Rails AntiPatterns

Best Practice Refactoring

Chad Pytel
cpytel@thoughtbot.com

Tammer Saleh
tsaleh@thoughtbot.com

This presentation is not going to teach you how to do test driven development.

In order to refactor your application must be backed by a good test suite.

This presentation is going to show you code that you might recognize.

This presentation is going to show you how to make that code better.

Moving Code from the Controller to the Models

An Offending Controller

```
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication_id = @publication.id
    @article.reporter_id = current_user.id

    begin
      Article.transaction do
        @version = @article.create_version!(params[:version],
                                            current_user)
      end
    rescue ActiveRecord::RecordNotSaved, ActiveRecord::RecordInvalid
      index
      render :action => :index and return false
    end
    redirect_to article_path(@publication, @article)
  end
```

```ruby
class ArticlesController < ApplicationController

  #...

 def update
    old_state = @article.state

    year = params[:article].delete("to_be_published_at(1i)")
    month = params[:article].delete("to_be_published_at(2i)")
    day = params[:article].delete("to_be_published_at(3i)")
    hour = params[:article].delete("to_be_published_at(4i)")
    minute = params[:article].delete("to_be_published_at(5i)")
    to_be_published_at = (year and month and day and hour and minute ?
                          Time.mktime(year,
                                      month, day, hour, minute) : nil)

    saved = Article.transaction do
      if params[:send] or params[:new_version] or params[:subedit] or params[:back] or
params[:send_later] or @article.state == "Published" or current_user !=
@article.current_version.writer
        @version = @article.create_version!(params[:article], current_user)
      else
        @version = @article.current_version
        @version.attributes = params[:article]
        @version.writer_id = current_user.id
        @version.written_at = Time.now
      end

      # update existing corrections
      @version.corrections = params[:corrections]

      # add correction for published article
      if old_state == "Published"
        if params[:correction] and !params[:correction][:correction].blank?
          @correction = @version.corrections.build(params[:correction])
          @correction.person = current_user.name
        end
      end

      @version.authors = params[:authors]
      @version.categories = params[:categories]
      @version.tags = params[:tags]
      @version.references = params[:references]
      @version.relateds = params[:relateds]
      @version.links = params[:links]
```

```
     state_map = {
        "Raw" => {:send => "Edit"},
        "Edit Check" => {:send => "Edit"},
        "Edit" => {:send => "Published", :back => "Edit Check", :subedit => "Sub Edit", :send_later
=> "Publish Ready"},
        "Sub Check" => {:send => "Published", :back => "Edit Check", :subedit => "Sub
Edit", :send_later => "Publish Ready"},
        "Sub Edit" => {:subedit => "Sub Check"},
        "Publish Ready" => {:send_later => "Publish Ready"}
      }

      # if everything else is valid, then save the state
      if params[:send] and @version.valid?
        @version.state = state_map[old_state][:send]
      elsif params[:send_later] and @version.valid?
        @version.state = state_map[old_state][:send_later]
      elsif params[:subedit] and @version.valid?
        @version.state = state_map[old_state][:subedit]
      elsif params[:back] and @version.valid?
        @version.state = state_map[old_state][:back]
      end

      # default and fallback
      @version.state ||= @article.state

      # set publish dates if needed
      if @version.state == "Published"
        published_at = Time.now
        @article.first_published_at ||= published_at
        @article.last_published_at = published_at
      elsif @version.state == "Publish Ready"
        @article.to_be_published_at = to_be_published_at if to_be_published_at
      end
      if !params[:pdf].blank?
        @article.pdf = params[:pdf]
        @article.print = true
      end

      @article.save!
      @version.save!
    end rescue false
```

```
      if saved
        post_to_api(:staging)
        email_notifications(@article, old_state, @version.state)
        if @version.state == "Published"
          # this may become a special republish api call that doesn't have to do as much work if a
correction has been specified (meaning it's a republish)
          post_to_api(:live)
        end

        if params[:send] and @version.state == "Edit"
          flash[:success] = "Article saved and ready for editing."
        elsif params[:subedit]
          if @version.state == "Sub Check"
            flash[:success] = "Article sent back for editing."
          elsif @version.state == "Sub Edit"
            flash[:success] = "Article sent to sub editors."
          end
        elsif params[:back]
          if @version.state == "Edit Check"
            flash[:success] = "Article sent back to reporter for further work."
          end
        elsif @version.state == "Publish Ready"
          flash[:success] = "Article has been marked for publishing at a later date."
        elsif @version.state == "Published" and old_state != "Published"
          flash[:success] = "Article saved and published to live server."
        elsif @version.state == "Published"
          flash[:success] = "Article has been republished."
        end

        # default and fallback
        flash[:success] ||= "Article updated."

        # this is temporary, as dispatcher is set up only on staging environment for now
        if RAILS_ENV == "staging"
          flash[:success] += " Changes may not be immediately available for preview on foxtrot."
        end

        redirect_to article_path(@publication, @article) and return true
      else
        load_article_data

        render :action => 'edit' and return false
      end
    end
```

