Rails 最佳實務

http://ihower.tw
2016/4

Agenda

- Concept: What's good code?
- Move Code from Controller to Model
- Model best practices
- Controller best practices
- View best practices
- Next

Warning! you should have testing before modify!

修改重構程式前，

應有自動化測試

確保程式於修改後行為不變

Best Practice Lesson 0:

Concepts

Why best practices?

- Large & complicated application
  日漸複雜的程式
- Team & different coding style
  團隊開發

Your code become...

- 僵硬 (Rigidity)：難以修改，每改一處牽一髮動全身
- 脆弱 (Fragility)：一旦修改，別的無關地方也炸到
- 固定 (Immobility)：難以分解，讓程式再重用
- 黏滯 (Viscosity)：彈性不夠，把事情做對比做錯還難
- 不需要的複雜度 (Needless Complexity)：過度設計沒直接好處的基礎設施
- 不需要的重複 (Needless Repetition)：相同概念的程式碼被複製貼上重複使用
- 晦澀 (Opacity)：難以閱讀，無法了解意圖

出自 Agile Software Development: Principles, Patterns, and Practices 一書

We need good code:

我們需要好程式

What's Good code?

- Readability 易讀，容易了解
- Flexibility 彈性，容易擴充
- Effective 效率，撰碼快速
- Maintainability 維護性，容易找到問題
- Consistency 一致性，循慣例無需死背
- Testability 可測性，元件獨立容易測試

So, What we can do?

來開始學幾招吧

Best Practice Lesson:

Move code from Controller to Model

action code 超過15行請注意

http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model

Before

1.Move finder to scope

```
class PostsController < ApplicationController

  def index
    @public_posts = Post.where(:state => 'public' ).limit(10)
                        .order('created_at desc')
    @draft_posts  = Post.where(:state => ‘draft').limit(10)
                        .order('created_at desc')
  end

end
```

After

1.Move finder to scope

```
class UsersController < ApplicationController

  def index
    @published_post = Post.published
    @draft_post = Post.draft
  end

  end

  class Post < ActiveRecord::Base

    scope :published, -> { where(:state => ‘published').limit(10)
                           .order('created_at desc') }
    scope :draft, -> { where(:state => ‘draft').limit(10)
                       .order('created_at desc') }

  end
```

Before

2. Use model association

```
class PostsController < ApplicationController

  def create
    @post = Post.new(params[:post])
    @post.user_id = current_user.id
    @post.save
  end

end
```

After

2. Use model association

```
class PostsController < ApplicationController

  def create
    @post = current_user.posts.build(params[:post])
    @post.save
  end

end

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

3. Use scope access

不必要的權限檢查

Before

```
class PostsController < ApplicationController

  def edit
    @post = Post.find(params[:id)

    if @post.user != current_user
      flash[:warning] = 'Access denied'
      redirect_to posts_url
    end

  end

end
```

3. Use scope access

找不到自然會丟例外

After

```
class PostsController < ApplicationController

  def edit
    # raise RecordNotFound exception (404 error) if not found
    @post = current_user.posts.find(params[:id)
  end

end
```

Before

4. Add model virtual attribute

```
<% form_for @user do |f| %>
    <%= text_filed_tag :full_name %>
<% end %>

class UsersController < ApplicationController
  def create
    @user = User.new(params[:user)
    @user.first_name = params[:full_name].split(' ', 2).first
    @user.last_name = params[:full_name].split(' ', 2).last
    @user.save
  end

end
```

After

4. Add model virtual attribute

```ruby
class User < ActiveRecord::Base

  def full_name
    [first_name, last_name].join(' ')
  end

  def full_name=(name)
    split = name.split(' ', 2)
    self.first_name = split.first
    self.last_name = split.last
  end

end
```

example code from http://railscasts.com/episodes/16-virtual-attributes

After

```erb
<% form_for @user do |f| %>
  <%= f.text_field :full_name %>
<% end %>
```

```ruby
class UsersController < ApplicationController

  def create
    @user = User.create(params[:user)
  end

end
```

example code from http://railscasts.com/episodes/16-virtual-attributes

5. Use model callback

Before

```
<% form_for @post do |f| %>
  <%= f.text_field :content %>
  <%= check_box_tag 'auto_tagging' %>
<% end %>

class PostController < ApplicationController

  def create
    @post = Post.new(params[:post])

    if params[:auto_tagging] == '1'
      @post.tags = AsiaSearch.generate_tags(@post.content)
    else
      @post.tags = ""
    end

    @post.save
  end

end
```

After

5. Use model callback

```
class Post < ActiveRecord::Base

  attr_accessor :auto_tagging
  before_save :generate_taggings

  private

    def generate_taggings
      return unless auto_tagging == '1'
      self.tags = Asia.search(self.content)
    end

end
```

After

```
<% form_for :note, ... do |f| %>
  <%= f.text_field :content %>
  <%= f.check_box :auto_tagging %>
<% end
```

```
class PostController < ApplicationController
  def create
    @post = Post.new(params[:post])
    @post.save
  end

end
```

6. Replace Complex Creation with Factory Method

Before

```
class InvoiceController < ApplicationController

  def create
    @invoice = Invoice.new(params[:invoice])
    @invoice.address = current_user.address
    @invoice.phone = current_user.phone
    @invoice.vip = ( @invoice.amount > 1000 )

    if Time.now.day > 15
      @invoice.delivery_time = Time.now + 2.month
    else
      @invoice.delivery_time = Time.now + 1.month
    end

    @invoice.save
  end

end
```

After

6. Replace Complex Creation with Factory Method

```
class Invoice < ActiveRecord::Base

  def self.new_by_user(params, user)
    invoice = self.new(params)
    invoice.address = user.address
    invoice.phone = user.phone
    invoice.vip = ( invoice.amount > 1000 )

    if Time.now.day > 15
      invoice.delivery_time = Time.now + 2.month
    else
      invoice.delivery_time = Time.now + 1.month
    end
    return invoice
  end

end
```

After

```
class InvoiceController < ApplicationController
  def create
    @invoice = Invoice.new_by_user(params[:invoice], current_user)
    @invoice.save
  end
end
```

Before

7. Move Logic into the Model

