REFACTORING IN PRACTICE

WHO IS TEH ME?

Alex Sharp

Lead Developer at OptimisDev

```
{ :tweets_as => '@ajsharp'
  :blogs_at  => 'alexjsharp.com'
  :ships_at  => 'github.com/ajsharp' }
```

_*WHY* THIS TALK?

RAILS IS GROWING UP

RAILS IS ~~GROWING UP~~

MATURING

IT'S NOT ALL GREEN FIELDS AND "SOFT LAUNCHES"

APPS MATURE TOO

so what is this talk about?

large apps are a completely different beast than small apps

domain complexity

tight code coupling across domain concepts

these apps become harder to maintain as size increases

tight domain coupling exposes itself as a symptom of application size

this talk is about working with large apps

it's about staying on the

straight and narrow

it's about fixing bad code.

it's about responsibly fixing bad code.

it's about responsibly fixing ~~bad~~ and improving code.


RULES
OF
REFACTORING

RULE #1

TESTS ARE CRUCIAL

we're out of our element without tests

red, green, refactor doesn't work without the red

RULE #2

AVOID RABBIT HOLES

RULE #3:

TAKE SMALL WINS

I.E. AVOID THE **EPIC** REFACTOR

(IF POSSIBLE)

what do we *mean* when we use the term “refactor”

TRADITIONAL DEFINITION

To *refactor* code is to change it's implementation
while maintaining it's behavior.

- via Martin Fowler, Kent Beck et. al

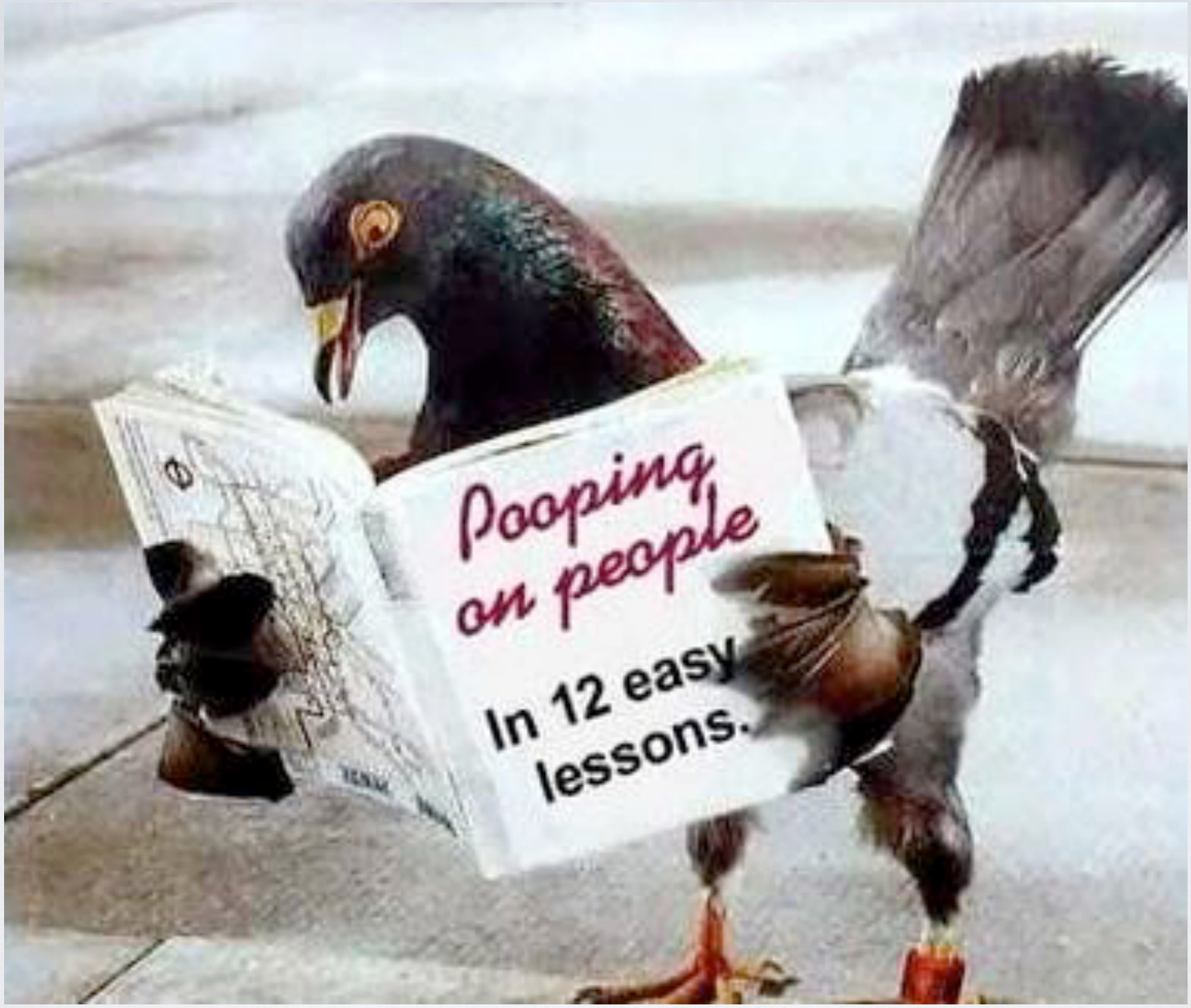
in web apps this means that the behavior for the *end user* remains unchanged

let's look at some code

DEPRECATE ACTION

MAJOR VIOLATIONS

- Zero tests
- Performance bottlenecks
  - main impetus for touching this code
- Phantom calling code
  - this is why we need to deprecate


Pooping
on people
In 12 easy
lessons.

