Web Engineering:
Finding your way around Rails

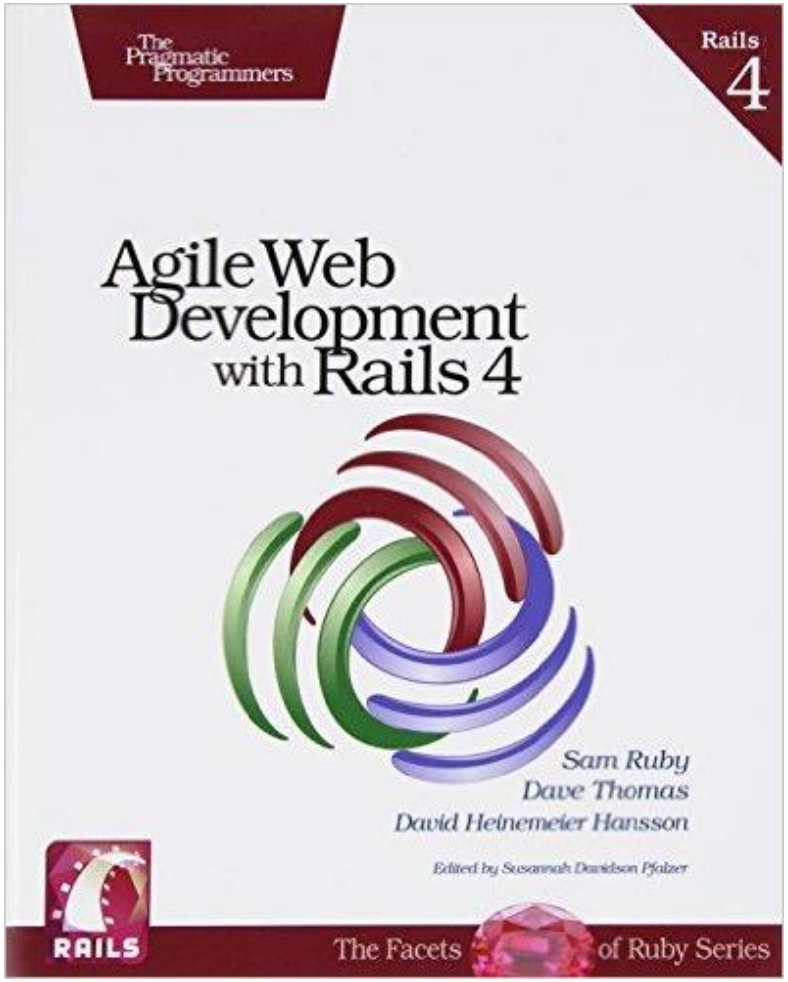
The University of Aizu
Quarter 2, AY 2018