An Offending Controller

```
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication_id = @publication.id
    @article.reporter_id = current_user.id

    begin
      Article.transaction do
        @version = @article.create_version!(params[:version],
                                            current_user)
      end
    rescue ActiveRecord::RecordNotSaved, ActiveRecord::RecordInvalid
      index
      render :action => :index and return false
    end
    redirect_to article_path(@publication, @article)
  end
```

A Better Controller

```ruby
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication_id = @publication.id
    @article.reporter_id = current_user.id

    if @article.save
      redirect_to article_path(@publication, @article)
    else
      index
      render :action => :index
    end
  end
```

Here is How We Get There

The Current create_version! Method

```
def create_version!(attributes, user)
  return create_first_version!(attributes, user) if self.versions.empty?

  # mark old related links as not current
  if self.current_version.relateds.any?
    self.current_version.relateds.each { |rel|
      rel.update_attribute(:current, false) }
  end

  version = self.versions.build(attributes)
  version.article_id = self.id
  version.written_at = Time.now
  version.writer_id = user.id
  version.version = self.current_verison.version + 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

The Current create_first_version! Method

```ruby
def create_first_version!(attributes, user)
  version = self.versions.build(attributes)
  version.written_at = Time.now
  version.writer_id = user.id
  version.state ||= "Raw"
  version.version = 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

Identifying Things to Refactor: Similar Code

```
def create_first_version!(attributes, user)
  version = self.versions.build(attributes)
  version.written_at = Time.now
  version.writer_id = user.id
  version.state ||= "Raw"
  version.version = 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end

def create_version!(attributes, user)
  return create_first_version!(attributes, user) if self.versions.empty?

  # mark old related links as not current
  if self.current_version.relateds.any?
    self.current_version.relateds.each { |rel| rel.update_attribute(:current, false) }
  end

  version = self.versions.build(attributes)
  version.article_id = self.id
  version.written_at = Time.now
  version.writer_id = user.id
  version.version = self.current_version.version + 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

Unnecessary Code

```ruby
def create_first_version!(attributes, user)
  version = self.versions.build(attributes)
  version.written_at = Time.now
  version.writer_id = user.id
  version.state ||= "Raw"
  version.version = 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end

def create_version!(attributes, user)
  return create_first_version!(attributes, user) if self.versions.empty?

