TDD WITH DJANGO 1.4

Really, just do it!

Martin Brochhaus (@mbrochh)
PyCon APAC 2012

WHY TEST DRIVEN DEVELOPMENT?

- Think about your own API
- Helps with refactoring
- Confidence in your codebase
- Guidance for new team members
- Extra documentation
- Saves a lot of time (in the long run)
- Highly rewarding work experience
- IT IS FUN!

TWO KINDS OF TESTS

- **Unit Tests**
  - Testing small units of code
  - i.e. methods of a Model
  - they must run as fast as possible

- **Integration Tests**
  - Testing the whole system
  - i.e. Views with Selenium
  - they are slow

PYTHON TESTING TOOLBELT

- **Unittest** http://docs.python.org/library/unittest.html
- **Nose:** http://readthedocs.org/docs/nose/en/latest/
- **Coverage:** http://nedbatchelder.com/code/coverage/
- **Mock:** http://python-mock.sourceforge.net/
- **Fabric:** http://fabric.readthedocs.org/en/1.4.1/index.html
- **Watchdog:** https://github.com/gorakhargosh/watchdog

DJANGO TESTING TOOLBELT

- **Django**
  - https://docs.djangoproject.com/en/dev/topics/testing/
- **Beware:**
  - Don't use self.client.get or Selenium in your unit tests
  - Don't use .json fixtures
- **Django Ecosystem**
  - Use factory_boy (https://github.com/dnerdy/factory_boy)
  - Use django-nose (https://github.com/jbalogh/django-nose)
  - Use django-coverage (https://github.com/kmike/django-coverage)
  - Use django-jasmine for Javascript (https://github.com/Fandekasp/django-jasmine)

STRUCTURE YOUR TESTS

- Use a custom testrunner
- Separate integration tests from unit tests
- Create factories for all your models
- Provide requirements_dev.txt file for new developers
- Use Fabric to run your tests
- Provide a README and tell us how to run your tests

PROJECT LAYOUT

```
$ ./django-admin.py startproject myproject .
```

```
- manage.py
- requirements.txt
- fabfile.py
- myproject/
  - __init__.py
  - settings.py
  - testrunner.py
  - test_settings.py
  - urls.py
  - wsgi.py
  - tests/
    - __init__.py
    - factories.py
```

```
$ ./django-admin.py startapp myapp
```

```
- myapp/
  - forms.py
  - models.py
  - views.py
  - urls.py
  - tests/
    - __init__.py
    - factories.py
    - forms_tests.py
    - models_tests.py
    - integration_tests/
      - __init__.py
      - views_tests.py
```

PROJECT LAYOUT

```
$ ./django-admin.py startproj                    n.py startapp myapp

- manage.py
- requirements.txt
- fabfile.py
- myproject/
  - __init__.py
  - settings.py
  - testrunner.py
  - test_settings.py
  - urls.py
  - wsgi.py
  - tests/
    - __init__.py
    - factories.py
                                                  .py
                                                  es.py
                                                - forms_tests.py
                                                - models_tests.py
                                                - integration_tests/
                                                  - __init__.py
                                                  - views_tests.py
```

```
django==1.4
django-extensions
fabric
factory_boy
django-nose
coverage
django-coverage
mock
watchdog
selenium
```

PROJECT LAYOUT

```
$ ./django-admin.py st

- manage.py
- requirements.txt
- fabfile.py
- myproject/
  - __init__.py
  - settings.py
  - testrunner.py
  - test_settings.py
  - urls.py
  - wsgi.py
  - tests/
    - __init__.py
    - factories.py
```

```
"""Custom test runner for the project."""
from django_coverage.coverage_runner import CoverageRunner
from django_nose import NoseTestSuiteRunner


class NoseCoverageTestRunner(CoverageRunner, NoseTestSuiteRunner):
    """Custom test runner that uses nose and coverage"""
    pass
```

```
- tests/
  - __init__.py
  - factories.py
  - forms_tests.py
  - models_tests.py
  - integration_tests/
    - __init__.py
    - views_tests.py
```