Outline

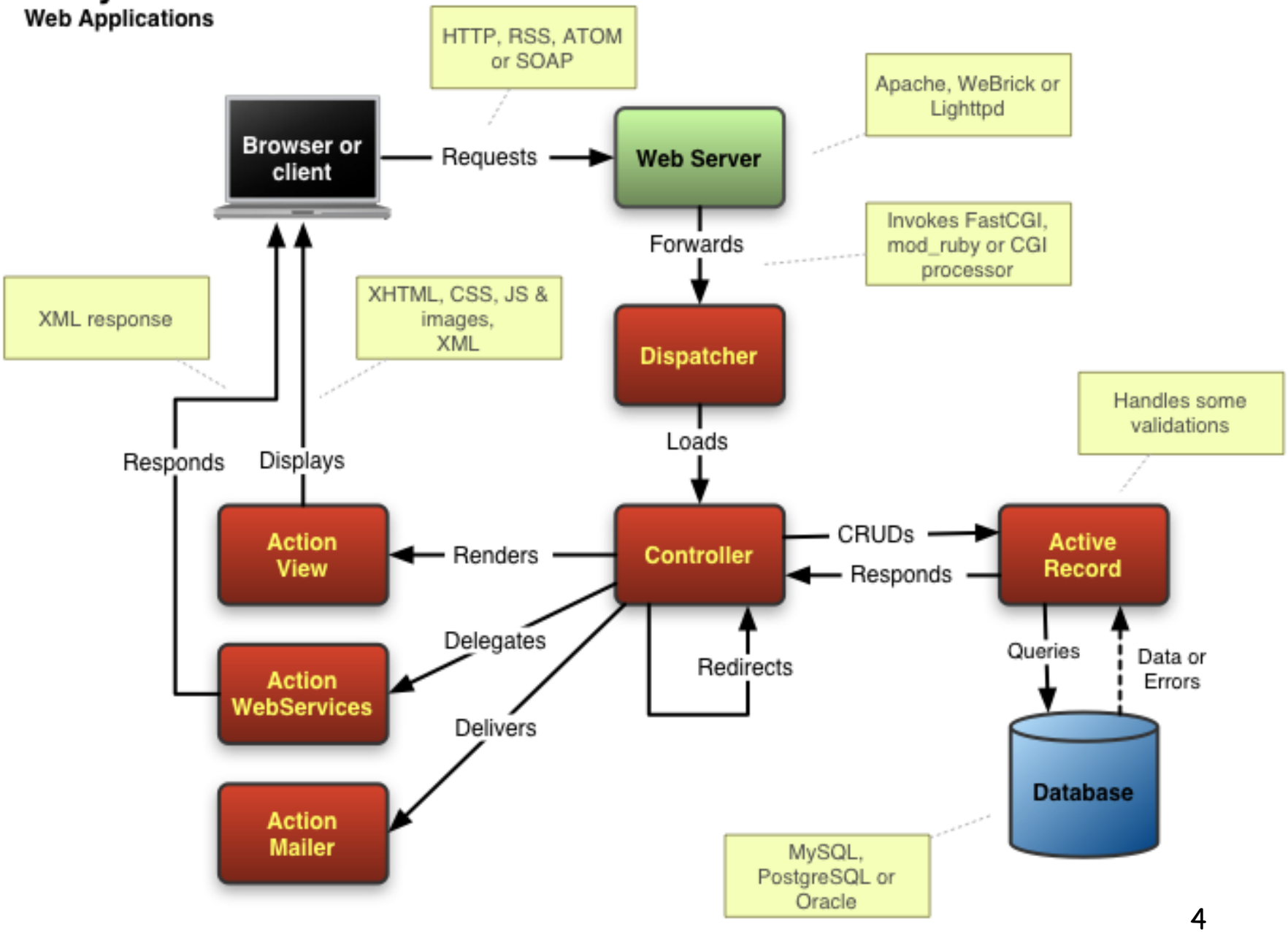
- There things go to
- Naming conventiion

Literature

- Agile Web Development with Rails 4 (1st edition) by Sam Ruby, Dave Thomas and Devid Hansson, The Pragmatic Bookshelf, 2013.
  - Chapters 18.