  # mark old related links as not current
  if self.current_version.relateds.any?
    self.current_version.relateds.each { |rel| rel.update_attribute(:current, false) }
  end

  version = self.versions.build(attributes)
  version.article_id = self.id
  version.written_at = Time.now
  version.writer_id = user.id
  version.version = self.current_version.version + 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

Use Active Record's Built-in Functionality

```ruby
def create_first_version!(attributes, user)
  version = self.versions.build(attributes)
  version.written_at = Time.now
  version.writer_id = user.id
  version.state ||= "Raw"
  version.version = 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end

def create_version!(attributes, user)
  return create_first_version!(attributes, user) if self.versions.empty?

  # mark old related links as not current
  if self.current_version.relateds.any?
    self.current_version.relateds.each { |rel| rel.update_attribute(:current, false) }
  end

  version = self.versions.build(attributes)
  version.written_at = Time.now
  version.writer_id = user.id
  version.version = self.current_version.version + 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

Manually Setting Default Values

```
def create_first_version!(attributes, user)
  version = self.versions.build(attributes)
  version.writer_id = user.id
  version.state ||= "Raw"
  version.version = 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end

def create_version!(attributes, user)
  return create_first_version!(attributes, user) if self.versions.empty?

  # mark old related links as not current
  if self.current_version.relateds.any?
    self.current_version.relateds.each { |rel| rel.update_attribute(:current, false) }
  end

  version = self.versions.build(attributes)
  version.article_id = self.id
  version.writer_id = user.id
  version.version = self.current_version.version + 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

Set Default Values in the Database

```ruby
class AddRawDefaultToState < ActiveRecord::Migration
  def self.up
    change_column_default :article_versions, :state, "Raw"
  end

  def self.down
    change_column_default :article_versions, :state, nil
  end
end
```

Here is How We Get There

Fodder for Callbacks

```
def create_first_version!(attributes, user)
  version = self.versions.build(attributes)
  version.writer_id = user.id
  version.version = 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end

def create_version!(attributes, user)
  return create_first_version!(attributes, user) if self.versions.empty?

  # mark old related links as not current
  if self.current_version.relateds.any?
    self.current_version.relateds.each { |rel| rel.update_attribute(:current, false) }
  end

  version = self.versions.build(attributes)
  version.writer_id = user.id
  version.version = self.current_version.version + 1
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

A Callback on ArticleVersion

```
class ArticleVersion < ActiveRecord::Base

  before_validation_on_create :set_version_number

  private

  def set_version_number
    self.version = (article.current_version ?
article.current_version.version : 0) + 1
  end
```

Here is How We Get There

Now Code is Identical

```ruby
def create_first_version!(attributes, user)
  version = self.versions.build(attributes)
  version.writer_id = user.id
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end

def create_version!(attributes, user)
  return create_first_version!(attributes, user) if self.versions.empty?

  # mark old related links as not current
  if self.current_version.relateds.any?
    self.current_version.relateds.each { |rel| rel.update_attribute(:current, false) }
  end

  version = self.versions.build(attributes)
  version.writer_id = user.id
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

Identify Another Callback

```ruby
def create_version!(attributes, user)
  unless self.versions.empty?
    # mark old related links as not current
    if self.current_version.relateds.any?
      self.current_version.relateds.each do |rel|
        rel.update_attribute(:current, false)
      end
    end
  end

  version = self.versions.build(attributes)
  version.writer_id = user.id
  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

Expressive Callback Names

```
class ArticleVersion < ActiveRecord::Base

  before_validation_on_create :set_version_number
  before_create :mark_related_links_not_current

  private

  def set_version_number
    self.version = (article.current_version ?
article.current_version.version : 0) + 1
  end

  def mark_related_links_not_current
    unless article.versions.empty?
      # mark old related links as not current
      if article.current_version.relateds.any?
        article.current_version.relateds.each do |rel|
          rel.update_attribute(:current, false)
        end
      end
    end
  end
```

Here is How We Get There

Do What You Mean

```ruby
class ArticleVersion < ActiveRecord::Base

  before_validation_on_create :set_version_number
  before_create :mark_related_links_not_current

  private

  def set_version_number
    self.version = (article.current_version ?
article.current_version.version : 0) + 1
  end

  def mark_related_links_not_current
    unless article.versions.empty? ←
      if article.current_version.relateds.any?
        article.current_version.relateds.each do |rel|
          rel.update_attribute(:current, false)
        end
      end
    end
  end
```

Here is How We Get There

More Unnecessary Code