PROJECT LAYOUT

```
$ ./django-admin.py st                                    myapp

- manage.py
- requirements.txt
- fabfile.py
- myproject/
  - __init__.py
  - settings.py
  - testrunner.py
  - test_settings.py
  - urls.py
  - wsgi.py
  - tests/
    - __init__.py
    - factories.py
                                                  - models_tests.py
                                                - integration_tests/
                                                  - __init__.py
                                                  - views_tests.py
```

```
EXTERNAL_APPS = [
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

INTERNAL_APPS = [
    'myapp',
]

INSTALLED_APPS = EXTERNAL_APPS + INTERNAL_APPS
```

PROJECT LAYOUT

```
$ ./django-admin.py startp

- manage.py
- requirements.txt
- fabfile.py
- myproject/
  - __init__.py
  - settings.py
  - testrunner.py
  - test_settings.py
  - urls.py
  - wsgi.py
  - tests/
    - __init__.py
    - factories.py
```

```
form os.path import join
from myproject.settings import *

INSTALLED_APPS.append('django_nose')

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": ":memory:",
    }
}

PASSWORD_HASHERS = (
    'django.contrib.auth.hashers.MD5PasswordHasher',
)


EMAIL_BACKEND = 'django.core.mail.backends.locmem.EmailBackend'
SOUTH_TESTS_MIGRATE = False

TEST_RUNNER = 'myproject.testrunner.NoseCoverageTestRunner'
COVERAGE_MODULE_EXCLUDES = [
    'tests$', 'settings$', 'urls$', 'locale$',
    'migrations', 'fixtures', 'admin$', 'django_extensions',
]
COVERAGE_MODULE_EXCLUDES += EXTERNAL_APPS
COVERAGE_REPORT_HTML_OUTPUT_DIR = join(__file__, '../../coverage')
```

```python
import factory

from myapp.models import Entry
from myproject.tests.factories import UserFactory

class EntryFactory(factory.Factory):
    FACTORY_FOR = Entry

    user = factory.SubFactory(UserFactory)
    message = 'A message'
```

YOUT

$ ...in.py startapp myapp

```
- myproject/
  - __init__.py
  - settings.py
  - testrunner.py
  - test_settings.py
  - urls.py
  - wsgi.py
  - tests/
    - __init__.py
    - factories.py
```

```
  - urls.py
  - tests/
    - __init__.py
    - factories.py
    - forms_tests.py
    - models_tests.py
    - views_tests.py
    - integration_tests/
      - __init__.py
      - views_tests.py
```

PROJECT LAYOUT

```
from django.test import TestCase
from django.core.urlresolvers import reverse
from myapp.tests.factories import EntryFactory


class EntryDetailViewTestCase(TestCase):

    def test_view(self):
        entry = EntryFactory()
        resp = self.client.get(reverse('entry_detail',
            kwargs={'pk': entry.pk}))
        self.assertEqual(resp.status_code, 200)
```

THE TDD **DANCE**

- **TEST**
  - self.client.get(reverse('home'))
  - add urls.py and call HomeView.as_view()
  - from myapp.views import HomeView
  - Implement HomeView(TemplateView)
- **FAILURE**
  - NoReverseMatch: Reverse for 'home' with arguments '()' and keyword arguments '{}' not found.
  - NameError: name 'HomeView' is not defined
  - ImportError: cannot import name HomeView
  - TemplateDoesNotExist: home.html

REUSABLE APP LAYOUT

- AUTHORS
- DESCRIPTION
- LICENSE
- MANIFEST.in
- README.rst
- requirements.txt
- setup.py
- **myapp2/**
  - __init__.py
  - models.py
  - urls.py
  - views.py
  - **templates/**

- **myapp2/**
  - **tests/**
    - __init__.py
    - factories.py
    - forms_tests.py
    - models_tests.py
    - urls.py
    - runtests.py
    - **integration_tests/**
      - __init__.py
      - views_tests.py

- These files are needed to upload your app on pypi.python.org
- For local tests use **python setup.py develop**

```
- AUTHORS
- DESCRIPTION
- LICENSE
- MANIFEST.in
- README.rst
- requirements.txt
- setup.py
- myapp2/
  - __init__.py
  - models.py
  - urls.py
  - views.py
  - templates/
```

```
  - models_tests.py
  - urls.py
  - runtests.py
  - integration_tests/
    - __init__.py
    - views_tests.py
```

REUSABLE APP LAYOUT

- AUTHORS
- DESCRIPTION
- LICENSE
- MANIFEST.in
- README.rst
- requirements.txt
- setup.py
- **myapp2/**
  - __init__.py
  - models.py
  - urls.py
  - views.py
  - **templates/**