```
class PostController < ApplicationController

  def publish
    @post = Post.find(params[:id])
    @post.is_published = true
    @post.approved_by = current_user
    if @post.create_at > Time.now - 7.days
      @post.popular = 100
    else
      @post.popular = 0
    end
    @post.save

    redirect_to post_url(@post)
  end

end
```

7. Move Model Logic into the Model

After

```
class Post < ActiveRecord::Base

  def publish(user)
    self.is_published = true
    self.approved_by = user
    if self.create_at > Time.now-7.days
      self.popular = 100
    else
      self.popular = 0
    end
  end

end
```

After

```
class PostController < ApplicationController

  def publish
    @post = Post.find(params[:id])
    @post.publish(current_user)
    @post.save

    redirect_to post_url(@post)
  end

end
```

Best Practice Lesson:

Model

Before

1. the Law of Demeter

```
class Invoice < ActiveRecord::Base
  belongs_to :user
end

<%= @invoice.user.name %>
<%= @invoice.user.address %>
<%= @invoice.user.cellphone %>
```

After

1. the Law of Demeter

```
class Invoice < ActiveRecord::Base
  belongs_to :user
  delegate :name, :address, :cellphone, :to => :user,
                                        :prefix => true,
                                       :allow_nil => true
end
```

```
<%= @invoice.user_name %>
<%= @invoice.user_address %>
<%= @invoice.user_cellphone %>
```

2. DRY: Metaprogramming

Before

```
class Post < ActiveRecord::Base

  validate_inclusion_of :status, :in => ['draft', 'published', 'spam']

  def self.all_draft
    where( :status => 'draft'  )
  end

  def self.all_published
    where( :status => 'published' )
  end

  def self.all_spam
    where( :status => 'spam' )
  end

  def draft?
    self.status == 'draft'
  end

  def published?
    self.status == 'published'
  end

  def spam?
    self.status == 'spam'
  end

end
```

2. DRY: Metaprogramming

After

```
class Post < ActiveRecord::Base

  STATUSES = ['draft', 'published', 'spam']
  validate_inclusion_of :status, :in => STATUSES

  class << self
    STATUSES.each do |status_name|
      define_method "all_#{status}" do
        where( :status => status_name )
      end
    end
  end

  STATUSES.each do |status_name|
    define_method "#{status_name}?" do
      self.status == status_name
    end
  end

end
```

Breaking Up Models

幫 Model 減重

Before

3. Extract into Module

```ruby
class User < ActiveRecord::Base

  validates_presence_of :cellphone
  before_save :parse_cellphone

  def parse_cellphone
    # do something
  end

  def self.foobar
    # do something
  end

end
```

After

```ruby
# /app/models/concerns/has_cellphone.rb
module HasCellphone

  def self.included(base)
    base.validates_presence_of :cellphone
    base.before_save :parse_cellphone

    base.send(:extend, ClassMethods)
  end

  def parse_cellphone
    # do something
  end

  module ClassMethods
    def foobar
      # do something
    end
  end

end
```

Or use http://api.rubyonrails.org/classes/ActiveSupport/Concern.html

After

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

4. save v.s. save!

- 有搭配 if … else 情況做處理用 save。這表示你會處理儲存失敗流程
- 沒有的話，用 save! 驚嘆號版本。這表示你認為 99% 應該會存成功，懶得處理存不成功。如果出錯會有例外通知

Best Practice Lesson:

Migration

Before

1. Isolating Seed Data

```
class CreateRoles < ActiveRecord::Migration
  def self.up
    create_table "roles", :force => true do |t|
      t.string :name
    end

    ["admin", "author", "editor","account"].each do |name|
      Role.create!(:name => name)
    end
  end

  def self.down
    drop_table "roles"
  end
end
```

After

1. Isolating Seed Data

```
# /db/seeds.rb
["admin", "author", "editor","account"].each do |name|
  Role.create!(:name => name)
end
```

```
rake db:seed
```

Before

2. Always add DB index

```
class CreateComments < ActiveRecord::Migration
  def self.up
    create_table "comments", :force => true do |t|
      t.string :content
      t.integer :post_id
      t.integer :user_id
    end
  end

  def self.down
    drop_table "comments"
  end
end
```

After

2. Always add DB index

```
class CreateComments < ActiveRecord::Migration
  def self.up
    create_table "comments", :force => true do |t|
      t.string :content
      t.integer :post_id
      t.integer :user_id
    end

    add_index :comments, :post_id
    add_index :comments, :user_id
  end

  def self.down
    drop_table "comments"
  end
end
```

Best Practice Lesson:

Controller

1. Use before_action

Before

```
class PostController < ApplicationController

  def show
    @post = current_user.posts.find(params[:id]
  end

  def edit
    @post = current_user.posts.find(params[:id]
  end

  def update
    @post = current_user.posts.find(params[:id]
    @post.update_attributes(params[:post])
  end

  def destroy
    @post = current_user.posts.find(params[:id]
    @post.destroy
  end

end
```

1. Use before_action

After

```
class PostController < ApplicationController

  before_action :find_post, :only => [:show, :edit, :update, :destroy]

  def update
    @post.update_attributes(params[:post])
  end

  def destroy
    @post.destroy
  end

  protected

  def find_post
    @post = current_user.posts.find(params[:id])
  end

end
```

http://jeromedalbert.com/how-dhh-organizes-his-rails-controllers/

Best Practice Lesson:

View

最重要的守則：

Never logic code in Views

1. Move code into controller

Before

```
<% @posts = Post.all %>
<% @posts.each do |post| %>
 <%=h post.title %>
 <%=h post.content %>
<% end %>
```

After

```
class PostsController < ApplicationController

  def index
    @posts = Post.all
  end

end
```

2. Move code into model

Before

```
<% if current_user && (current_user == @post.user ||
                       @post.editors.include?(current_user) %>
  <%= link_to 'Edit this post', edit_post_url(@post) %>
<% end %>
```

After

```
<% if @post.editable_by?(current_user) %>
  <%= link_to 'Edit this post', edit_post_url(@post) %>
<% end %>
```

```
class Post < ActiveRecord::Base
  def ediable_by?(user)
    user && ( user == self.user || self.editors.include?(user)
  end
end
```

3. Move code into helper

Before