Ruby on Rails
Web Applications

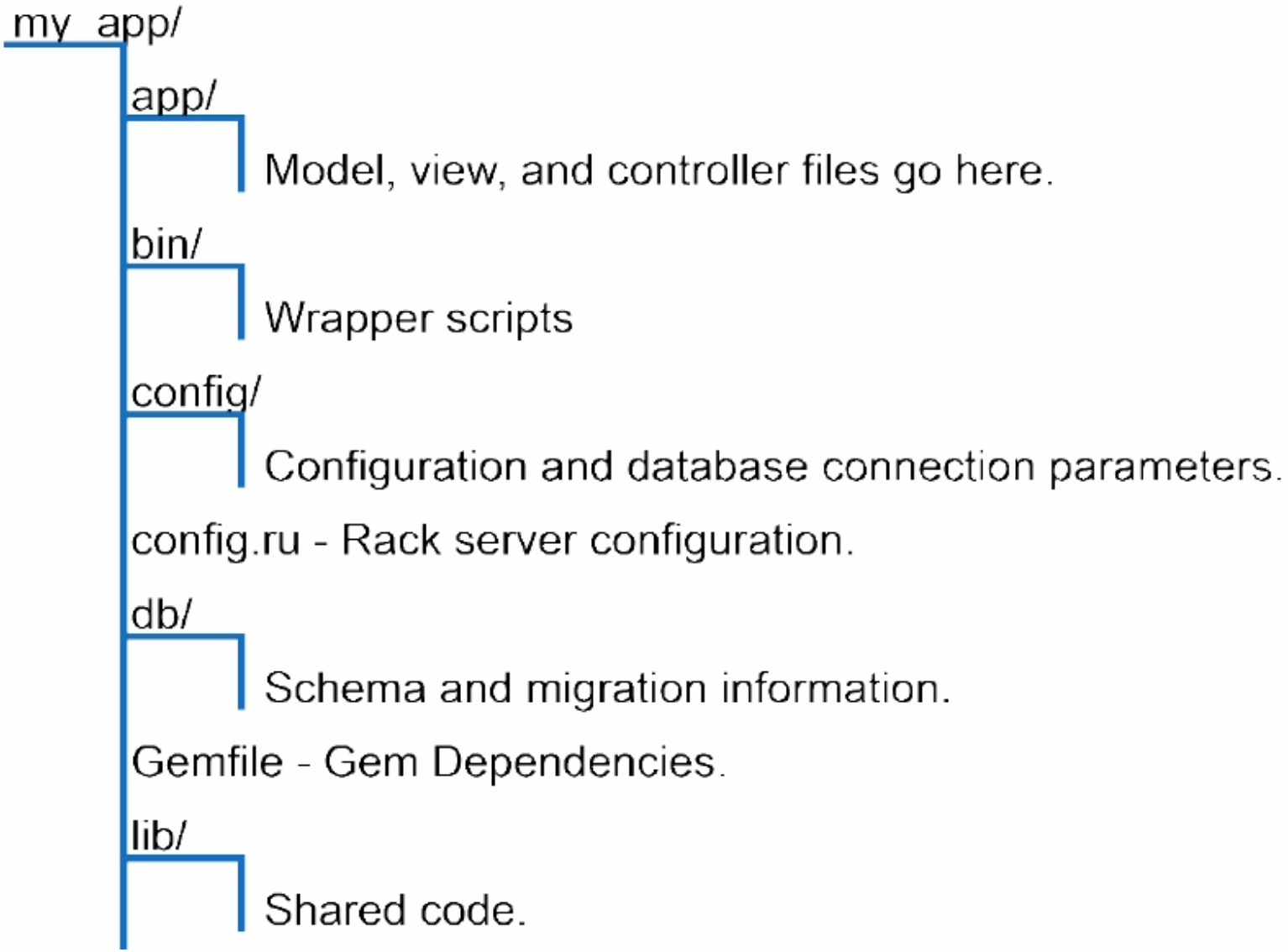


There things go

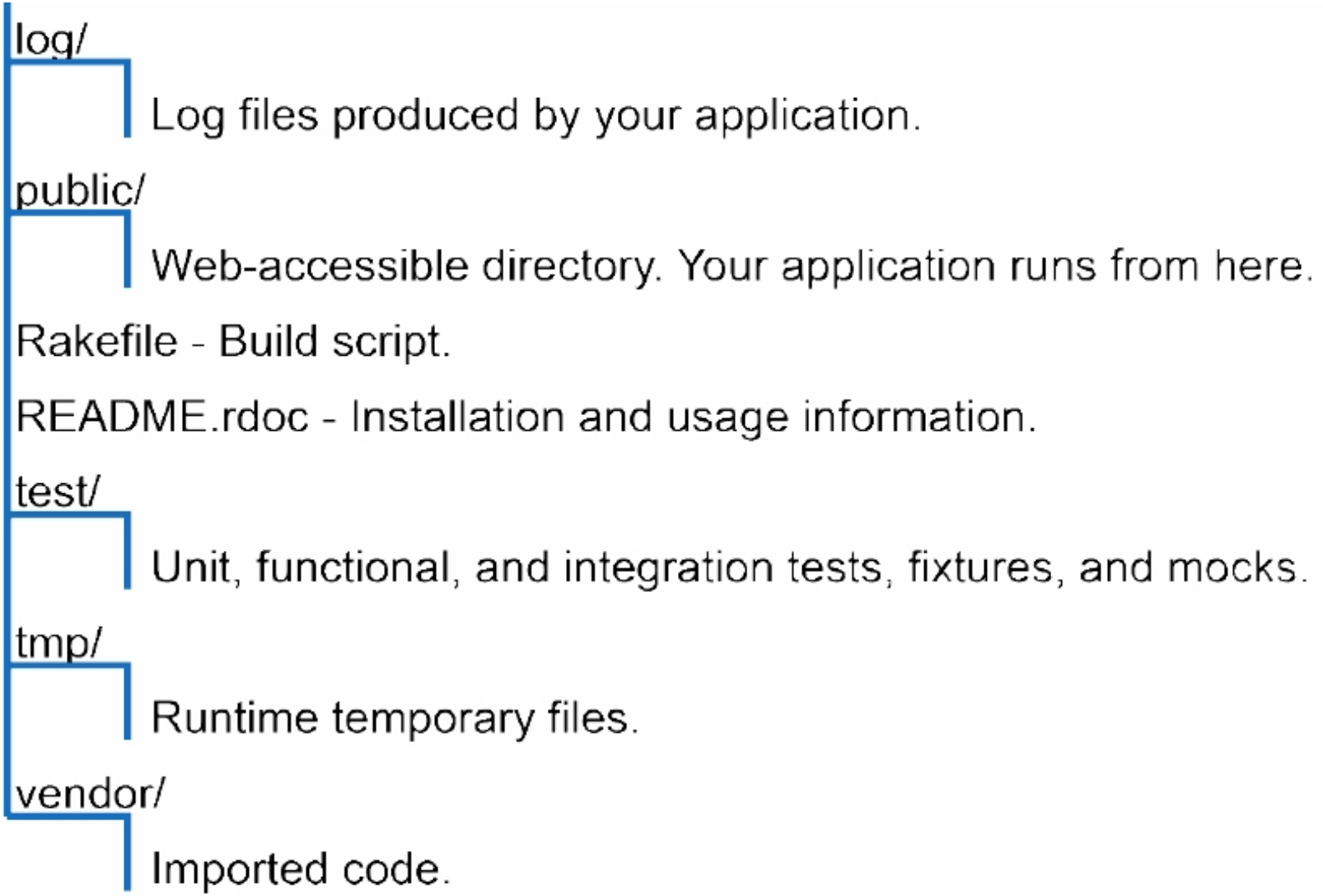
- Now. It is time to all high-level stuff you need to know:
  - Directory structures
  - Configuration, and
  - Environments.
- When you run the command:
  - rails new my_app

  The top-level directory for our new application appears as shown in Figure (see next Slide).

Top-level directory structure (1)



Top-level directory structure (2)



Comments on Slides 6 and 7

- Text files in the top of application directory:
  - *config.ru* configurates Rack Web server Interface
  - *Gemfile* specifies the dependences of your Rail application
  - Rakefile defines tasks to run tests, create documentation, etc.
    - To see a full list, type:
      - rake --tasks
    - To see a complete description of a specific task
      - rake --describe task
  - README.rdoc contains general information about the Rails framework.

Comments on Slides 6 and 7

- The *test* directory is the home for all testing-related activities
- The *lib* directory holds application code that
  - does not fit neatly into a model, view, or controller;
  - That is shared among models, views, or controllers.

