save_cache_file!
    end
  end
  puts "All done!"
end
```

One possible way to test

```
context "rake twitter:search" do
  setup do
    # How slow is this going to be? Very.
    @out = `cd #{Rails.root} &&
           rake twitter:search 2>&1`
  end
end
```

```
should "print a message at the beginning" do
  assert_match /Searching/i, @out
end
```

```
should "find all tweets containing @cpytel" do
  # this one would be based entirely on luck.
end
```

This has problems

- Slow
- No mocking or stubbing
- Task isn't in a transaction

Basically, no sandboxing

How do we fix this?

Rake tasks are just Ruby

Move it into the model

```
class Alert < ActiveRecord::Base
  def self.create_all_from_twitter_search(output = $stdout)
    output.puts "Searching twitter."
    Twitter.search("@cpytel").each do |result|
      output.puts "Processing #{result.inspect}."
      alert = create(:body => result)
      alert.save_cache_file!
    end
  end
end
```

```
    output.puts "All done!"
end

def save_cache_file!
  # Removes a file from the filesystem.
end
end
```

The rake task is nice and skinny

```
namespace :twitter do
  task :search => :environment do
    Alert.create_all_from_twitter_search
  end
end
```

Testing it is pretty much normal

```
# test/unit/alert_test.rb
class AlertTest < ActiveSupport::TestCase
  context "create_all_from_twitter_search" do
    setup do
      # Make sure none of the tests below hit the
      # network or touch the filesystem.
      Alert.any_instance.stubs(:save_cache_file!)
      Twitter.stubs(:search).returns([])
      @output = StringIO.new
    end
  end
end
```

```
should "print a message at the beginning" do
  Alert.create_all_from_twitter_search(@output)
  assert_match /Searching/i, @output.string
end
```

```
should "save some cache files" do
  Twitter.stubs(:search).returns(["one"])
  alert = mock("alert")
  alert.expects(:save_cache_file!)
  Alert.stubs(:create).returns(alert)
  Alert.create_all_from_twitter_search(@output)
end
```

```
should "find all tweets containing @cpytel" do
  Twitter.expects(:search).
    with("@cpytel").
      returns(["body"])
  Alert.create_all_from_twitter_search(@output)
end
```

We can mock and stub!

We can use normal tools

- Fakeweb
- FileUtils::NoWrite

In summary

- You can test drive development of your rake tasks
- Rake tasks should live inside a model

Rails Views

Know your Rails helpers

Know how they change

```
# Edit form
<% form_for :user,
            :url => user_path(@user),
            :html => {:method => :put} do |form| %>
```

```
<% form_for @user do |form| %>
```

- New HTML

```
<form action="/users" method="POST"
      class="new_user" id="new_user">
```

- Update HTML

```
<form action="/users/5" method="post"
      class="edit_user" id="edit_user_5">
  <div style="margin:0;padding:0">
    <input name="_method" type="hidden" value="put" />
  </div>
```

```
<!-- posts/index.html.erb -->
<% @posts.each do |post| -%>
  <h2><%=h post.title %></h2>
  <%= format_content post.body %>
  <p><%= link_to 'Email author',
               mail_to(post.user.email) %></p>
<% end -%>
```

Move the post content to a partial

```
<!-- posts/index.html.erb -->
<% @posts.each do |post| -%>
  <%= render :partial => 'post', :object => :post %>
<% end -%>
```

```
<!-- posts/_post.erb -->
<h2><%=h post.title %></h2>
<%= format_content post.body %>
<p><%= link_to 'Email author',
             mail_to(post.user.email) %></p>
```

Looping was built into render

```
<!-- posts/index.html.erb -->
<%= render :partial => 'post', :collection => @posts %>
```

```
<!-- posts/_post.erb -->
<h2><%=h post.title %></h2>
<%= format_content post.body %>
<p><%= link_to 'Email author',
             mail_to(post.user.email) %></p>
```

But there is duplication and common naming here