FIRST STEP:

READ CODE

Before: /patients

```
def index
  @patients = Patient.search(params[:search], params[:page])

  respond_to do |format|
    format.html { render :layout => 'application' }
    format.json { render :json => @patients.to_json }
  end
end
```

Before: /patients

```
def index
  @patients = Patient.search(params[:search], params[:page])

  respond_to do |format|
    format.html { render :layout => 'application' }
    format.json { render :json => @patients.to_json }
  end
end
```

This method is crazy. Crazy slow, that is.

```ruby
  def self.search(search, page)
    includes = {}
    unless search.nil?
      if search.index('@')
        conditions = ['(contacts.email like ?) and contacts.ispatient = ?', "%#{search}%", true]
        includes = [:contact]
      elsif search.match(/^[0-9]+\//)
        d = Date.parse( search, true )
        if d.year > Date.today.year
          d = Date.civil( d.year - 100, d.month, d.day )
        end
        conditions = ['(patients.birth_date = ? )', d]
      elsif search.match(/^[0-9]+/)
        conditions = ['(patients.ssn like ? or phones.number like ?) and contacts.ispatient
= ?',  "%#{search}%", "%#{search}%", true]
        includes = {:contact => [:phones]}
      else
        conditions =  ['(patients.last_name like ? or patients.first_name like ?)', "%#{search}
%", "%#{search}%"]
      end
    end

    paginate :per_page => 15, :page => page,
           :include => includes,
           :conditions => conditions,
           :order => 'patients.last_name, patients.first_name'
  end
```

very hard to test

```
  def self.search(search, page)
    includes = {}
    unless search.nil?
      if search.index('@')
        conditions = ['(contacts.email like ?) and contacts.ispatient = ?', "%#{search}%", true]
        includes = [:contact]
      elsif search.match(/^[0-9]+\//)
        d = Date.parse( search, true )
        if d.year > Date.today.year
          d = Date.civil( d.year - 100, d.month, d.day )
        end
        conditions = ['(patients.birth_date = ? )', d]
      elsif search.match(/^[0-9]+/)
        conditions = ['(patients.ssn like ? or phones.number like ?) and contacts.ispatient
= ?',  "%#{search}%", "%#{search}%", true]
        includes = {:contact => [:phones]}
      else
        conditions =  ['(patients.last_name like ? or patients.first_name like ?)', "%#{search}
%", "%#{search}%"]
      end
    end

    paginate :per_page => 15, :page => page,
           :include => includes,
           :conditions => conditions,
           :order => 'patients.last_name, patients.first_name'
  end
```

```ruby
  def self.search(search, page)
    includes = {}
    unless search.nil?
      if search.index('@')
        conditions = ['(contacts.email like ?) and contacts.ispatient = ?', "%#{search}%", true]
        includes = [:contact]
      elsif search.match(/^[0-9]+\//)
        d = Date.parse( search, true )
        if d.year > Date.today.year
          d = Date.civil( d.year - 100, d.month, d.day )
        end
        conditions = ['(patients.birth_date = ? )', d]
      elsif search.match(/^[0-9]+/)
        conditions = ['(patients.ssn like ? or phones.number like ?) and contacts.ispatient
= ?',  "%#{search}%", "%#{search}%", true]
        includes = {:contact => [:phones]}
      else
        conditions =  ['(patients.last_name like ? or patients.first_name like ?)', "%#{search}
%", "%#{search}%"]
      end
    end

    paginate :per_page => 15, :page => page,
           :include => includes,
           :conditions => conditions,
           :order => 'patients.last_name, patients.first_name'
  end
```

could not figure out the bottleneck

```ruby
def self.search(search, page)
  includes = {}
  unless search.nil?
    if search.index('@')
      conditions = ['(contacts.email like ?) and contacts.ispatient = ?', "%#{search}%", true]
      includes = [:contact]
    elsif search.match(/^[0-9]+\//)
      d = Date.parse( search, true )
      if d.year > Date.today.year
        d = Date.civil( d.year - 100, d.month, d.day )
      end
      conditions = ['(patients.birth_date = ? )', d]
    elsif search.match(/^[0-9]+/)
      conditions = ['(patients.ssn like ? or phones.number like ?) and contacts.ispatient
= ?',  "%#{search}%", "%#{search}%", true]
      includes = {:contact => [:phones]}
    else
      conditions =  ['(patients.last_name like ? or patients.first_name like ?)', "%#{search}
%", "%#{search}%"]
    end
  end

  paginate :per_page => 15, :page => page,
           :include => includes,
           :conditions => conditions,
           :order => 'patients.last_name, patients.first_name'
end
```