Comments on Slides 6 and 7

- All logs are stored in the *log* directory. The main log files are:
  - *development.log*
  - *test.log*
  - *production.log*
- The public directory is the external face of your application. The Web server takes this directory as the base of the application. This is a place for static (unchangeable) files: style sheets, JavaScripts.

Comments on Slides 6 and 7

- The *bin* directory holds the Rails script: You run this script when run *rails* command from the command line. The first argument for this command is the function Rails should perform
  - *console* to interact with your Rails application methods;
  - *dbconcole* to directly interact with your database via command line;
  - *destroy* to remove autogenerated files created by generate
  - generate to create controllers, mailers, models, scaffolds, and wev services.
- See the next slide

Comments on Slides 6 and 7

- The *bin* directory (continues)
  - *new* generates Rails application code;
  - *plugin* helps you to install and administer plugins (pieces of functionality that extend the capabilities of Rails);
  - *runner* executes a method in your application outside the context of the Web. This is the noninteractive equivalent of *rails console;*
  - *server* runs your application in a self-contained web server using Mongrel or WEBrick.
- The *tmp* directory for cache contents, sessions, sockets. These files are cleaned up automatically by Rails.

Comments on Slides 6 and 7

- The *vendor* directory is for third-party code:
  - There two sources for this code
    - Rails installs plugins into directories below *vendor/plugins*. Plugins are the ways to extend Rails functionality, both during development and at runtime.
    - You may install Rails and all of its dependences into the *vendor* directory.

Comments on Slides 6 and 7

- The *config* directory contains files that configure Rails.
  - Before running your application, Rails loads and executes config/environment.rb and config/application.rb. These files set up the standard environment. It includes the following directories (relative to your application's base directory):
    - The app/controllers directory and its subdirectories;
    - The app/models directory and its subdirectories whose names start with an underscore or a lowercase letter;
    - The vendor directory and lib contained in each plugin subdirectories;
    - The directories app, app/helpers, app/mailers, app/services, and lib.