```
class ArticleVersion < ActiveRecord::Base

  before_validation_on_create :set_version_number
  before_create :mark_related_links_not_current

  private

  def set_version_number
    self.version = (article.current_version ?
article.current_version.version : 0) + 1
  end

  def mark_related_links_not_current
    if article.current_version
      if article.current_version.relateds.any?  ←
        article.current_version.relateds.each do |rel|
          rel.update_attribute(:current, false)
        end
      end
    end
  end
```

Here is How We Get There

Minor Law of Demeter Violation

```
class ArticleVersion < ActiveRecord::Base

  before_validation_on_create :set_version_number
  before_create :mark_related_links_not_current

  private

  def set_version_number
    self.version = (article.current_version ?
article.current_version.version : 0) + 1
  end

  def mark_related_links_not_current
    if article.current_version
      article.current_version.relateds.each do |rel|
        rel.update_attribute(:current, false)
      end
    end
  end
```

Here is How We Get There

Conditional Callbacks

```
class ArticleVersion < ActiveRecord::Base

  before_validation_on_create :set_version_number
  before_create :mark_related_links_not_current

  private

  def current_version
    article.current_version
  end

  def set_version_number
    self.version = (current_version ? current_version.version : 0) + 1
  end

  def mark_related_links_not_current
    if current_version  <------
      current_version.relateds.each do |rel|
        rel.update_attribute(:current, false)
      end
    end
  end
```

Here is How We Get There

Conditional Callback

```
class ArticleVersion < ActiveRecord::Base

  before_validation_on_create :set_version_number
  before_create :mark_related_links_not_current, :if => :current_version

  private

  def current_version
    article.current_version
  end

  def set_version_number
    self.version = (current_version ? current_version.version : 0) + 1
  end

  def mark_related_links_not_current
    current_version.relateds.each do |rel|
      rel.update_attribute(:current, false)
    end
  end
```

Here is How We Get There

The New create_version! Method

```ruby
def create_version!(attributes, user)
  version = self.versions.build(attributes)
  version.writer_id = user.id

  self.save!
  self.update_attribute(:current_version_id, version.id)
  version
end
```

Here is How We Get There

Callback to Update the Current Version

```
class ArticleVersion < ActiveRecord::Base

  before_validation_on_create :set_version_number
  before_create :mark_related_links_not_current, :if => :current_version
  after_create :set_current_version_on_article

  private

  def set_current_version_on_article
    article.update_attribute :current_version_id, self.id
  end
```

Here is How We Get There

The New create_version! Method

```
def create_version!(attributes, user)
  version = self.versions.build(attributes)
  version.writer_id = user.id

  self.save!
  version
end
```

Here is How We Get There

The Current Create Action

```
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication_id = @publication.id
    @article.reporter_id = current_user.id

    begin
      Article.transaction do
        @version = @article.create_version!(params[:version],
                                             current_user)
      end
    rescue ActiveRecord::RecordNotSaved, ActiveRecord::RecordInvalid
      index
      render :action => :index and return false
    end
    redirect_to article_path(@publication, @article)
  end
```

Here is How We Get There

```
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication_id = @publication.id
    @article.reporter_id = current_user.id
    @version = self.versions.build(attributes)
    @version.writer_id = current_user.id

    begin
      Article.transaction do
        @version = @article.create_version!(params[:version],
                                            current_user)
      end
    rescue ActiveRecord::RecordNotSaved, ActiveRecord::RecordInvalid
      index
      render :action => :index and return false
    end
    redirect_to article_path(@publication, @article)
  end
```

Here is How We Get There

The New EMPTY create_version! Method

```
def create_version!(attributes, user)
  self.save!
  version
end
```

Here is How We Get There

Remove the Transaction