good candidate for extraction

a slight improvement

```
  def self.search(search, page)

    includes = {}
    unless search.nil?
      # i.e. are we searching by email
      if search.index('@')
        conditions = ['(contacts.email like ?) and contacts.ispatient = ?', "%#{search}%", true]
        includes = [:contact]
      elsif search.match(/^[0-9]+\//)
        d = Date.parse( search, true )
        if d.year > Date.today.year
          d = Date.civil( d.year - 100, d.month, d.day )
        end
        conditions = ['(patients.birth_date = ? )', d]
      elsif search.match(/^[0-9]+/)
        conditions = ['(patients.ssn like ? or phones.number like ?) and contacts.ispatient = ?',  "%#{search}%", "%#{search}
%", true]
        includes = {:contact => [:phones]}
      else
        conditions =  ['(patients.last_name like ? or patients.first_name like ?)', "%#{search}%", "%#{search}%"]
      end
    end

    all_paginated(includes, conditions, page)
  end

  def self.all_paginated(includes, conditions, page)
    includes = { :contact => [:primary_phone] } if includes.empty?
    paginate(:per_page   => 15,
             :page       => page,
             :include    => includes,
             :conditions => conditions,
             :order      => 'patients.last_name, patients.first_name')
  end
```

so why not just fix the whole thing?

crazy phantom javascript code

We want to alter some behavior here

PatientsController#index

Calling Code

index.erb

show.erb

Known calls

phantom ajax call

Unknown calls

We want to alter some behavior here

Calling Code

PatientsControllerV2#index

index.erb

show.erb

Known calls

PatientsController#index

phantom ajax call

Unknown calls

After: /patients

```
class PatientsController < ApplicationController
  before_filter :deprecate_action, :only => [:index]

  def index
    @patients = Patient.search(params[:search], params[:page])

    respond_to do |format|
      # ...
    end
  end

  private
    def deprecate_action
      notify_hoptoad(:error_message => "DEPRECATION WARNING!: /
patients/index invoked")
    end
end
```

After: /patients

```
def deprecate_action
  notify_hoptoad(:error_message => "DEPRECATION WARNING!: /patients/
index invoked")
end
```

use hoptoad in production to notify us of hits to deprecated code

After: /patients

```
def deprecate_action
  notify_hoptoad(:error_message => "DEPRECATION WARNING!: /patients/
index invoked")
end
```

use hoptoad in production to notify us of hits to deprecated code

CAREFUL! This can crush your app w/ lots of requests. Consider pushing into a BG job.

After: /v2/patients

```
class V2::PatientsController < ApplicationController
  def index
    @patients = Patient.all_paginated([], [], params[:page])
  end
end
```

WINS

- Massive performance improvement
- Separation of responsibilities

COMMON PROBLEMS (I.E. ANTI-PATTERNS)

LACK OF TECHNICAL LEADERSHIP

LACK OF TECHNICAL LEADERSHIP

- Many common anti-pattens violated consistently

LACK OF TECHNICAL LEADERSHIP

- Many common anti-pattens violated consistently
    - DRY

LACK OF TECHNICAL LEADERSHIP

- Many common anti-pattens violated consistently
    - DRY
    - single responsibility

LACK OF TECHNICAL LEADERSHIP

- Many common anti-pattens violated consistently
    - DRY
    - single responsibility
    - large procedural code blocks

LACK OF PLATFORM KNOWLEDGE

- Often developers replicate things that Rails gives you for free
- This a frequent offense, but very low-hanging fruit to fix

LACK OF TESTS

A few typical scenarios:

LACK OF TESTS

A few typical scenarios:

- Aggressive deadlines: "there's no time for tests"

LACK OF TESTS

A few typical scenarios:

- Aggressive deadlines: "there's no time for tests"
- Lack of testing experience leads to abandonment of tests

LACK OF TESTS

A few typical scenarios:

- Aggressive deadlines: "there's no time for tests"
- Lack of testing experience leads to abandonment of tests
    - two files with tests in a 150+ model app

LACK OF TESTS

A few typical scenarios:

- Aggressive deadlines: "there's no time for tests"
- Lack of testing experience leads to abandonment of tests
    - two files with tests in a 150+ model app
- "We'll add tests later"

POOR DOMAIN MODELING

POOR DOMAIN MODELING

- UDD: UI-driven development

POOR DOMAIN MODELING

- UDD: UI-driven development
- Unfamiliarity with domain modeling

POOR DOMAIN MODELING

- UDD: UI-driven development
- Unfamiliarity with domain modeling
- probably b/c many apps don't need it

WRANGLING VIEW LOGIC WITH A PRESENTER

COMMON PROBLEMS

- i-var bloat
- results in difficult to test controllers
    - high barriers like this typically lead to developers just not testing
- results in brittle views
- instance variables are an implementation, not an interface

THE BILLINGS SCREEN

THE BILLINGS SCREEN

- Core functionality was simply broken, not working

THE BILLINGS SCREEN

- Core functionality was simply broken, not working
- This area of the app is somewhat of a "shanty town"