```
<%= select_tag :state, options_for_select( [[t(:draft),"draft" ],
                                            [t(:published),"published"]],
                                           params[:default_state] ) %>
```

After

```
<%= select_tag :state, options_for_post_state(params[:default_state]) %>

# /app/helpers/posts_helper.rb
def options_for_post_state(default_state)
  options_for_select( [[t(:draft),"draft" ],[t(:published),"published"]],
                      default_state )
end
```

4. Replace instance variable with local variable

```ruby
class Post < ApplicationController
  def show
    @post = Post.find(params[:id)
  end
end
```

Before

```
<%= render :partial => "sidebar" %>
```

After

```
<%= render :partial => "sidebar", :locals => { :post => @post } %>
```

6. Organize Helper files

Before

```
# app/helpers/user_posts_helper.rb
# app/helpers/author_posts_helper.rb
# app/helpers/editor_posts_helper.rb
# app/helpers/admin_posts_helper.rb
```

After

```
# app/helpers/posts_helper.rb
```

7. Learn Rails Helpers

- Learn content_for and yield
- Learn how to pass block parameter in helper
  - my slide about helper: http://www.slideshare.net/ihower/building-web-interface-on-rails
- Read Rails helpers source code
  - /actionpack-x.y.z/action_view/helpers/*

View Component

改善 partial template 封裝問題

- http://cells.rubyforge.org/
- https://github.com/apotonick/cells
- http://blog.xdite.net/posts/2013/09/01/cells-partial-cache-1

More lesson?