```
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication_id = @publication.id
    @article.reporter_id = current_user.id
    @version = @article.versions.build(params[:version])
    @version.writer_id = current_user.id

    begin
      Article.transaction do
        @article.save!
      end
    rescue ActiveRecord::RecordNotSaved, ActiveRecord::RecordInvalid
      index
      render :action => :index and return false
    end
    redirect_to article_path(@publication, @article)
  end
```

Here is How We Get There

Change to the Non-Bang Save

```
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication_id = @publication.id
    @article.reporter_id = current_user.id
    @version = @article.versions.build(params[:version])
    @version.writer_id = current_user.id

    begin
      @article.save! ←
    rescue ActiveRecord::RecordNotSaved, ActiveRecord::RecordInvalid
      index
      render :action => :index and return false
    end
    redirect_to article_path(@publication, @article)
  end
```

Here is How We Get There

```ruby
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication_id = @publication.id
    @article.reporter_id = current_user.id
    @version = @article.versions.build(params[:version])
    @version.writer_id = current_user.id

    if @article.save
      redirect_to article_path(@publication, @article)
    else
      index
      render :action => :index
    end
  end
```

Here is How We Get There

The Final Create Action

```
class ArticlesController < ApplicationController

  def create
    @article = Article.new(params[:article])
    @article.publication = @publication
    @article.reporter = current_user
    @version = @article.versions.build(params[:version])
    @version.writer = current_user

    if @article.save
      redirect_to article_path(@publication, @article)
    else
      index
      render :action => :index
    end
  end
```

Phew!
What Did We Learn?

- Even a 15 Line Controller Action is too long.
- Exceptions should be exceptional.
- Your models should use callbacks to add complex behavior.

Too Much Domain Knowledge

```
if params[:send] and @version.state == "Edit"
  flash[:success] = "Article saved and ready for editing."
elsif params[:subedit]
  if @version.state == "Sub Check"
    flash[:success] = "Article sent back for editing."
  elsif @version.state == "Sub Edit"
    flash[:success] = "Article sent to sub editors."
  end
elsif params[:back]
  if @version.state == "Edit Check"
    flash[:success] = "Article sent back to reporter for further work."
  end
elsif @version.state == "Publish Ready"
  flash[:success] = "Article has been marked for publishing at a later date."
elsif @version.state == "Published" and old_state != "Published"
  flash[:success] = "Article saved and published to live server."
elsif @version.state == "Published"
  flash[:success] = "Article has been republished."
end
```

Too Much Domain Knowledge

```
if params[:send] and @version.edit?
  flash[:success] = "Article saved and ready for editing."
elsif params[:subedit]
  if @version.sub_check?
    flash[:success] = "Article sent back for editing."
  elsif @version.sub_edit?
    flash[:success] = "Article sent to sub editors."
  end
elsif params[:back]
  if @version.edit_check?
    flash[:success] = "Article sent back to reporter for further work."
  end
elsif @version.publish_ready?
  flash[:success] = "Article has been marked for publishing at a later date."
elsif @version.published? and old_state != ArticleVersion::STATES[:published]
  flash[:success] = "Article saved and published to live server."
elsif @version.published?
  flash[:success] = "Article has been republished."
end
```

Too Much Domain Knowledge

```
class ArticleVersion < ActiveRecord::Base

  STATES = { :edit => 'Edit',
             :edit_check => 'Edit Check',
             :sub_edit => 'Sub Edit',
             :publish_ready => 'Publish Ready',
             :published => 'Published' }

  STATES.each do |key, value|
    define_method "#{key}?", {
      self.state == "#{value}"
    }
  end
```

What Did We Learn?

- Tackle large refactorings iteratively
- Push as much business logic into the model as possible
- In lots of code, simple DRY principles will lead a lot of refactoring

Finders in the Controller

```ruby
class ArticlesController < ApplicationController
  def index
    @articles = Article.find_all_by_state(Article::STATES[:published],
                                             :order => "created_at DESC")
  end
end
```

Move it into the Model

