# djangocon-2012-notes Documentation

***Release 1.0.0***

**Kenneth Love**

September 24, 2012

# CONTENTS

Notes, duh!

CHAPTER

ONE

# DJANGOCON 2012

## 1.1 Maintaining Your Sanity While Maintaining Your Open Source App

Mark Lavin

### 1.1.1 Packaging

- Use pip
- PEP 386
- List a description and *__version__* number in *__init__.py*

### 1.1.2 Anti-Patterns

- giant README
- Docs which aren't available online
- Use Sphinx and RTD
- Create docs before you think you need them

### 1.1.3 Things to Document

- Description of the project and its goals
- How to install, including requirements
- How to configure
- Release notes (not svn/git/hg logs)

### 1.1.4 Test Only Models

- Define models in tests.py and they're only available to the test runner (already used by Django)

### 1.1.5 Testing

- Tox uses virtualenv to run a test matrix * different Python versions * different Django versions * different DB backends
- You can have Tox build your Sphinx docs
- sul

## 1.2 Creating Dynamic Applications with Django and Backbone.js

Mjumbe Poe

### 1.2.1 Overview

- Backbone.js is cool, but sorry for all the JS
- Hijax: Accessibility is important
- Django REST Framework: it's cool & we're goinna make it cooler
- Templating in Django: Great but could be better.

### 1.2.2 Django Analogues

- **Backbone has models, which are roughly like Django models**
  - No defined structure
  - Also has form-like responsibilities, like validation
- **Backbone has collections, which are like Django model managers or querysets**
  - Use them find/destroy/iterate through instances of models
- Backbone has events, which are similar to Django signals
- Backbone has views which are managers of portions of the page
- Backbone has routers which are similar to urlpatterns in Django

### 1.2.3 RESTful API

- Using a basic *views.View* view from Django REST Framework (not what I'd consider best practices -kl)

### 1.2.4 Hijax

- Plan for AJAX from the start, but implement at the end.

## 1.3 Designing Your Open-Source Project

Bryan Veloso

### 1.3.1 The problem

- Designers are picky
- Designers are stubborn
- Designers are just like you (developers)

Pixel perfection === PEP8 perfection Clarity through interface === clarity through documentation

A designer's ability to code === nosql (things we bicker about)

Designers to developers: "make it work" Developers to designers: "make it pretty"

### 1.3.2 Recruiting a Designer

- Pitch it on github
- Find them on Dribbble
- The hallway track

### 1.3.3 To My Fellow Designers

- Open source is experiential
- Open source is crucial
- Make yourself known

### 1.3.4 Hybrids

- Universal translators
- Knowledge exchanges
- Find them, train them
- Learn from hybrids
- Pair with hybrids

### 1.3.5 Working with Designers

- Time investment
- They need space
- Restrictions and boundaries
- Death to spec work

Design matters to Django so design should matter to you.

## 1.4 API Design Tips

Daniel Lindsley

### 1.4.1 What?

- Not HTTP APIs
- Programmatic APIs
- **Think libraries**
  - especially ones you hand off to other people...

### 1.4.2 Why?

- Think about how many times you've started with someone else's library...
- Used it some...
- Then got really upset and frustrated
- Other people use your code all the time
- Nice APIs begets *Happiness*
- Happiness begets *Recommendations*
- Recommendations beget *Users/Community*

### 1.4.3 Philosophy

- **You can't make everyone happy by default**
  - You should still have sane defaults
- **But *more* people will be happy if they can tweak it**
  - You can bet they'll need to
- And most people will be happy if it's easy to tweak
- **No copy-paste should be needed**
  - Boilerplate sucks
- **They shouldn't have to constantly refer to the docs**
  - Especially not for things they use *all* the freaking time
- **Good docs matter**
  - Saves you (support) & them (implementation) time. Everyone wins
- Real World use is the best sanity check
- **Someone is going to something weird/insane with your code**
  - It's inevitable, so design for it up front

### 1.4.4 Approaches on Design

- **Common Methodologies:**
  - **Bottom-up**
    - small components over time that eventually all work together in the final product

    - **Top-down**
        * build the ideal code at first, then writing the underlying code to make the ideal real
- **Bottom-up sucks**
    - sure, you built little pieces that work
    - **but do they really work well together?**
        * likely not
        * how is php formed
- **Top-down feels better**
    - everything fits together right
    - less duplication
    - helps you to resist the urge to duct tape things together
- **With some instant TDD, you get your tests started from the get-go**
    - fewer massive, painful refactories down the road

### 1.4.5 Things You Should Do

- **Small components**
    - worked for UNIX, it'll work for you
- **Composition >= Inheritance**
    - Why do the work yourself when you can delegate?
- **Reflection**
    - If data can flow one way, add the opposite
- **Broad familiarity**
    - If it's a similar task to something else they know, mimic that something else
- **Narrow familiarity**
    - Call signatures matter
- **Assume the worst**
    - Don't code for just the easy case

### 1.4.6 Things You Should Not Do

- Stop at a low level
- Wildly different return values
- Useless "implementation" code
- If it's difficult to test...

### 1.4.7 Django-specific Topics

- Pluggable backend all the things
- Internationalize all the things
- **Dynamically loaded classe/code**
  - 60% more error-handling, every time
- **Declarative syntax**
  - Metaclasses: call your doctor if headaches last more than 4 hours
- *Don't* metaclass all the things
- **The ORM**
  - Love it or hate it, there's some great learning opportunities there
- Decrease reliance on *self*
- **Resist the urge to magic**
  - Be explicit *first*, then add shortcuts (which can be a little more magical)

### 1.4.8 In Conclusion

- Use the golden rule
- Consistency is key
- Plan for the worst & include sweet shortcuts
- Make something that you love and make it better