

Order and Chaos: The Rails Saga



A Presentation by Margaret K. McNulty-Beldyk

Aulë is getting impatient!



∞ Eru Ilúvatar has been announcing his new release of Elves and Humans for **centuries**, but all we've seen are delays, delays, delays. It's just **vaporware!** I want there to be inhabitants of Arda **now!**

Melkor Chimes In



I know, it's lame, isn't it?
I'll bet that I could create
better inhabitants for Arda
than Eru could, and I'd be
able to do it faster and
under budget too!



I have an idea!



Let's make inhabitants of Arda of our own! If we do that, then Eru Ilúvatar will surely be super proud of us and reward us with lots of development contracts.

And So It Begins



☞ A good application begins with a good framework. I know! I'll use Ruby on Rails!

Framework? Bah.

☞ I'll just do my own thing. There's no way I'd be bound up in something so rigid, that's stupid! I can make a way better application on my own!



Are you sure?



Well, if you say so ...
But I'm going to use
Rails, and I'll bet Eru
will like my application
better.

Then May the Best Valar Win!



About Ruby on Rails



☞ Ruby on Rails, also known as Rails, is a web development framework for the Ruby programming language.

About Ruby on Rails

∞ Wait, I thought Ruby on Rails was a programming language.



About Ruby on Rails



❧ Quite incorrect, Melkor. Ruby is a general-purpose object-oriented programming language developed by Yukihiro Matsumoto in the 1990's. Rails is a web development framework based on that language.

About Ruby on Rails

∞ Oh, well, that's dumb. I think I'll just use CGI.



Ruby on Rails History



Ahem Anyway, Rails was developed by David Heinemeier Hansson in 2004, although it did not really take off until 2007 when it was shipped with the Mac OS X operating system, which, as you know, is what everyone uses in the Undying Lands.

Ruby on Rails History

∞ I prefer Windows myself.



Ruby on Rails History



✧ I'm sure that you do. Anyway, Rails 2.3 was released in 2009, and with it came many improvements, pervasive Rack integration, refreshed support for Rails Engines, nested transactions for Active Record, dynamic and default scopes, unified rendering, more efficient routing, application templates, and quiet backtraces.*

*source: http://guides.rubyonrails.org/2_3_release_notes.html

Ruby on Rails History



2011 brought even more exciting changes with the release of Rails 3.0, which emphasizes RESTful practices, a chainable query language for ActiveRecord, and delightfully unobtrusive JavaScript helpers.

*source: http://guides.rubyonrails.org/3_0_release_notes.html

Ruby on Rails History

∞ Blah, blah, this is boring ... I think I'll set my monitor on fire.



Getting Started



Well since you're so impatient, let's get started. Since everyone in the Undying Lands uses Mac OS X, I'm going to assume you have Rails already. If you don't, then you can find out how to get it at <http://rubyonrails.org/download>

Getting Started



✎ To start a new Rails project, fire up a console, navigate to the folder where you keep your projects, and type “rails new <<project name>>.” Just hit enter, and Rails generates everything you need to get started. Simple, neh?

Getting Started



```
Nebula:~ Nebula$ rails new dwarves
create
create  README
create  Rakefile
create  config.ru
create  .gitignore
create  Gemfile
create  app
create  app/assets/images/rails.png
create  app/assets/javascripts/application.js
create  app/assets/stylesheets/application.css
create  app/controllers/application_controller.rb
create  app/helpers/application_helper.rb
create  app/mailers
create  app/models
create  app/views/layouts/application.html.erb
create  app/mailers/.gitkeep
create  app/models/.gitkeep
create  config
create  config/routes.rb
```

Getting Started



```
dwarves — bash — 128x19
Nebula:~ Nebula$ cd dvarves/
Nebula:dvarves Nebula$ ls
Gemfile      Rakefile      config.ru     lib           script        vendor
Gemfile.lock app            db            log           test
README       config        doc           public        tmp
Nebula:dvarves Nebula$
```

Rails Principles



There are several important principles of Ruby on Rails which I believe leads to better code, and better web development. And the first one is ...

Don't Repeat Yourself



∞ DRY. Don't repeat yourself.

Don't Repeat Yourself



∞ What?



Don't Repeat Yourself



∞ Don't repeat your ...
Heeeey!

Don't Repeat Yourself

∞ Sorry, mate, I
couldn't resist.



Don't Repeat Yourself



☞ **grumble grumble**
Repetitive code causes a lot of bugs, takes longer to write, and is difficult to read. If you decide to make a change, then isn't it much easier to only make that change once? Therefore, good Rails developers always extract repeated code into methods or utilize helper classes.

Don't Repeat Yourself



- ☞ There are also lots of Rails libraries and Ruby gems available that can help create common elements, like pagination. No need to re-invent the Song of the Ainur!