- **myapp2/**

- This is the actual implementation of your reusable app

REUSABLE APP LAYOUT

- Same test structure as with project apps

- **Problem:**
  - How to run the tests?
  - No manage.py
  - No Django project
  - No main urls.py
  - No settings.py

```
- myapp2/
  - tests/
    - __init__.py
    - factories.py
    - forms_tests.py
    - models_tests.py
    - urls.py
    - runtests.py
  - integration_tests/
    - __init__.py
    - views_tests.py
```

REUSABLE APP LAYOUT

```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
    url(r'^$', include('myapp2.urls')),
)
```

- LICENSE
- MANIFEST.in
- README.rst
- requirements.txt
- setup.py
- **myapp2/**
  - __init__.py
  - models.py
  - urls.py
  - views.py
  - **templates/**

- __init__.py
- factories.py
- forms_tests.py
- models_tests.py
- **urls.py**
- runtests.py
- **integration_tests/**
  - __init__.py
  - views_tests.py

runtests.py (1/2)

```
from django.conf import settings

if not settings.configured:
    settings.configure(
        DATABASES={
            "default": {
                "ENGINE": "django.db.backends.sqlite3",
                "NAME": ":memory:",
            }
        },
        INSTALLED_APPS=[ ..., 'myapp2' ],
        ROOT_URLCONF='myapp2.tests.urls',
        TEMPLATE_DIRS=(
            os.path.join(os.path.dirname(__file__), '../templates'),
        ),
        COVERAGE_MODULE_EXCLUDES=[ ... ],
        COVERAGE_REPORT_HTML_OUTPUT_DIR=os.path.join(
            os.path.dirname(__file__), 'coverage'),
        [ ... ]
    )
```

runtests.py (2/2)

```
from django_coverage.coverage_runner import CoverageRunner
from django_nose import NoseTestSuiteRunner

class NoseCoverageTestRunner(CoverageRunner, NoseTestSuiteRunner):
    pass

def runtests(*test_args):
    failures = NoseCoverageTestRunner(verbosity=2,
        interactive=True).run_tests(test_args)
    sys.exit(failures)

if __name__ == '__main__':
    runtests(*sys.argv[1:])
```

TRAVIS-CI.ORG

- Host your reusable app on GitHub
- Create service hook for Travis-CI
- Create .travis.yml file in project root

```
language: python
python:
  - "2.6"
  - "2.7"
install: pip install -r requirements.txt --use-mirrors
script: python myapp2/tests/runtests.py
```

HOW TO TEST **JAVASCRIPT**

- Use django-jasmine
  (https://github.com/Fandekasp/django-jasmine)

- Write tests with jasmine and jasmine-jquery
  (http://pivotal.github.com/jasmine/)
  (https://github.com/velesin/jasmine-jquery)

- Create one test that calls /jasmine/ via Selenium

```
class JasmineSeleniumTests(LiveServerTestCase):
    [ ... ]
    def test_login(self):
        self.selenium.get('%s%s' % (self.live_server_url, '/jasmine/'))
        result = self.selenium.find_element_by_class_name('description')
        self.assertTrue('0 failures' in result)
```

RUN, RUN, **RUN**

- Execute your unit tests on each file save
- Watchdog is a good cross platform file system watcher (https://github.com/gorakhargosh/watchdog)

```
#!/bin/bash
watchmedo shell-command --recursive --ignore-directories --patterns="*.py" --wait --command='fab test' .
```

I CAN HAZ **FIXTURES?**

- Fixtures can still be useful
- Provide bootstrap fixtures
- Create Fabric tasks to dumpdata and loaddata