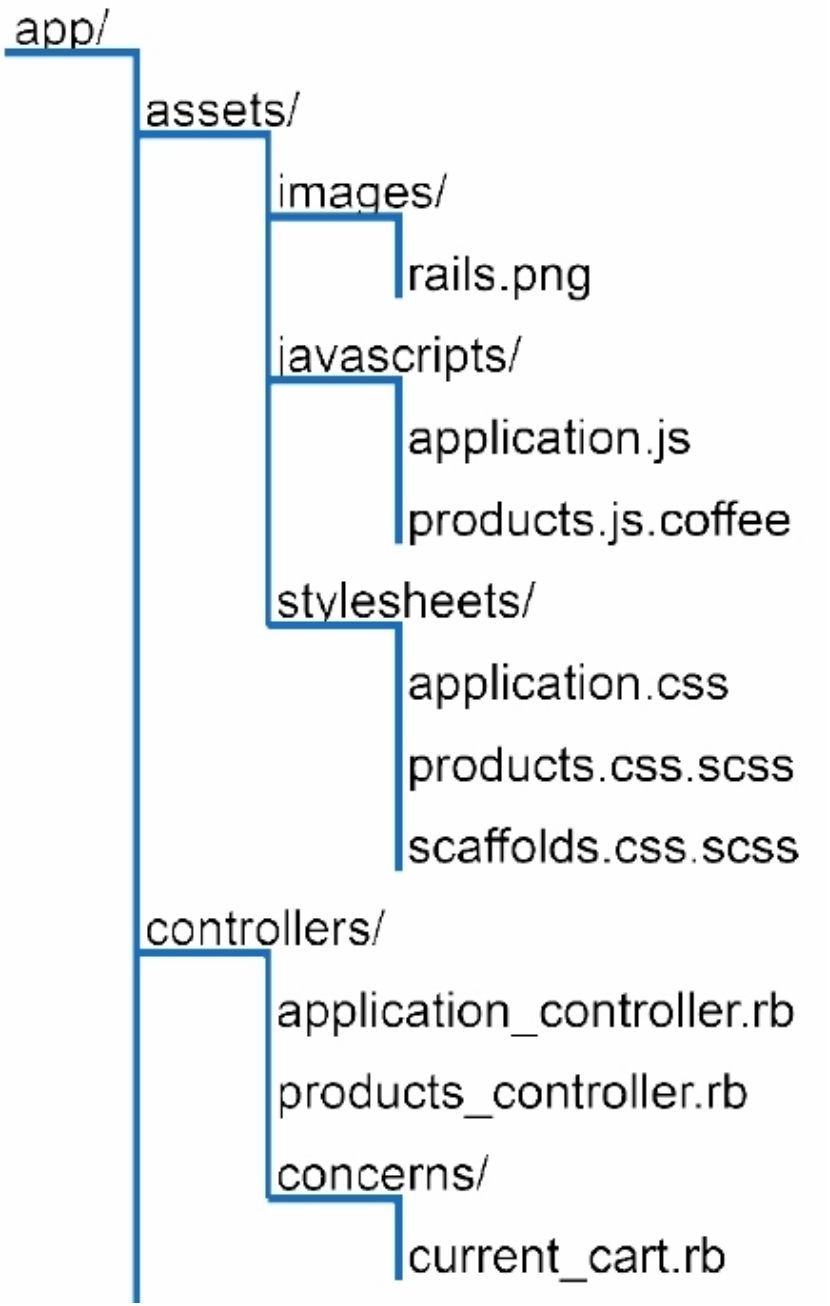
Comments on Slides 6 and 7

- The *config* directory (continued)
  - In additional, Rails loads a per environment configuration file. This file lives in the environment directory and is where you place configuration options that vary depending on the environment.
    - This is done because Rails recognizes that you needs, as a developer are very different when writing code, testing code, and running code in production.
      - When writing code, you want lots of logging, convenient reloading of changed source files, in-your face notification of errors, etc.
      - When testing, you want a system that exist in isolation so you can have repeatable results.
      - In production, your system should be tuned for performance and users should be kept away from errors.
    - This means that no application code needs to be changed as you move from development through testing to production.