1. READ CODE

```ruby
class BillingsController < ApplicationController
  def index
    @visit_statuses = Visit::Statuses
    @visit_status = 'Complete'
    @visit_status = params[:visit_status] unless params[:visit_status].blank?

    @billing_status =
      if  @visit_status.upcase != 'COMPLETE' then ''
      elsif ( params[:billing_status] && params[:billing_status] != 'â€šÃ„Ã¬' ) then params[:billing_status]
      else Visit::Billing_pending
      end

    @noted = 'true'
    @noted = 'false' if params[:noted] == 'false'

    @clinics = current_user.allclinics( @practice.id )
    @visits = current_clinic.visits.billable_visits(@billing_status, @visit_status).order_by("visit_date")

    session[:billing_visits] = @visits.collect{ |v| v.id }
  end
end
```

```ruby
class BillingsController < ApplicationController
  def index
    @visit_statuses = Visit::Statuses
    @visit_status = 'Complete'
    @visit_status = params[:visit_status] unless params[:visit_status].blank?

    @billing_status =
      if  @visit_status.upcase != 'COMPLETE' then ''
      elsif ( params[:billing_status] && params[:billing_status] != 'â€šÃ„Ã¬' ) then params[:billing_status]
      else Visit::Billing_pending
      end

    @noted = 'true'
    @noted = 'false' if params[:noted] == 'false'

    @clinics = current_user.allclinics( @practice.id )
    @visits = current_clinic.visits.billable_visits(@billing_status, @visit_status).order_by("visit_date")

    session[:billing_visits] = @visits.collect{ |v| v.id }
  end
end
```

```
%th=collection_select(:search, :clinic_id, @clinics, :id, :name , {:prompt => "All"}, \
  :class => 'filterable')
%th=select_tag('search[visit_status_eq]', \
  options_for_select( Visit::Statuses.collect(&:capitalize), @visit_status ), {:class => 'filterable'})
%th=select_tag('search[billing_status_eq]', \
  options_for_select([ "Pending", "Rejected", "Processed" ], @billing_status), {:class => 'filterable'})
%th=collection_select(:search, :resource_id, @providers, :id, :full_name, { :prompt => 'All' }, \
  :class => 'filterable')
%tbody
= render :partial => 'visit', :collection => @visits
```

```
%th=collection_select(:search, :clinic_id, @clinics  :id, :name , {:prompt => "All"}, \
  :class => 'filterable')
%th=select_tag('search[visit_status_eq]', \
  options_for_select( Visit::Statuses.collect(&:capitalize), @visit_status ), {:class => 'filterable'})
%th=select_tag('search[billing_status_eq]', \
  options_for_select([ "Pending", "Rejected", "Processed" ], @billing_status), {:class => 'filterable'})
%th=collection_select(:search, :resource_id, @providers, :id, :full_name, { :prompt => 'All' }, \
  :class => 'filterable')
%tbody
= render :partial => 'visit', :collection => @visits
```

SECOND STEP:
DOCUMENT SMELLS

SMELLS

- Tricky presentation logic quickly becomes brittle
- Lot's of hard-to-test, important code
- Needlessly storing constants in ivars
- Craziness going on with @billing_status. Seems important.

```ruby
class BillingsController < ApplicationController
  def index
    @visit_statuses = Visit::Statuses
    @visit_status = 'Complete'
    @visit_status = params[:visit_status] unless params[:visit_status].blank?

    @billing_status =
      if  @visit_status.upcase != 'COMPLETE' then ''
      elsif ( params[:billing_status] && params[:billing_status] != '-' ) then params[:billing_status]
      else Visit::Billing_pending
      end

    @noted = 'true'
    @noted = 'false' if params[:noted] == 'false'

    @clinics = current_user.allclinics( @practice.id )
    @visits = current_clinic.visits.billable_visits(@billing_status, @visit_status).order_by("visit_date")

    session[:billing_visits] = @visits.collect{ |v| v.id }
  end
end
```

seems important...maybe for searching?

```ruby
class BillingsController < ApplicationController
  def index
    @visit_statuses = Visit::Statuses
    @visit_status = 'Complete'
    @visit_status = params[:visit_status] unless params[:visit_status].blank?

    @billing_status =
      if  @visit_status.upcase != 'COMPLETE' then ''
      elsif ( params[:billing_status] && params[:billing_status] != '-' ) then params[:billing_status]
      else Visit::Billing_pending
      end

    @noted = 'true'
    @noted = 'false' if params[:noted] == 'false'

    @clinics = current_user.allclinics( @practice.id )
    @visits = current_clinic.visits.billable_visits(@billing_status, @visit_status).order_by("visit_date")

    session[:billing_visits] = @visits.collect{ |v| v.id }
  end
end
```

WTF!?!?