Coding Style

http://guides.ruby.tw/ruby-rails-style-guides/

- 靜態程式碼分析工具 (static code analyzer)
  - https://github.com/bbatsov/rubocop
  - https://github.com/troessner/reek
  - https://github.com/railsbp/rails_best_practices
  - 第三方服務 https://houndci.com/
  - 第三方服務 https://codeclimate.com
- Coding style 是面試看 code 的第一印象，很容易可以看出你是不是初學者，常不常寫 code。
  - 可讀的變數命名、適當縮排和空格

Next decompose fat model!
(其中 service object 最常用)

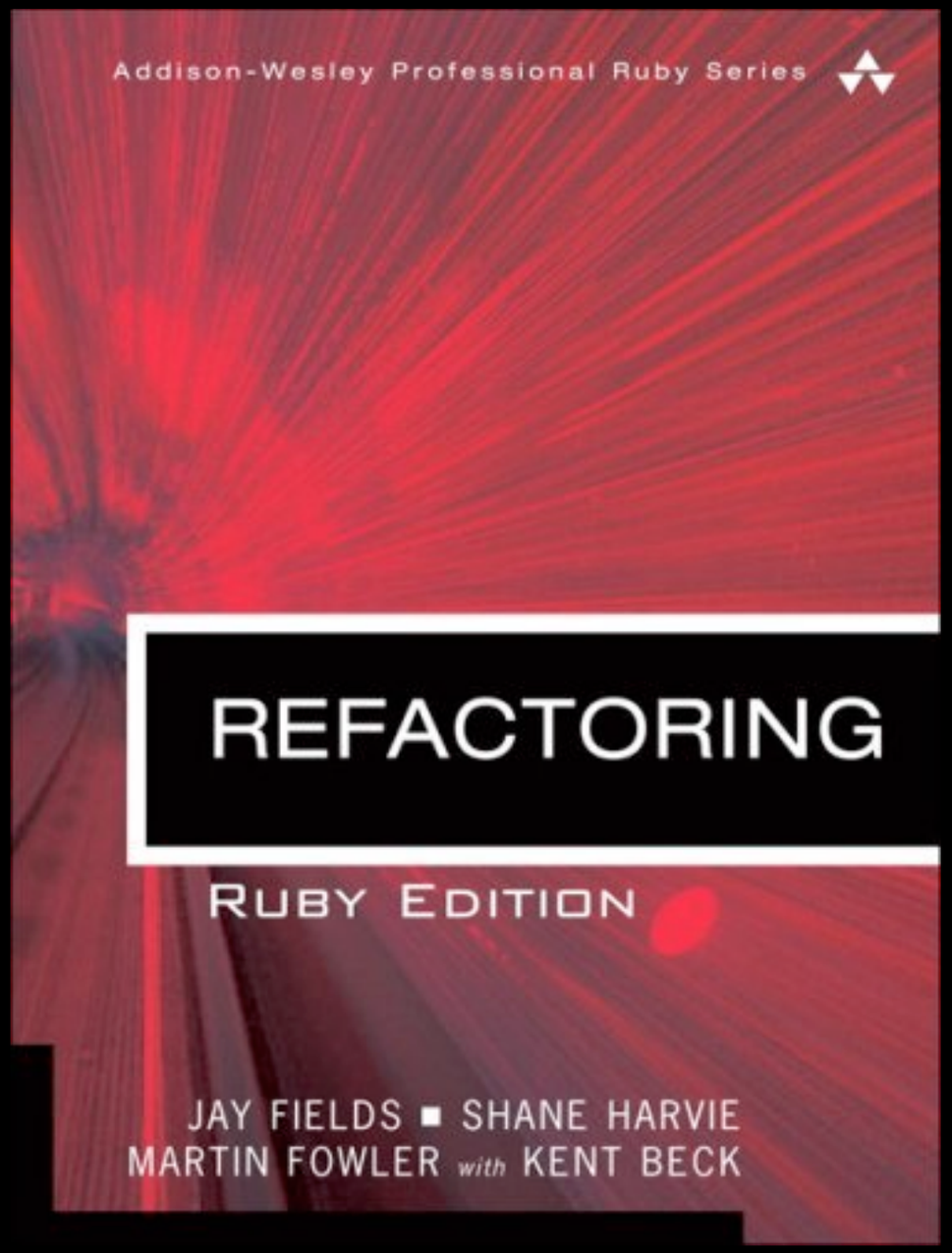
- http://blog.codeclimate.com/blog/2012/10/17/7-ways-to-decompose-fat-activerecord-models/
- http://pivotallabs.com/object-oriented-rails-writing-better-controllers/
- https://www.viget.com/articles/slimming-down-your-models-and-controllers

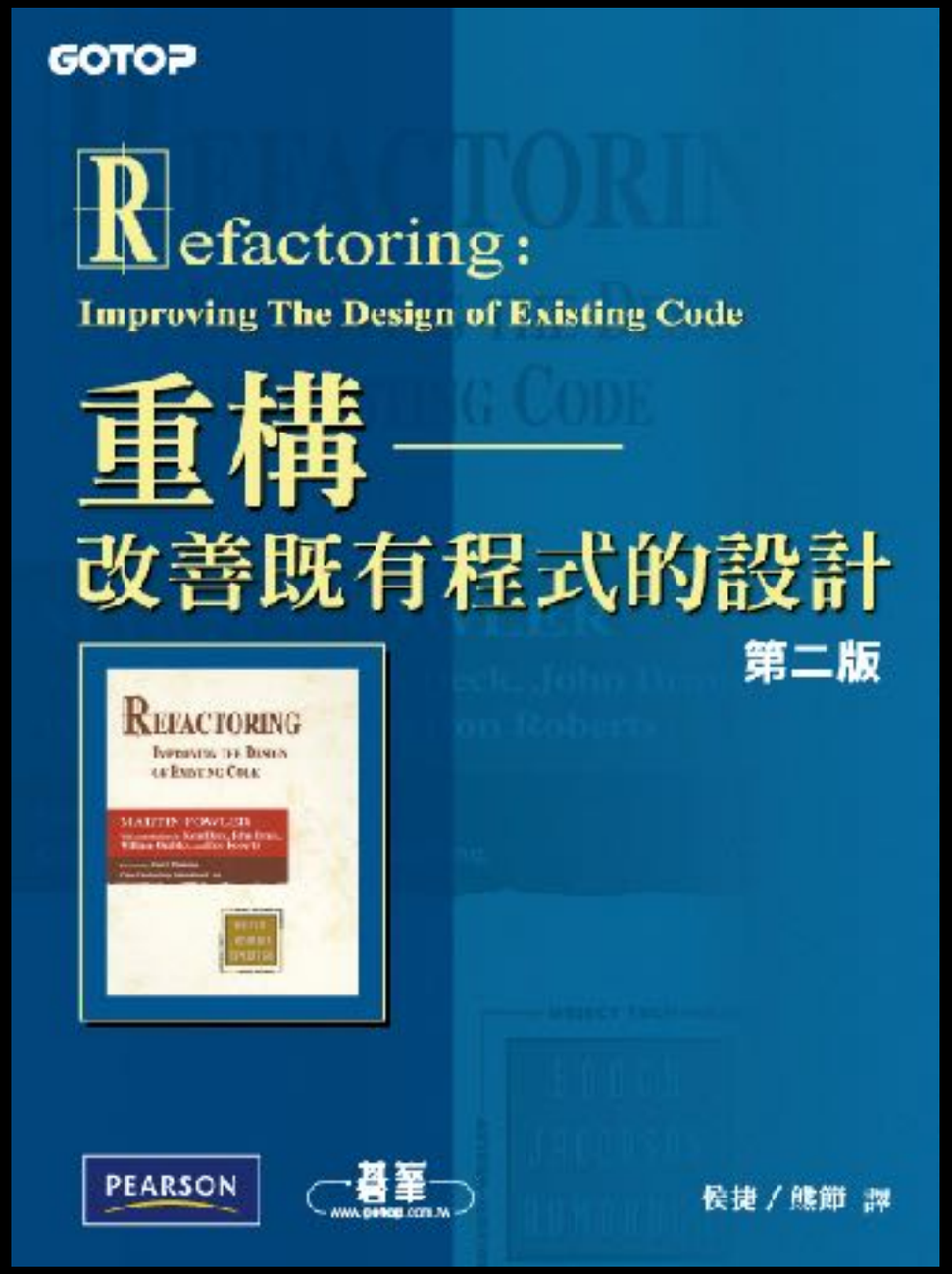