Comments on Slides 6 and 7

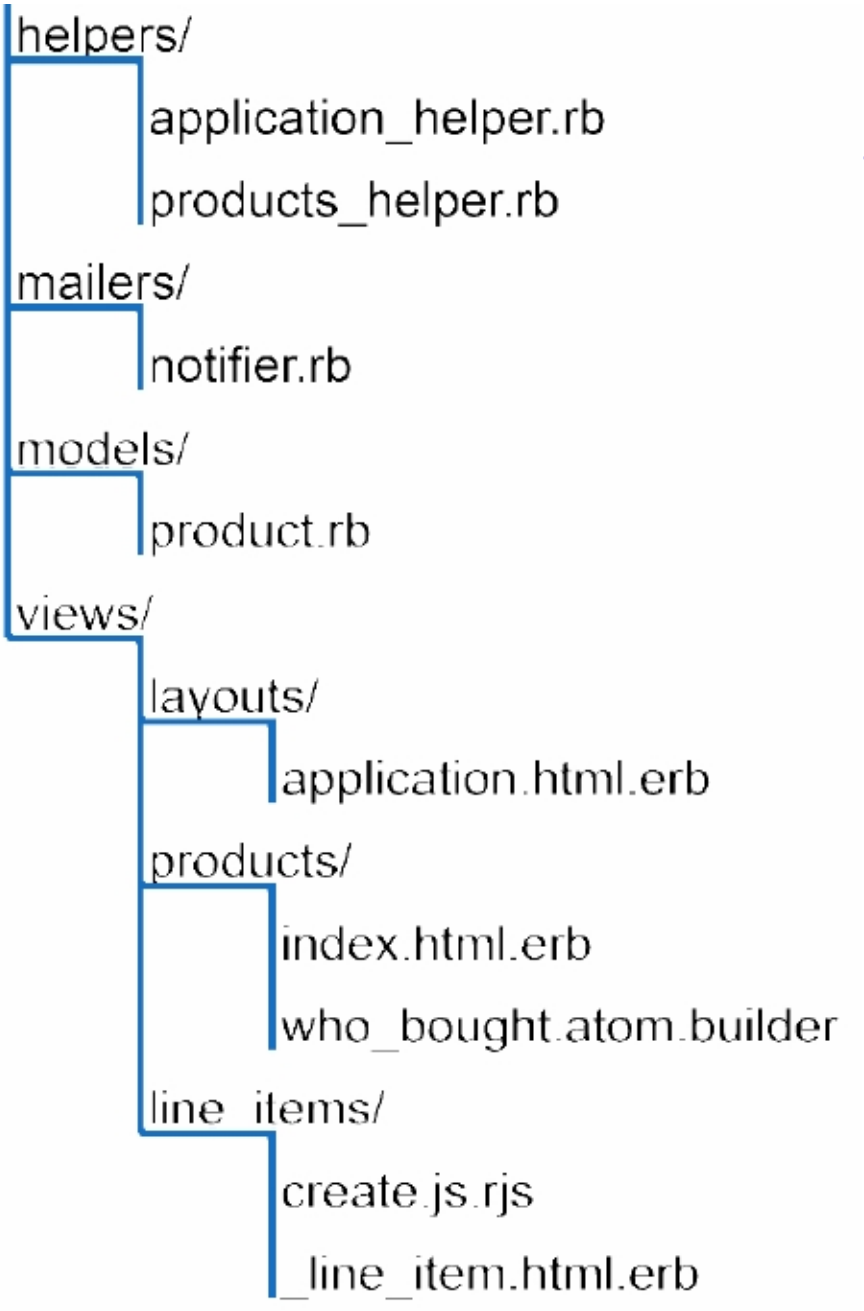
- The *config* directory (continued)
  - When starting WEBrick, you may type to specify environment:

```
depot> rails server -e development
depot> rails server -e test
depot> rails server -e production
```



Structure of the *app* directory

The main code for the application lives in the *app* directory.



Structure of the *app* directory (2)

The main code for the application lives in the *app* directory.

Naming conventions

- Variable names and file names are all lower case and words are separated by underscores.
  - Example: order_status
- Names of classes and modules:
  - No underscores in phrases and each word is capitalized.
    - Example: LineItem
- Rules for database table names, such as variable names, are the same as for variable name above plus table names are always plural.
  - Examples: orders, third_parties

Naming conventions

- Rails uses this knowledge of naming conventions to convert names automatically.
  - If you define the class named *LineItem*, Rails understands the following:
    - The corresponding database table is called *line_items*.
    - File with the class definition is *line_item.rb* (in the app/models directory).

Naming conventions

- Controllers: If the application has a *store* controller, then the following happens:
  - Rails assumes the class is called *StoreController* and that it is in a file named *store_controller.rb* in the *app/controllers* directory.
  - It assumes there is helper module named *StoreHelper* in the file *store_helper.rb* in the *app/helpers directory*.
  - It will look for view templates for this controller in the *app/views/store* directory.
  - It will take the output of these views and wrap them in the layout template contained in the file *store.html.erb* or *store.xml.erb* in the *app/views/layouts* directory.
- All these conventions are on the next slide.

Naming conventions

Model Naming	
Table	line_items
File	app/models/line_item.rb
Class	LineItem

Controller Naming	
URL	http://../store/list
File	app/controllers/store_controller.rb
Class	StoreController
Method	list
Layout	app/views/layouts/store.html.erb

Naming conventions

View Naming

URL	http://../store/list
File	app/views/store/list.html.erb (or .builder)
Helper	module StoreHelper
File	app/helpers/store_helper.rb

What we just did

- Everything in Rails has a place
- In each place, files and data contained in them follow the naming convention
- Each of the Rails execution environments can be separately configured.