A QUICK NOTE:

“PERPETUAL WTF SYNDROME”

TOO MANY WTF'S WILL DRIVE YOU NUTS

AWFUL CODE IS DEMORALIZING

YOU FORGET THAT GOOD CODE EXISTS

THAT'S ONE REASON WHY THE NEXT STEP IS SO IMPORTANT

IT'S KEEPS THE DIALOGUE POSITIVE

3. IDENTIFY ENGINEERING GOALS

ENGINEERING GOALS

- *Program to an interface, not an implementation*
- Only one instance variable
- Use extract class to wrap up presentation logic

```ruby
describe BillingsPresenter do
  before do
    @practice = mock_model(Practice)
    @user     = mock_model(User, :practice => @practice)
    @presenter =  BillingsPresenter.new({}, @user,
@practice)
  end

  describe '#visits' do
    it 'should receive billable_visits' do
      @practice.should_receive(:billable_visits)
      @presenter.visits
    end
  end

  describe '#visit_status' do
    it 'should default to the complete status' do
      @presenter.visit_status.should == "Complete"
    end
  end

  describe '#billing_status' do
    it 'should default to default_billing_status' do
      @presenter.billing_status.should ==
BillingsPresenter.default_billing_status
    end

    it 'should use the params[:billing_status] if it exists'
do
      presenter = BillingsPresenter.new({:billing_status =>
'use me'}, @user, @practice)
      presenter.billing_status.should == 'use me'
    end
  end

  describe '#clinics' do
    it 'should receive user.allclinics' do
      @user.should_receive(:allclinics).with(@practice.id)
      @presenter.clinics
    end
  end

  describe '#providers' do
    it 'should get the providers from the visit' do
      visit = mock_model Visit
      @presenter.stub!(:visits).and_return([visit])
      visit.should_receive(:resource)

      @presenter.providers
    end
  end

  describe '.default_visit_status' do
    it 'is the capitalized version of COMPLETE' do
      BillingsPresenter.default_visit_status.should ==
'Complete'
    end
  end

  describe '.default_billing_status' do
    it 'is the capitalized version of COMPLETE' do
      BillingsPresenter.default_billing_status.should ==
'Pending'
    end
  end

  describe '#visit_statuses' do
    it 'is the capitalized version of the visit statuses' do
      @presenter.visit_statuses.should == [['All', '']] +
Visit::Statuses.collect(&:capitalize)
    end
  end

  describe '#billing_stauses' do
    it 'is the capitalized version of the billing statuses'
do
      @presenter.billing_statuses.should == [['All', '']] +
Visit::Billing_statuses.collect(&:capitalize)
    end
  end

end
```

```ruby
class BillingsPresenter
  attr_reader :params, :user, :practice
  def initialize(params, user, practice)
    @params   = params
    @user     = user
    @practice = practice
  end

  def self.default_visit_status
    Visit::COMPLETE.capitalize
  end

  def self.default_billing_status
    Visit::Billing_pending.capitalize
  end

  def visits
    @visits ||= practice.billable_visits(params[:search])
  end

  def visit_status
    params[:visit_status] || self.class.default_visit_status
  end

  def billing_status
    params[:billing_status] || self.class.default_billing_status
  end

  def clinics
    @clinics ||= user.allclinics practice.id
  end

  def providers
    @providers ||= visits.collect(&:resource).uniq
  end

  def visit_statuses
    [['All', '']] + Visit::Statuses.collect(&:capitalize)
  end

  def billing_statuses
    [['All', '']] + Visit::Billing_statuses.collect(&:capitalize)
  end
end
```

```ruby
class BillingsController < ApplicationController
  def index
    @presenter = BillingsPresenter.new params, current_user, current_practice

    session[:billing_visits] = @presenter.visits.collect{ |v| v.id }
  end
end
```

```
%th=collection_select(:search, :clinic_id, @presenter.clinics, :id, :name , {:prompt => "All"}, \
  :class => 'filterable')
%th=select_tag('search[visit_status_eq]', \
  options_for_select( Visit::Statuses.collect(&:capitalize), @presenter.visit_status ), { \
   :class => 'filterable'})
%th=select_tag('search[billing_status_eq]', \
  options_for_select([ "Pending", "Rejected", "Processed" ], @presenter.billing_status), {:class =>
'filterable'})
%th=collection_select(:search, :resource_id, @presenter.providers, :id, :full_name, { :prompt => 'All' },
\
  :class => 'filterable')
%tbody
= render :partial => 'visit', :collection => @presenter.visits
```

WINS

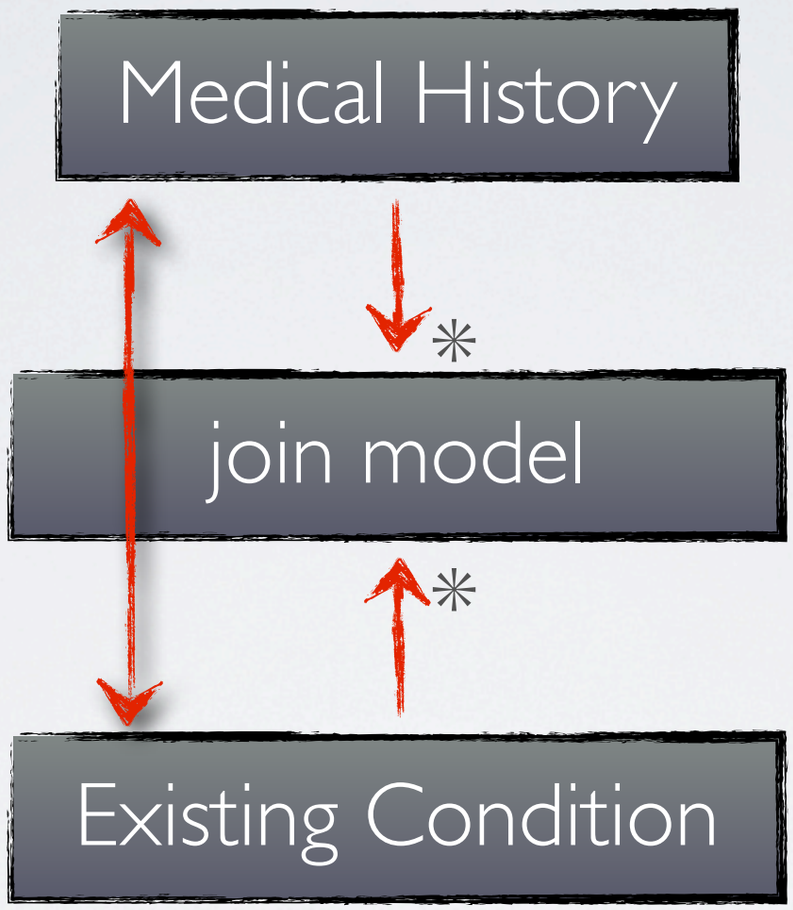
- Much cleaner, more testable controller
- Much less brittle view template
- The behavior of this action actually works