```
<%= render :partial => @posts %>
```

Dynamic page titles

```
<!-- layouts/application.html.erb -->
<head>
  <title>Acme Widgets : TX-300 Utility Widget</title>
</head>
```

```
class PagesController < ApplicationController
  def show
    @widget = Widgets.find(params[:id])
    @title = @widget.name
  end
end
```

```
<!-- layouts/application.html.erb -->
<head>
  <title>Acme Widgets : <%= @title %></title>
</head>
```

This is all presentation (view) stuff and there is a helper to do it in the view

```
<!-- layouts/application.html.erb -->
<head>
  <title>Acme Widgets : <%= yield(:title) %></title>
</head>

<!-- widgets/show.html.erb -->
<% content_for :title, @widget.title %>
```

Default title

```
<!-- layouts/application.html.erb -->
<head>
  <title>
    Acme Widgets :
    <%= yield(:title) || "Home" %>
  </title>
</head>
```

What else can we use this for

```
<!-- layouts/application.html.erb -->
<div class="sidebar">
  This is content for the sidebar.
  <%= link_to "Your Account", account_url %></li>
</div>

<div class="main">
  The main content of the page
</div>
```

```
<!-- layouts/application.html.erb -->
<%= yield(:sidebar) %>

<div class="main">
  The main content of the page
</div>

<!-- layouts/application.html.erb -->
<% content_for :sidebar do %>
  <div class="sidebar">
    This is content for the sidebar.
    <%= link_to "Your Account", account_url %></li>
  </div>
<% end %>
```

Avoid duplication

```
<!-- layouts/application.html.erb -->
<div class="sidebar">
  <%= yield(:sidebar) %>
</div>
```

```
<div class="main">
  The main content of the page
</div>
```

```
<!-- layouts/application.html.erb -->
<% content_for :sidebar do %>
  This is content for the sidebar.
  <%= link_to "Your Account", account_url %></li>
<% end %>
```

Conditional sidebar

```
<!-- layouts/application.html.erb -->
<% if yield(:sidebar) -%>
  <div class="sidebar">
    <%= yield(:sidebar) %>
  </div>
<% end -%>

<div class="main">
  The main content of the page
</div>

<!-- layouts/application.html.erb -->
<% content_for :sidebar do %>
  This is content for the sidebar.
  <%= link_to "Your Account", account_url %></li>
<% end %>
```

Read the book

Addison-Wesley Professional Ruby Series



Rough Cuts

RAILS ANTIPATTERNS

BEST PRACTICE RUBY ON
RAILS REFACTORING

CHAD PYTEL ■ TAMMER SALEH

Addison-Wesley Professional Ruby Series



Rough Cuts

RAILS ANTIPATTERNS

BEST PRACTICE RUBY ON
RAILS REFACTORING

CHAD PYTEL ■ TAMMER SALEH

Questions?

Audience submissions & one-offs.

You're doing it wrong.

```
photos = photos[0, photos.size > limit ?  
    limit :  
    photos.size]
```

Better

```
photos = photos[0, limit]
```

You're doing it wrong.

```
@order_type = params[:order_type] ?  
  params[:order_type] :  
  "New"
```

Better

```
params[:order_type] ||= "New"
```

Best

Set the default in the database. Rails will see it.

You're doing it wrong.

```
class Invitation < ActiveRecord::Base
  def encoded_id
    self.id*10011981 * 1820062 - 1981
  end
  # ...
end
```

You're (still) doing it wrong.

```
# ...
def self.decode_id(id)
  i = ((id.to_i + 1981)/1820062)/10011981
  if (i*10011981 * 1820062 - 1981)==id.to_i
    Invitation.find(i).id rescue false
  else
    false
  end
end
end
end
```

Better

```
class Invitation < ActiveRecord::Base
  def self.decoded_id(id)
    Base64.decode64s(id)
  end

  def self.find_by_encoded_id(id)
    find(decoded_id(id))
  end

  def encoded_id
    Base64.encode64s(id)
  end
end
```