```
class ArticlesController < ApplicationController
  def index
    @articles = Article.published
  end
end

class Article < ActiveRecord::Base
  def self.published
    find_all_by_state(STATES[:published],
                           :order => "created_at DESC")
  end
end
```

Move it into the Model

named_scope

```
class ArticlesController < ApplicationController
  def index
    @articles = Article.published.ordered
  end
end

class Article < ActiveRecord::Base
  named_scope :published, :conditions => {:state =>
                                          STATES[:published]}
  named_scope :ordered, :order => "created_at DESC"
end
```

Common Domain Pattern: User Roles

User Roles

The User Model

```
class User < ActiveRecord::Base
  has_and_belongs_to_many :roles, :uniq => true

  def has_role?(role_in_question)
    self.roles.find(:first, :conditions => { :name => role_in_question }) ? true : false
  end

  def has_roles?(roles_in_question)
    roles_in_question = self.roles.find(:all, :conditions => ["name in (?)", roles_in_question])
    roles_in_question.length > 0
  end

  def can_post?
    self.has_roles?(['admin', 'editor', 'associate editor'])
  end

  def can_review_posts?
    self.has_roles?(['admin', 'editor', 'associate editor'])
  end

  def can_edit_content?
    self.has_roles?(['admin', 'editor', 'associate editor'])
  end

  def can_edit_post?(post)
    self == post.user || self.has_roles?(['admin', 'editor', 'associate editor'])
  end
end
```

User Roles

The Role Model

```
class Role < ActiveRecord::Base

  has_and_belongs_to_many :users

  validates_presence_of :name
  validates_uniqueness_of :name

  def name=(value)
    write_attribute("name", value.downcase)
  end

  def self.[](name) # Get a role quickly by using: Role[:admin]
    self.find(:first, :conditions => ["name = ?", name.id2name])
  end

  def add_user(user)
    self.users << user
  end

  def delete_user(user)
    self.users.delete(user)
  end
end
```

User Roles

Thoughtless Code

```ruby
class User < ActiveRecord::Base
  has_and_belongs_to_many :roles, :uniq => true

  def has_role?(role_in_question)
    self.roles.find(:first, :conditions => { :name => role_in_question }) ? true : false
  end

  def has_roles?(roles_in_question)
    roles_in_question = self.roles.find(:all, :conditions => ["name in (?)", roles_in_question])
    roles_in_question.length > 0
  end

  def can_post?
    self.has_roles?(['admin', 'editor', 'associate editor'])
  end

  def can_review_posts?
    self.has_roles?(['admin', 'editor', 'associate editor'])
  end

  def can_edit_content?
    self.has_roles?(['admin', 'editor', 'associate editor'])
  end

  def can_edit_post?(post)
    self == post.user || self.has_roles?(['admin', 'editor', 'associate editor'])
  end
end
```

User Roles

What We've Done

```ruby
class User < ActiveRecord::Base
end
```

User Roles

What We've Done

```
class User < ActiveRecord::Base
end
```

- Get rid of the role model
- Make an admin, editor, and writer boolean on the user model
- Active Record gives us nice admin?, editor?, writer? methods on User, and the UI for giving users roles is straightforward

User Roles

Dealing with More Roles in the Future

- If down the road, you need one more role, add one more boolean
- Two more roles: add the Role model back, but don’t use has and belongs to many

What Did We Learn?

- Don't Build Beyond Requirements
- Don't Jump To a New Model Prematurely
- No UI to Add == No Model

Bad Code Happens to Good People

This Code Worked

Refactoring != Bug Fixing

Always Be Refactoring

Questions?

Chad Pytel
cpytel@thoughtbot.com

Tammer Saleh
tsaleh@thoughtbot.com

Recommend Us On Working With Rails
http://www.workingwithrails.com/person/5509
http://www.workingwithrails.com/person/7844