FAT MODEL / SKINNY CONTROLLER

UPDATE ACTION CRAZINESS

- Update a collection of associated objects in the parent object's update action

Object Model



STEP 1: READ CODE

```ruby
def update
  @medical_history = @patient.medical_histories.find(params[:id])

  @medical_history.taken_date = Date.today if @medical_history.taken_date.nil?

  success = true
  conditions = @medical_history.existing_conditions_medical_histories.find(:all)

  params[:conditions].each_pair do |key,value|
    condition_id = key.to_s[10..-1].to_i
    condition = conditions.detect {|x| x.existing_condition_id == condition_id }
    if condition.nil?
      success = @medical_history.existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id, :has_condition => value) && success
    elsif condition.has_condition != value
      success = condition.update_attribute(:has_condition, value) && success
    end
  end

  respond_to do |format|
    if @medical_history.update_attributes(params[:medical_history]) && success
      format.html {
                    flash[:notice] = 'Medical History was successfully updated.'
                    redirect_to( patient_medical_histories_url(@patient) ) }
      format.xml  { head :ok }
      format.js
    else
      format.html { render :action => "edit" }
      format.xml  { render :xml => @medical_history.errors, :status => :unprocessable_entity }
      format.js   { render :text => "Error Updating History", :status => :unprocessable_entity}
    end
  end
end
```

```ruby
def update
  @medical_history = @patient.medical_histories.find(params[:id])

  @medical_history.taken_date = Date.today if @medical_history.taken_date.nil?

  success = true
  conditions = @medical_history.existing_conditions_medical_histories.find(:all)

  params[:conditions].each_pair do |key,value|
    condition_id = key.to_s[10..-1].to_i
    condition = conditions.detect {|x| x.existing_condition_id == condition_id }
    if condition.nil?
      success = @medical_history.existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id, :has_condition => value) && success
    elsif condition.has_condition != value
      success = condition.update_attribute(:has_condition, value) && success
    end
  end

  respond_to do |format|
    if @medical_history.update_attributes(params[:medical_history]) && success
      format.html {
                    flash[:notice] = 'Medical History was successfully updated.'
                    redirect_to( patient_medical_histories_url(@patient) ) }
      format.xml  { head :ok }
      format.js
    else
      format.html { render :action => "edit" }
      format.xml  { render :xml => @medical_history.errors, :status => :unprocessable_entity }
      format.js   { render :text => "Error Updating History", :status => :unprocessable_entity}
    end
  end
end
```

smell-fest

STEP 2: DOCUMENT SMELLS

Lot's of violations:

1. Fat model, skinny controller
2. Single responsibility
3. Not modular
4. Repetition of rails
5. Really hard to fit on a slide

Let's focus here

```
conditions = @medical_history.existing_conditions_medical_histories.find(:all)

params[:conditions].each_pair do |key,value|
  condition_id = key.to_s[10..-1].to_i
  condition = conditions.detect {|x| x.existing_condition_id == condition_id }
  if condition.nil?
    success = @medical_history.existing_conditions_medical_histories.create(
      :existing_condition_id => condition_id, :has_condition => value) && success
  elsif condition.has_condition != value
    success = condition.update_attribute(:has_condition, value) && success
  end
end
```

STEP 3: IDENTIFY ENGINEERING GOALS

I want to refactor in the following ways:

1. Push this code into the model

2. Extract varying logic into separate methods

We need to write some characterization tests and

get this action under test so we can figure out what it's doing.

