@ddemaree
I like pie.

typekit
The web’s best fonts.

Adobe
We make Photoshop™.
REFACTORIZING

vs

PREFACTORIZING
Don’t Repeat Yourself vs Too much abstraction
THE PROBLEM

It's really easy to convince yourself that today's code will be more complex in the future, and design for that.
THE PROBLEM

It’s really easy to convince yourself that today’s code will be more complex in the future, and design for that.
When is it time to refactor?
When is it *not* time to refactor?
Rules of thumb
The Rule of Three

“Code can be copied once, but that when the same code is used three times, it should be extracted into a new procedure.”

def foo
    # Does some stuff ...
    if @user.status == :active
        @user.status = :inactive
        @user.save!
    end
end
def foo
    # Does some stuff ... 
    if @user.status == :active
        @user.status = :inactive
        @user.save!
    end
end

def bar
    # Does some other stuff ... 
    if @user.status == :active
        @user.status = :inactive
        @user.save!
    end
end
def foo
    # Does some stuff ...
    if @user.status == :active
        @user.status = :inactive
        @user.save!
    end
end

def bar
    # Does some other stuff ...
    if @user.status == :active
        @user.status = :inactive
        @user.save!
    end
end

def baz
    # Does yet other stuff ...
    if @user.status == :active
        @user.status = :inactive
        @user.save!
    end
end
def deactivate_user(user)
    if user.status == :active
        user.status = :inactive
        user.save!
    end
end

def foo
    # Does some stuff ...
    deactivate_user(@user)
end

def bar
    # Does some other stuff ...
    deactivate_user(@user)
end

def baz
    # Does yet other stuff ...
    deactivate_user(@user)
end
def deactivate_user(user)
    if user.status == :active
        user.status = :inactive
        user.save!
    end
end

def foo
    # Does some stuff ...
    deactivate_user(@user)
end

def bar
    # Does some other stuff ...
    deactivate_user(@user)
end

def baz
    # Does yet other stuff ...
    deactivate_user(@user)
end

This is better code

BUT

Resist the temptation to do this until you really have repeated yourself three times
Three of a Kind

Three or more related methods/symbols can be meaningfully grouped together
class User < ActiveRecord::Base

  def adobe_profile
    AdobeId::Profile.new(self.adobe_profile_id)
  end

  def connected_to_adobe_profile?
    !self.adobe_profile_id.nil?
  end

end
class User < ActiveRecord::Base

  def adobe_profile
    AdobeId::Profile.new(self.adobe_profile_id)
  end

  def connected_to_adobe_profile?
    !self.adobe_profile_id.nil?
  end

  def connect_to_adobe_profile(profile_obj)
    self.adobe_profile_id = profile_obj.id
    self.email_unique = false
    save
  end

end
class User < ActiveRecord::Base

  def adobe_profile
    AdobeId::Profile.new(self.adobe_profile_id)
  end

  def connected_to_adobe_profile?
    !self.adobe_profile_id.nil?
  end

  def connect_to_adobe_profile(profile_obj)
    self.adobe_profile_id = profile_obj.id
    self.email_unique = false
    save
  end

  def disconnect_adobe_profile
    self.adobe_profile_id = nil
    self.email_unique = true
    save
  end

end
module UserAdobeProfile

  def adobe_profile
    AdobeId::Profile.new(self.adobe_profile_id)
  end

  def connected_to_adobe_profile?
    !self.adobe_profile_id.nil?
  end

  # Anything else pertaining to users' Adobe profiles ...

end

class User < ActiveRecord::Base
  include UserAdobeProfile
end
No Assumptions

You don’t know what you know until you know it.
JEFF ATWOOD:

“I believe writing a truly reusable class is an order of magnitude harder than writing a single use class. Sometimes the right thing to do is resist the urge to write "general purpose" solutions.”

It’s awesome.
✉ ddemaree@adobe.com
🐦 @ddemaree
‡ log.demaree.me
It will not surprise you to learn that

WE ARE HIRING


or just come talk to me