Don't Repeat Yourself

∞ Well, I don't really expect to have to change my code much, since it will be perfect to begin with. Moving on...



Convention over Configuration



As you saw from the project generation, Rails does a lot of stuff for you, if you're willing to let it. However, it requires following Rails conventions in order to do it.

Convention over Configuration



☞ That sounds like a great way ... to stifle creativity. Thanks, but I'll do things my way.



Convention over Configuration



☞ There's still plenty of things you can control! But if you keep to convention, Rails will automatically know what to do with certain files. For instance, for my Dwarves project, the MiningController is called `mining_controller.rb` and is located in the `app/controllers` directory. Which brings us to...

Model, View, Controller



- ☞ Rails comes with extensive support for the Model View Controller architecture, which helps to keep code neat and well-organized. MVC architecture separates the “business logic” of the controller from the application data and the front-end GUI.

Model, View, Controller



☞ The code in Rails you'll spend the most time editing is located in the `app/` folder. There are many folders in the `app` folder, but the most important ones are `model/`, `view/`, and `controller/`. The code for each type of file is automatically placed in its respective folder when it is generated.

Model



- ∞ The model holds the raw data of the object. Any actions that operate on the data and work with the database are contained in the model. The model contains the name, gender, and age of my Dwarves, among other things.

Model



```
app — bash — 108x10
Nebula:app Nebula$ rails generate model dwarf
  invoke  active_record
  create  db/migrate/20120322154615_create_dwarves.rb
  create  app/models/dwarf.rb
  invoke  test_unit
  create  test/unit/dwarf_test.rb
  create  test/fixtures/dwarves.yml
Nebula:app Nebula$
```

Model

A screenshot of a code editor window titled "dwarf.rb — dwarves". The editor shows the following Ruby code:

```
1 class Dwarf < ActiveRecord::Base
2
3   attr_accessible :name, :beard, :age, :mining_skill, :fighting_skill
4   has_many :axes
5   belongs_to :aule
6
7 end
8
```

On the left side of the editor, a file explorer shows the project structure for "dwarves":

- dwarves
 - app
 - assets
 - controllers
 - helpers
 - mailers
 - models
 - dwarf.rb
 - views
 - config
 - config.ru

The Model and Rails



- Models work closely with ActiveRecord, the Rails tool for interacting with databases. ActiveRecord works with almost any kind of database, and greatly simplifies complicated SQL queries.

View



☞ The view is the front-end GUI for the application – basically, everything the user sees, everything that makes it look pretty. The short, stocky, bearded figure of the dwarf is the view for my application.

View



```
dwarves — bash — 138x27
Nebula:dwarves Nebula$ rails generate controller dwarf show
  create  app/controllers/dwarf_controller.rb
  route  get "dwarf/show"
  invoke erb
  create  app/views/dwarf
  create  app/views/dwarf/show.html.erb
  invoke test_unit
  create  test/functional/dwarf_controller_test.rb
  invoke helper
  create  app/helpers/dwarf_helper.rb
  invoke test_unit
  create  test/unit/helpers/dwarf_helper_test.rb
  invoke assets
  invoke coffee
  create  app/assets/javascripts/dwarf.js.coffee
  invoke scss
  create  app/assets/stylesheets/dwarf.css.scss
Nebula:dwarves Nebula$
```


View



```
show.html.erb — dwarves
x dwarf.rb ● show.html.erb
1 <h1>OMG IT'S A DWARF! His name is <%= @dwarf.name %>!!!</h1>
2
3 <%= image_tag("#{@dwarf.name}.png", :alt => @dwarf.name, :class => "awesome") %>
4
5 <p>Pretty sweet dwarf, neh?</p>
6
```

View



OMG IT'S A DWARF! His name is Gimli!!!



Pretty sweet dwarf, neh?

The View and Rails



- ☞ The Rails View is handled by the `ActionView` class. The most common use of `ActionView` is to use `ERB`, which allows coders to embed ruby code in `HTML` or `XML` files. It's important to only embed Ruby that helps with the view, however – the business logic should go in the model or controller.