This way we can rely on an automated testing

workflow for instant feedback

```ruby
describe MedicalHistoriesController, "PUT update" do
  context "successful" do
    before :each do
      @user = Factory(:practice_admin)
      @patient         = Factory(:patient_with_medical_histories, :practice => @user.practice)
      @medical_history = @patient.medical_histories.first

      @condition1      = Factory(:existing_condition)
      @condition2      = Factory(:existing_condition)

      stub_request_before_filters @user, :practice => true, :clinic => true

      params = { :conditions =>
        { "condition_#{@condition1.id}" => "true",
          "condition_#{@condition2.id}" => "true" },
        :id => @medical_history.id,
        :patient_id => @patient.id
      }

      put :update, params
    end

    it { should     redirect_to patient_medical_histories_url }
    it { should_not render_template :edit }

    it "should successfully save a collection of conditions" do
      @medical_history.existing_conditions.should include @condition1
      @medical_history.existing_conditions.should include @condition2
    end
  end
end
```

we can eliminate this, b/c it's an association of this model

```
conditions = @medical_history.existing_conditions_medical_histories.find(:all)
```

```
params[:conditions].each_pair do |key,value|
  condition_id = key.to_s[10..-1].to_i
  condition = conditions.detect {|x| x.existing_condition_id == condition_id }
  if condition.nil?
    success = @medical_history.existing_conditions_medical_histories.create(
      :existing_condition_id => condition_id, :has_condition => value) && success
  elsif condition.has_condition != value
    success = condition.update_attribute(:has_condition, value) && success
  end
end
```

Next, do extract method on this line

```
  params[:conditions].each_pair do |key,value|
    condition_id = key.to_s[10..-1].to_i
    condition = existing_conditions_medical_histories.detect {|x|
x.existing_condition_id == condition_id }
    if condition.nil?
      success = existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id, :has_condition => value) && success
    elsif condition.has_condition != value
      success = condition.update_attribute(:has_condition, value) && success
    end
  end
```

Cover this new method w/ tests

```
  params[:conditions].each_pair do |key,value|
    condition_id = get_id_from_string(key)
    condition = existing_conditions_medical_histories.detect {|x|
x.existing_condition_id == condition_id }
    if condition.nil?
      success = existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id, :has_condition => value) && success
    elsif condition.has_condition != value
      success = condition.update_attribute(:has_condition, value) && success
    end
  end
```

???

```
  params[:conditions].each_pair do |key,value|
    condition_id = get_id_from_string(key)
    condition = existing_conditions_medical_histories.detect {|x|
x.existing_condition_id == condition_id }
    if condition.nil?
      success = existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id, :has_condition => value) && success
    elsif condition.has_condition != value
      success = condition.update_attribute(:has_condition, value) && success
    end
  end
```

i.e. ecmh.find(:first, :conditions => { :existing_condition_id => condition_id })

```
params[:conditions].each_pair do |key,value|
  condition_id = get_id_from_string(key)
  condition = existing_conditions_medical_histories.detect {|x|
x.existing_condition_id == condition_id }
  if condition.nil?
    success = existing_conditions_medical_histories.create(
      :existing_condition_id => condition_id, :has_condition => value) && success
  elsif condition.has_condition != value
    success = condition.update_attribute(:has_condition, value) && success
  end
end
```

do extract method here...

```
  params[:conditions].each_pair do |key,value|
    condition_id = get_id_from_string(key)
    condition = existing_conditions_medical_histories.detect {|x|
x.existing_condition_id == condition_id }
    if condition.nil?
      success = existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id, :has_condition => value) && success
    elsif condition.has_condition != value
      success = condition.update_attribute(:has_condition, value) && success
    end
  end
```

do extract method here...

```
params[:conditions].each_pair do |key,value|
  condition_id = get_id_from_string(key)
  condition = find_existing_condition(condition_id)
  if condition.nil?
    success = existing_conditions_medical_histories.create(
      :existing_condition_id => condition_id, :has_condition => value) && success
  elsif condition.has_condition != value
    success = condition.update_attribute(:has_condition, value) && success
  end
end
```

???

```
params[:conditions].each_pair do |key,value|
  condition_id = get_id_from_string(key)
  condition = find_existing_condition(condition_id)
  if condition.nil?
    success = existing_conditions_medical_histories.create(
      :existing_condition_id => condition_id, :has_condition => value) && success
  elsif condition.has_condition != value
    success = condition.update_attribute(:has_condition, value) && success
  end
end
```

i.e. create or update

```
params[:conditions].each_pair do |key,value|
  condition_id = get_id_from_string(key)
  condition = find_existing_condition(condition_id)
  if condition.nil?
    success = existing_conditions_medical_histories.create(
      :existing_condition_id => condition_id, :has_condition => value) && success
  elsif condition.has_condition != value
    success = condition.update_attribute(:has_condition, value) && success
  end
end
```

```ruby
describe MedicalHistory, "#update_conditions" do
  context "when passed a condition that is not an existing conditions" do
    before do
      @medical_history = Factory(:medical_history)
      @condition       = Factory(:existing_condition)

      @medical_history.update_conditions({ "condition_#{@condition.id}" => "true" })
    end

    it "should add the condition to the existing_condtions association" do
      @medical_history.existing_conditions.should include @condition
    end

    it "should set the :has_condition attribute to the value that was passed in" do
      @medical_history.existing_conditions_medical_histories.first.has_condition.should == true
    end
  end

  context "when passed a condition that is an existing condition" do
    before do
      ecmh             = Factory(:existing_conditions_medical_history, :has_condition => true)
      @medical_history = ecmh.medical_history
      @condition       = ecmh.existing_condition

      @medical_history.update_conditions({ "condition_#{@condition.id}" => "false" })
    end

    it "should update the existing_condition_medical_history record" do
      @medical_history.existing_conditions.should_not include @condition
    end

    it "should set the :has_condition attribute to the value that was passed in" do
      @medical_history.existing_conditions_medical_histories.first.has_condition.should == false
    end
  end

end
```

```ruby
  def update_conditions(conditions_param = {})
    conditions_param.each_pair do |key, value|
      create_or_update_condition(get_id_from_string(key), value)
    end if conditions_param
  end

  private
    def create_or_update_condition(condition_id, value)
      condition = existing_conditions_medical_histories.find_by_existing_condition_id
(condition_id)
      condition.nil? ? create_condition(condition_id, value) : update_condition(condition,
value)
    end

    def create_condition(condition_id, value)
      existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id,
        :has_condition => value
      )
    end

    def update_condition(condition, value)
      condition.update_attribute(:has_condition, value) unless condition.has_condition == value
    end
```