Kent Beck 的四個簡單程式設計原則

- Pass All Tests 通過全部測試
- Reveals Intent (Self-Documenting Code) 程式能夠表達出意圖
  - Composed Method
- No Duplication (DRY) 不重複
- Has no superfluous parts (Minimizes the number of classes and methods) 不多餘

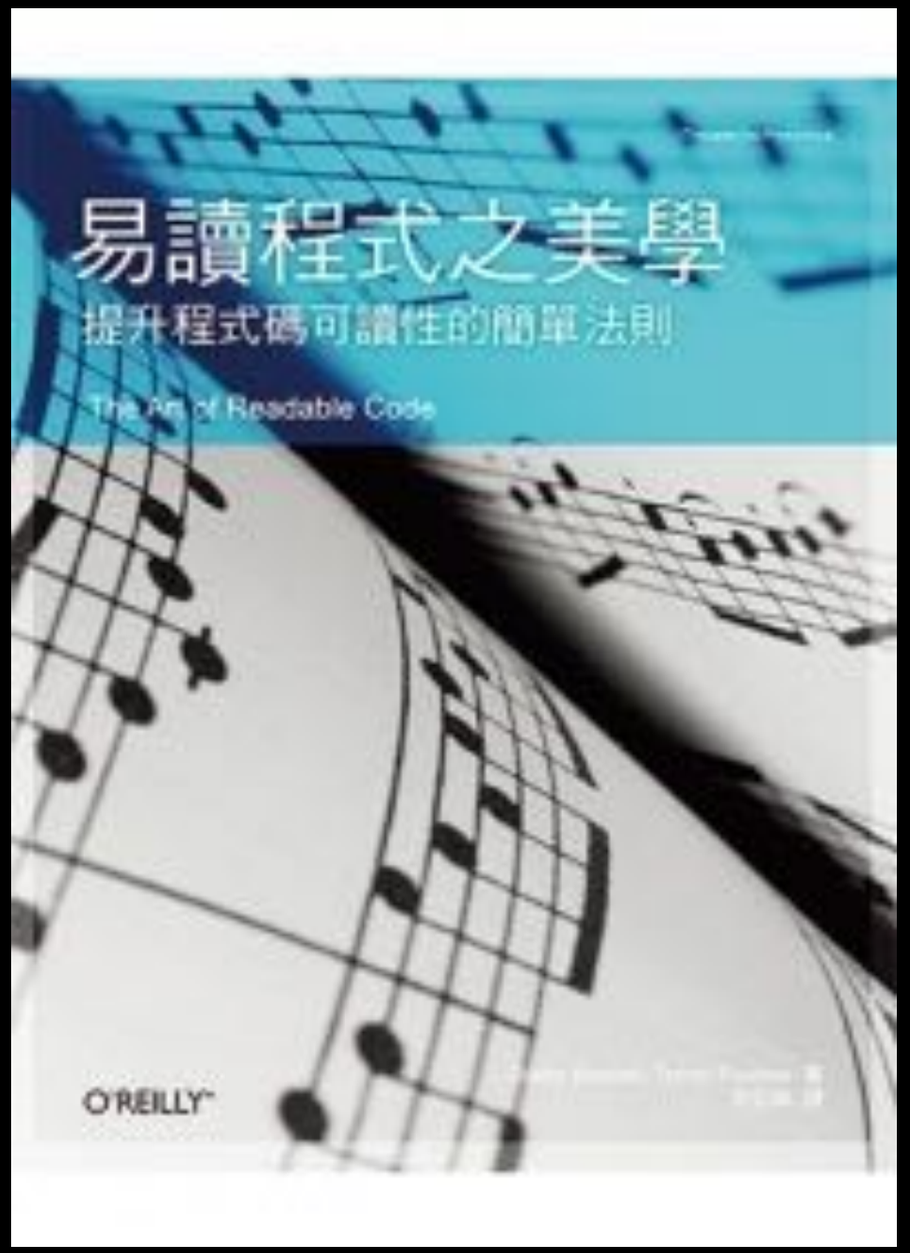
http://ihower.tw/blog/archives/7181

Books?

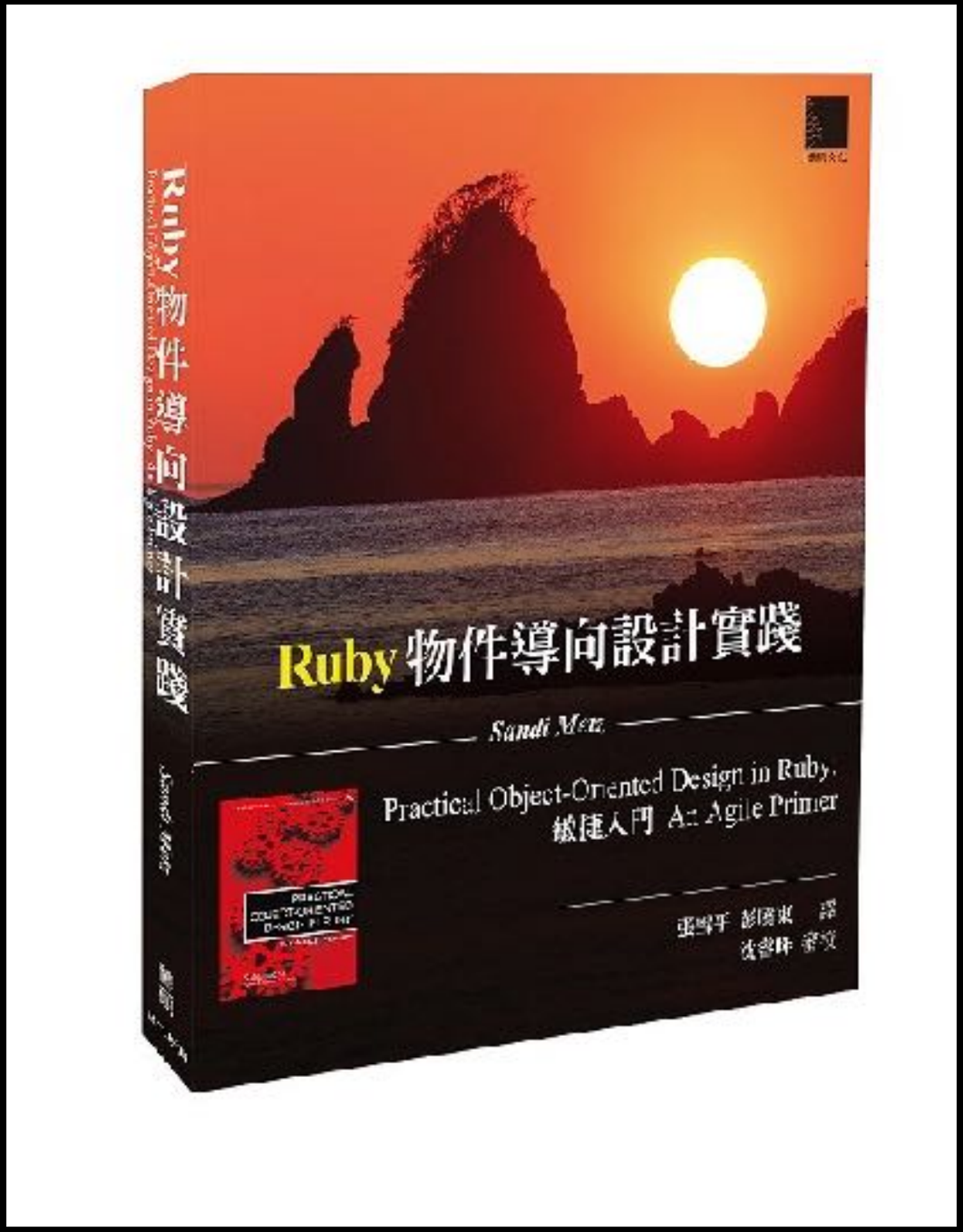

Addison-Wesley Professional Ruby Series
REFACTORING
RUBY EDITION
JAY FIELDS ■ SHANE HARVIE
MARTIN FOWLER with KENT BECK


GOTOP
Refactoring:
Improving The Design of Existing Code
重構——
改善既有程式的設計
第二版
REFACTORING
PEARSON
碁峰
www.gotop.com.tw
侯捷／熊節 譯

The Art of Readable Code



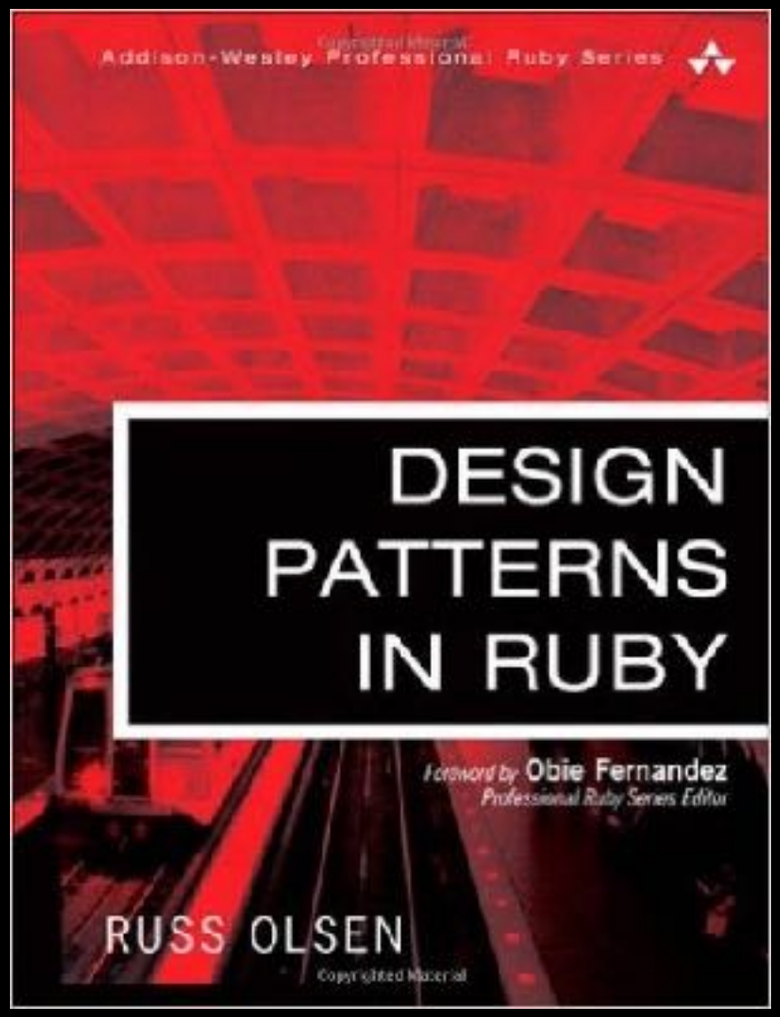
物件導向設計原則


Ruby物件導向設計實踐
Ruby物件導向設計實踐
Sandi Metz
Practical Object-Oriented Design in Ruby:
An Agile Primer
敏捷入門
張雪平 彭國東 譯
沈雪峰 審定

SOLID

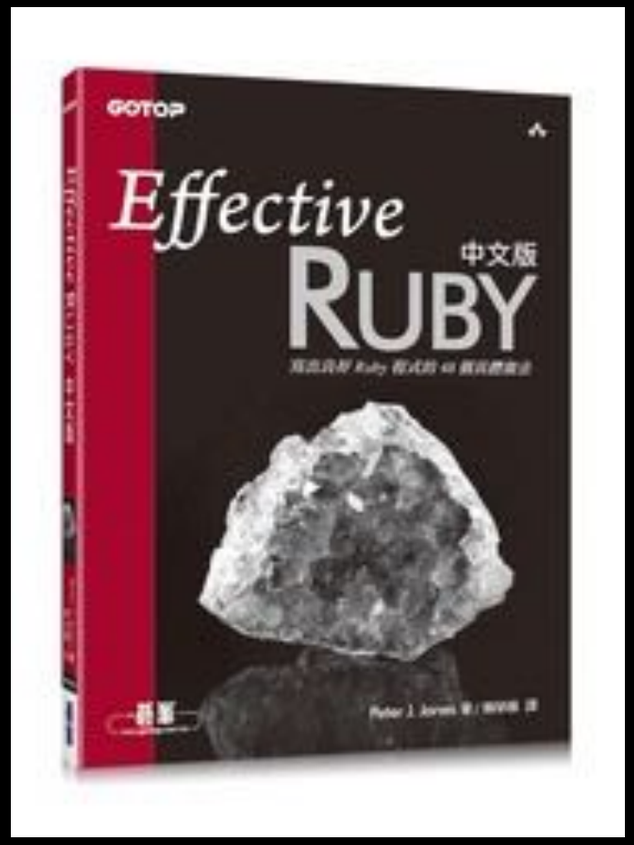
https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

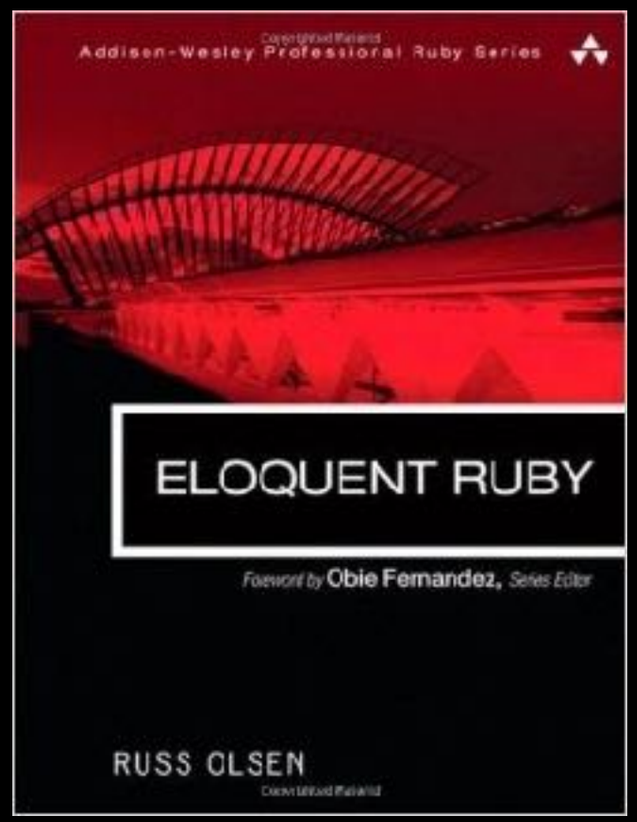
設計模式

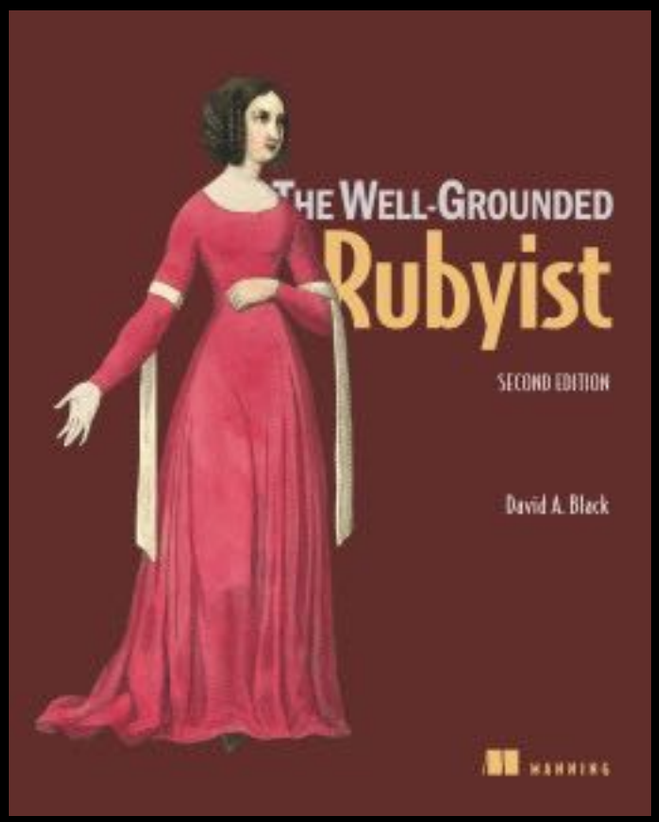

Addison-Wesley Professional Ruby Series
DESIGN PATTERNS IN RUBY
Foreword by Obie Fernandez
Professional Ruby Series Editor
RUSS OLSEN
Copyrighted Material


Head First Design Patterns
深入淺出
設計模式
O'REILLY

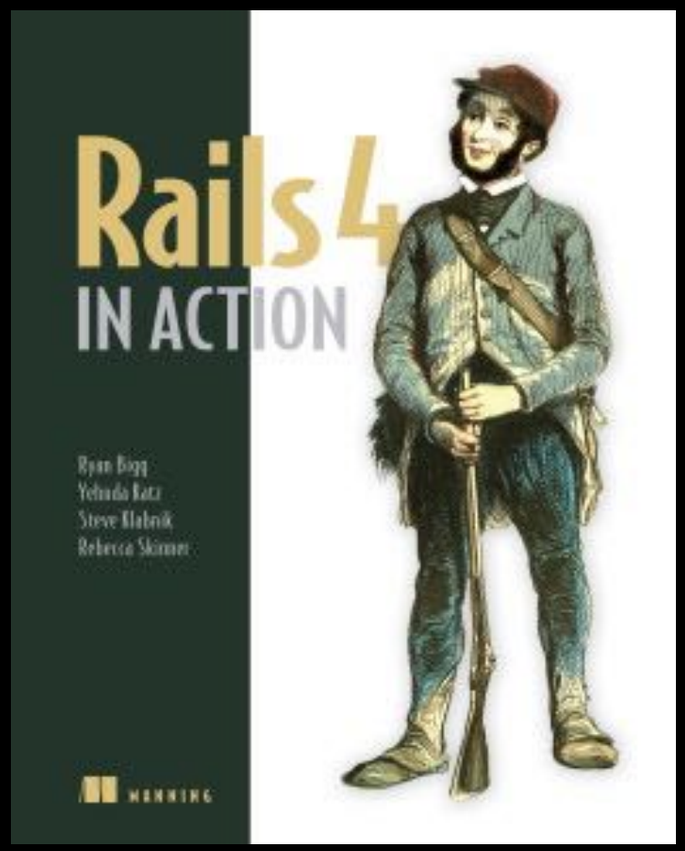
More Ruby books

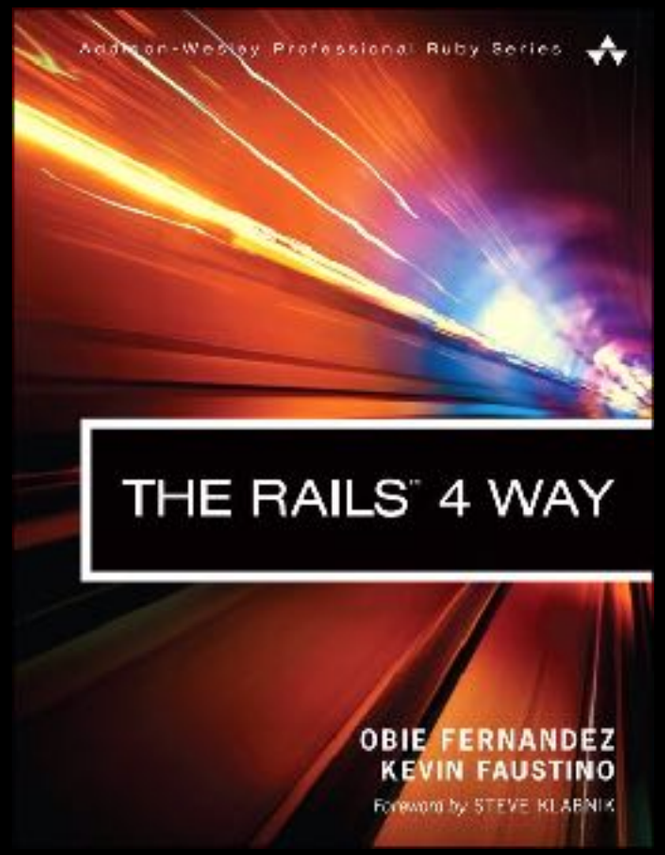


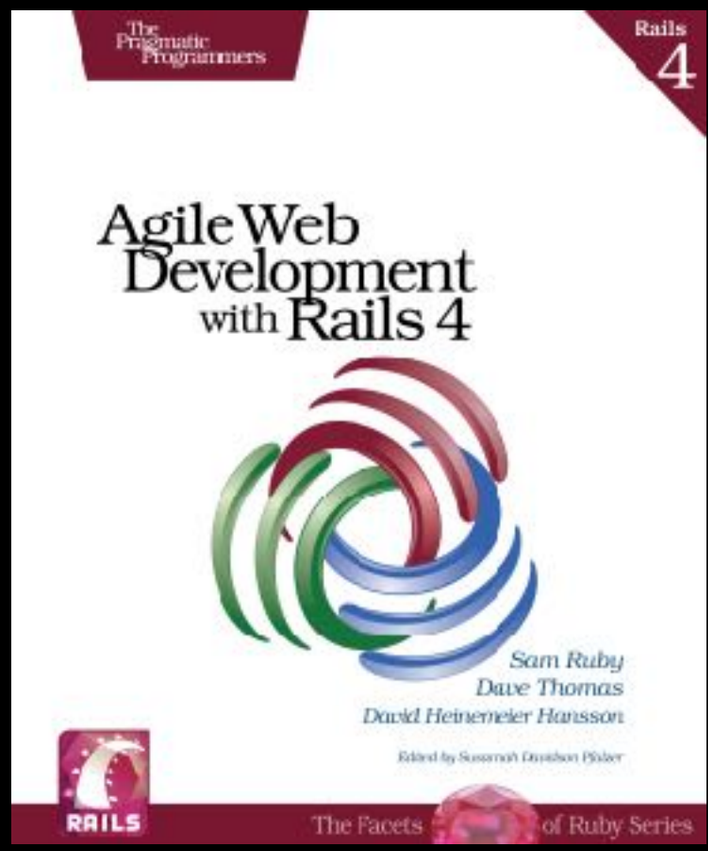




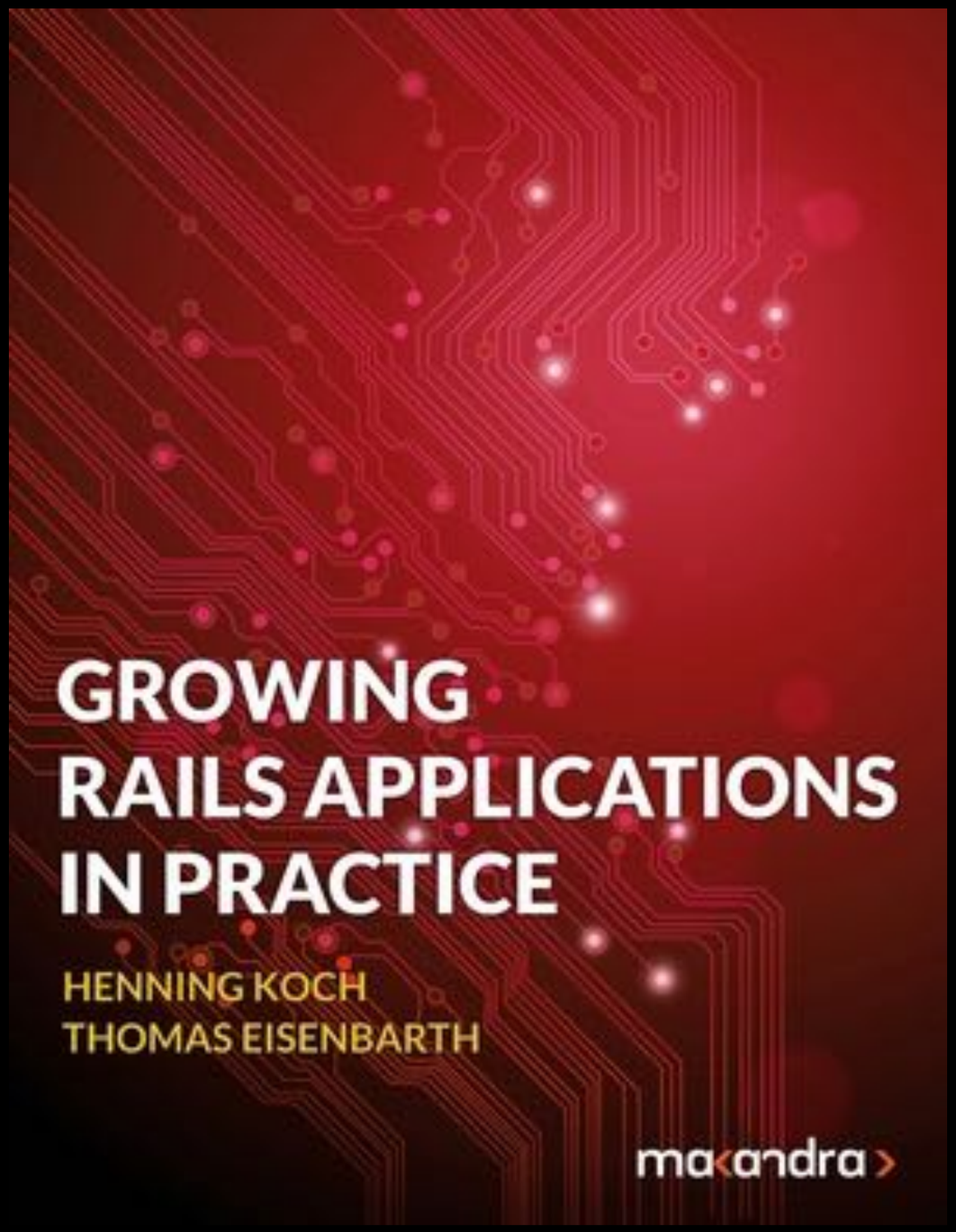
More Rails books







https://leanpub.com/growing-rails



Reference:

參考網頁:

http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model
http://www.matthewpaulmoore.com/ruby-on-rails-code-quality-checklist
http://www.chadfowler.com/2009/4/1/20-rails-development-no-no-s

參考資料:

Pragmatic Patterns of Ruby on Rails 大場寧子
Advanced Active Record Techniques Best Practice Refactoring Chad Pytel
Refactoring Your Rails Application RailsConf 2008
The Worst Rails Code You've Ever Seen Obie Fernandez
Mastering Rails Forms screencasts with Ryan Bates

參考書籍:

Agile Software Development: Principles, Patterns, and Practices
AWDwR 3rd
The Rails Way 2nd.
Advanced Rails Recipes
Refactoring Ruby Edition
Ruby Best Practices
Enterprise Rails
Rails Antipatterns
Rails Rescue Handbook
Code Review (PeepCode)
Plugin Patterns (PeepCode)

請多指教

Thank you.