Controller

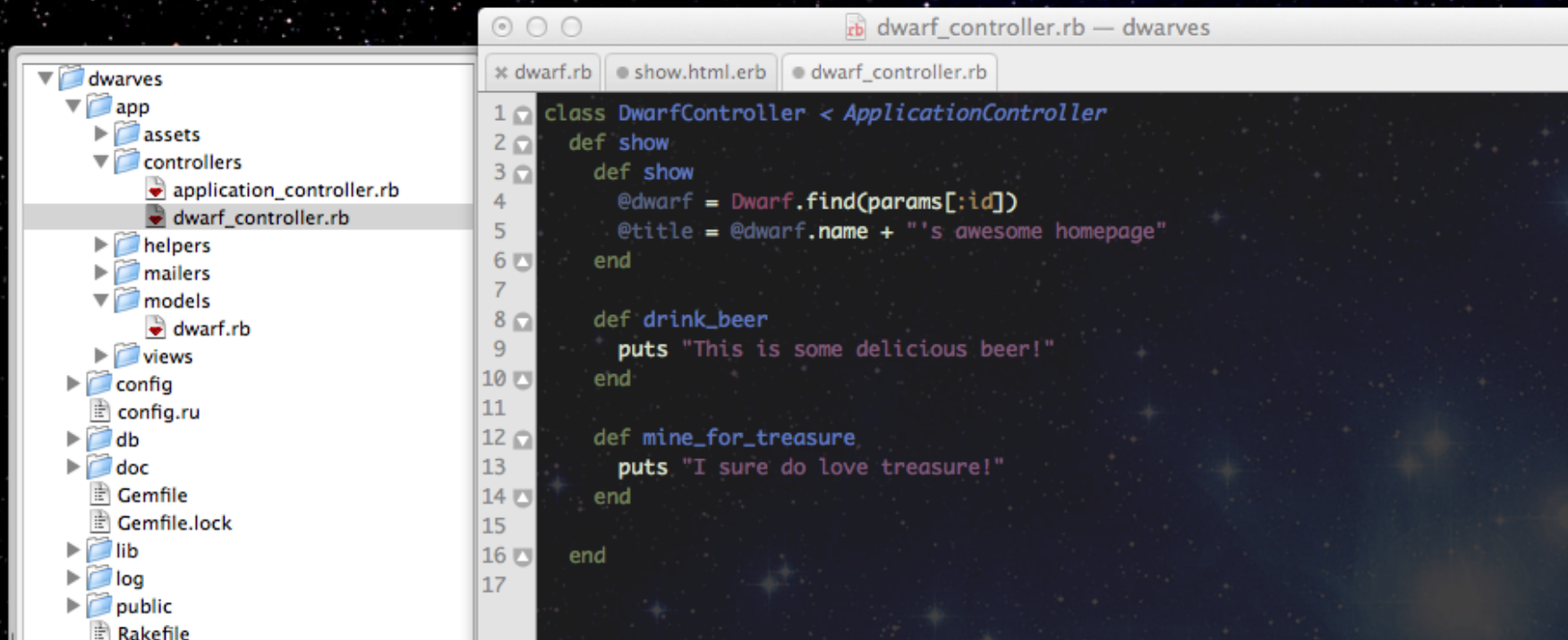


- ☞ The controller is the where the “brain” of the object – in this case, dwarves. The controller is what should hold the actions the object contains, such as `drink_beer` and `mine_for_treasure`.

Controller



- ☞ The controller was generated when the view was. It already has a “show” method from the view generation, but we can edit that and add some new methods as well.

A screenshot of a code editor window titled 'dwarf_controller.rb — dwarves'. The editor shows the code for the DwarfController class, which inherits from ApplicationController. The code includes a show method, a drink_beer method, and a mine_for_treasure method. The show method uses Dwarf.find to retrieve a dwarf and sets the title. The drink_beer method prints a message, and the mine_for_treasure method prints another message. The editor also shows a file explorer on the left with the project structure, including folders like app, assets, controllers, helpers, mailers, models, views, config, db, doc, Gemfile, lib, log, and public, and files like application_controller.rb, dwarf_controller.rb, dwarf.rb, config.ru, Gemfile.lock, and Rakefile.

```
1 class DwarfController < ApplicationController
2   def show
3     def show
4       @dwarf = Dwarf.find(params[:id])
5       @title = @dwarf.name + "'s awesome homepage"
6     end
7
8     def drink_beer
9       puts "This is some delicious beer!"
10    end
11
12    def mine_for_treasure
13      puts "I sure do love treasure!"
14    end
15
16  end
17
```

The Controller and Rails



- ☞ Rails uses something called ActionController to facilitate communication between the controller, model, and view. It's not really something you need to worry about, just make sure everything is in the right file. Convention over configuration, remember?

That Sounds Hard!

☞ Bah, this is all way too complicated. I put my entire web app in one giant Perl file! That's better, right?



Testing in Rails



... Yeah, Melky, you have fun with that. But MVC is not the only advantage of Rails. Ruby on Rails also offers a plethora of testing platforms to ensure that your code is as free from bugs as possible!

Testing in Rails



- ☞ The built-in Rails unit test framework is quite good, but you can also use tools like Rspec and Cucumber to create acceptance, integration, and functional tests. The rails community encourages test-driven design, and the Rails framework makes it easy!