We went from this mess in the controller

```
def update
  @medical_history = @patient.medical_histories.find(params[:id])

  @medical_history.taken_date = Date.today if @medical_history.taken_date.nil?

  success = true
  conditions = @medical_history.existing_conditions_medical_histories.find(:all)

  params[:conditions].each_pair do |key,value|
    condition_id = key.to_s[10..-1].to_i
    condition = conditions.detect {|x| x.existing_condition_id == condition_id }
    if condition.nil?
      success = @medical_history.existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id, :has_condition => value) && success
    elsif condition.has_condition != value
      success = condition.update_attribute(:has_condition, value) && success
    end
  end

  respond_to do |format|
    if @medical_history.update_attributes(params[:medical_history]) && success
      format.html {
                    flash[:notice] = 'Medical History was successfully updated.'
                    redirect_to( patient_medical_histories_url(@patient) ) }
      format.xml  { head :ok }
      format.js
    else
      format.html { render :action => "edit" }
      format.xml  { render :xml => @medical_history.errors, :status => :unprocessable_entity }
      format.js   { render :text => "Error Updating History", :status => :unprocessable_entity}
    end
  end
end
```

To this in the controller...

```
def update
  @medical_history = @patient.medical_histories.find(params[:id])
  @medical_history.taken_date = Date.today if @medical_history.taken_date.nil?

  respond_to do |format|
    if( @medical_history.update_attributes(params[:medical_history]) &&
        @medical_history.update_conditions(params[:conditions]) )
      format.html {
                    flash[:notice] = 'Medical History was successfully updated.'
                    redirect_to( patient_medical_histories_url(@patient) ) }
      format.js
    else
      format.html { render :action => "edit" }
      format.js   { render :text => "Error Updating History",
                           :status => :unprocessable_entity }
    end
  end
end
```

and this in the model

```
  def update_conditions(conditions_param = {})
    conditions_param.each_pair do |key, value|
      create_or_update_condition(get_id_from_string(key), value)
    end if conditions_param
  end

  private
    def create_or_update_condition(condition_id, value)
      condition = existing_conditions_medical_histories.find_by_existing_condition_id
(condition_id)
      condition.nil? ? create_condition(condition_id, value) : update_condition(condition,
value)
    end

    def create_condition(condition_id, value)
      existing_conditions_medical_histories.create(
        :existing_condition_id => condition_id,
        :has_condition => value
      )
    end

    def update_condition(condition, value)
      condition.update_attribute(:has_condition, value) unless condition.has_condition == value
    end
```

IT'S NOT PERFECT...

belongs in model

```
def update
  @medical_history = @patient.medical_histories.find(params[:id])
  @medical_history.taken_date = Date.today if @medical_history.taken_date.nil?

  respond_to do |format|
    if( @medical_history.update_attributes(params[:medical_history]) &&
        @medical_history.update_conditions(params[:conditions]) )
      format.html {
                    flash[:notice] = 'Medical History was successfully updated.'
                    redirect_to( patient_medical_histories_url(@patient) ) }
      format.js
    else
      format.html { render :action => "edit" }
      format.js   { render :text => "Error Updating History",
                           :status => :unprocessable_entity }
    end
  end
end
```

maybe a DB transaction? maybe nested params?

BUT REFACTORING MUST BE DONE IN SMALL STEPS

RECAP: WINS

- improved controller action API
- intention-revealing method names communicate what's occurring at each stage of the update
- clean, testable public model API

PROCESS RECAP

1. READ CODE

2. DOCUMENT SMELLS

3. IDENTIFY ENGINEERING GOALS

"We're going to push this nastiness down into the model. That way we can get it tested, so we can more easily change it in the future."

Rate my talk: http://bit.ly/ajsharp-hoedown-refactoring

Slides: http://slidesha.re/ajsharp-hoedown-refactoring-slides

RESOURCES

- Talks/slides
  - Rick Bradley's railsconf slides: http://railsconf2010.rickbradley.com/
  - Rick's (beardless) hoedown 08 talk: http://tinyurl.com/flogtalk
- Books
  - *Working Effectively with Legacy Code* by Michael Feathers
  - *Refactoring: Ruby Edition* by Jay Fields, Martin Fowler, et. al
  - *Domain Driven Design* by Eric Evans

CONTACT

- ajsharp@gmail.com
- @ajsharp on twitter

THANKS!