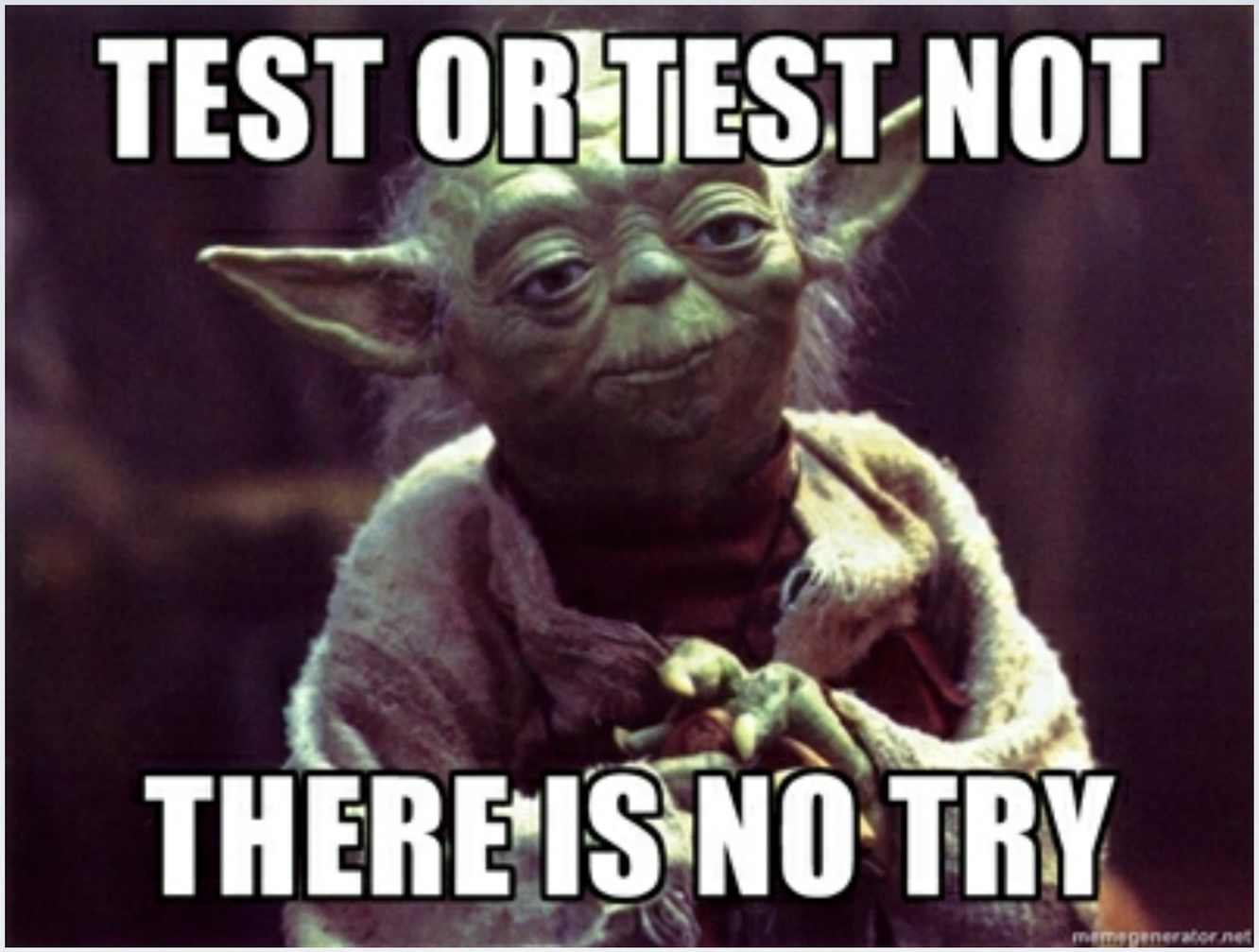
```
def dumpdata():
    local('./manage.py dumpdata --indent 4 --natural auth --exclude auth.permission > myproject/fixtures/bootstrap_auth.json')
    local('./manage.py dumpdata --indent 4 --natural myapp > myapp/fixtures/bootstrap.json')

def loaddata():
    local('python2.7 manage.py loaddata bootstrap_auth.json')
    local('python2.7 manage.py loaddata bootstrap.json')

def rebuild():
    local('python2.7 manage.py reset_db --router=default --noinput')
    local('python2.7 manage.py syncdb --all --noinput')
    local('python2.7 manage.py migrate --fake')
    loaddata()
```

MEDIA FIXTURES

- Create a test_media folder
- On **fab rebuild**:
  - delete MEDIA_ROOT
  - copy test_media to MEDIA_ROOT
  - ./manage.py collectstatic
- Also use these fixtures in your unit tests



THANK YOU!

(https://github.com/mbrochh/tdd-with-django-reusable-app)

(https://github.com/mbrochh/tdd-with-django-project)

@mbrochh