Testing in Rails



The screenshot shows a Rails application interface. On the left is a file explorer for a project named 'dwarves'. The 'test' directory is expanded, showing sub-directories for 'fixtures', 'functional', 'integration', 'performance', and 'unit'. The 'unit' directory is further expanded to show 'dwarf_test.rb' and 'helpers'. On the right is a code editor window titled 'dwarf_test.rb — dwarves'. It contains the following Ruby code:

```
1 require 'test_helper'
2
3 class DwarfTest < ActiveSupport::TestCase
4
5   before_each do
6     dwarf thorin = Dwarf.new
7   end
8
9   test "dwarf should mine for treasure" do
10     thorin.mine_for_treasure
11     assert :success
12   end
13
14   test "dwarf should have axe" do
15     assert thorin.has_axe == true
16   end
17
18 end
19
```

Testing in Rails

☞ But Aulë, I don't need to test my code, my code is perfect already.



Testing in Rails



∞ You may scoff at testing, but you may one day have an apprentice who embraces it*.

*Shameless plug:

<http://cse1.cs.colorado.edu/~mcnultym/oo/tdd/tdd.pdf>

Testing in Rails

☞ Bah, some apprentice.
If he tests his code,
he'll never be as evil as
me!



Conclusion



Well, all of **my** tests pass at least --my Rails application development is complete, let's have a look at the results!

Dwarves!



- Model: Name, Age, Beard, Gender
- View: This stylish picture of a dwarf!
- Controller: Swing axe, drink beer, hunt for treasure



Conclusion



☞ Sure, no code is completely bug-free, but those are some pretty sweet dwarves if I do say so myself.

Conclusion

∞ Ha! Sucker! I finished my code hours ago while you were wasting time with “MVC” and “Testing.”



Orcs!



∞ RRRAAAAWWR!!!!



Conclusion



☞ Eh... close enough. Let's approach Ilúvatar and see who won!



Conclusion



© www.TedNasmith.com

☞ You should not have gone behind my back! Waiting until the end to talk to the product owner is not very Agile 😞

Conclusion



∞ I'm sorry, Eru ☹

Conclusion



© www.TedNasmith.com

Oh, I can't stay mad at you, Aulë. I appreciate your use of Rails MVC architecture in creating the dwarves. For that, I shall gift them with sentience. But I urge you use more iterative design in the future.

Conclusion



∞ What about my orcs? Aren't they way more awesome than some stupid dwarves?



Conclusion



∞ As for you, Melkor, for this abomination I will give you the worst punishment I can think of.

Conclusion



∞ You must maintain this code you wrote until the end of time!

Conclusion



☞ NOOOOOOOOOO!!!!!!



Conclusion



∞ In other news, my own software project is complete and well-tested!

Humans and Elves!



Conclusion



☞ Wow, good job, Eru! I'll bet you used Rails to develop those, didn't you?

Conclusion



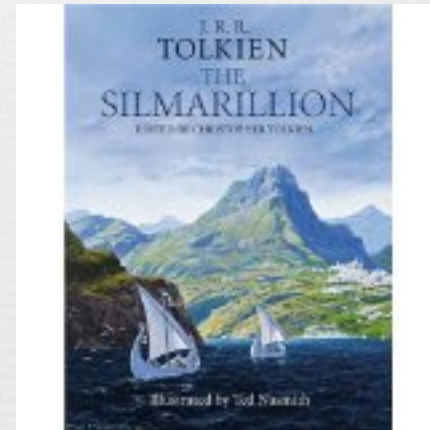
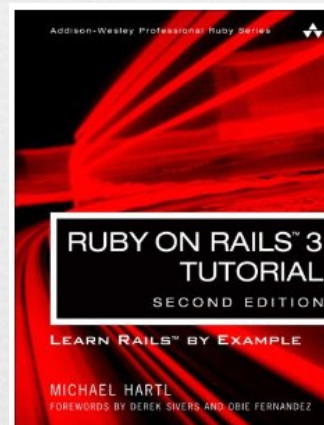
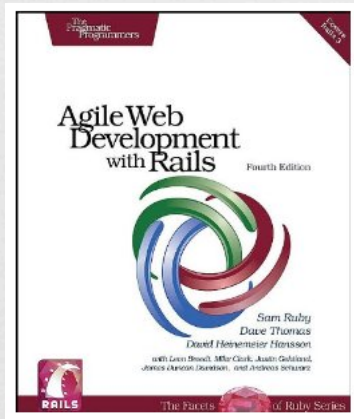
∞ No, I used Django.

The End!

Sources



☞ For more information, check out these books:



☞ Image Sources: http://tolkiengateway.net/wiki/Main_Page