# Factory Boy Fun

Adam Johnson - me@adamj.eu

9th September 2014

# What the problem?

- You've done this, right?

```
# tests/test_something.py
class MyTests(TestCase):
    fixtures = ['basic.json']
    def setUp(self):
        self.user = User.objects.create(
            username='adam',
            first_name='Adam',
            last_name='Johnson',
            email='adam@example.com'
        )
        # ...
    # (and some tests, I hope!)
```

```
# tests/test_something.py
class MyTests(TestCase):
    fixtures = ['basic.json']
    def setUp(self):
        self.user = User.objects.create(
            username='adam',
            first_name='Adam',
            last_name='Johnson',
            email='adam@example.com'
        )
```

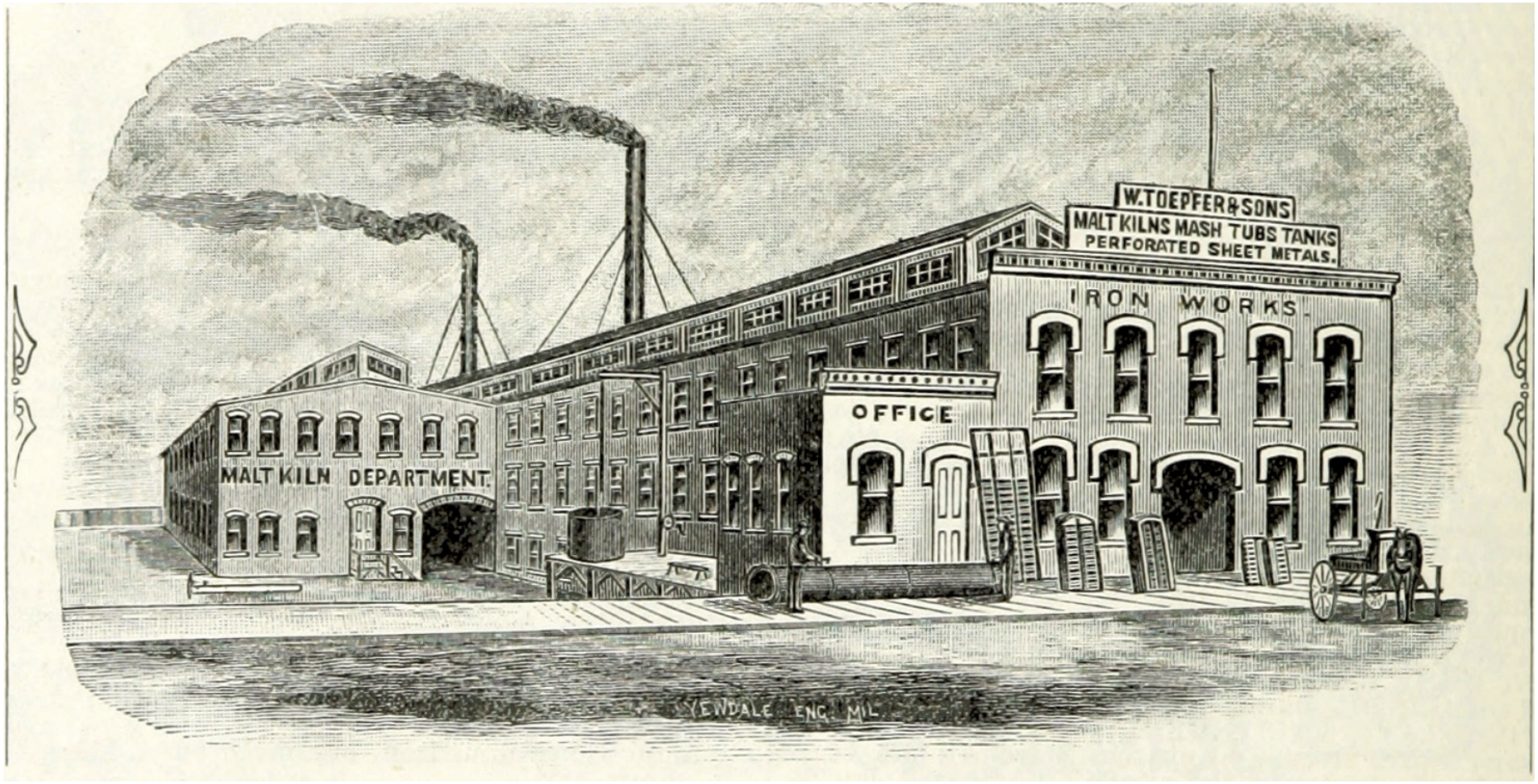
**Pain...**

- Test data in two places!
- A quirky json file that has to be maintained separately!
- I *just* want a User but I have to give *every* detail!

## Model-building can overtake testing

- Let's fix that... with a package ported from Ruby on Rails. (Trust me, it's gonna be okay!)

# Factories to the rescue!



# Factory Boy

- "A fixtures replacement based on thoughtbot's 'factory_girl'."

- http://factoryboy.readthedocs.org/en/latest/

# Example factory

- Creates auth.User instances

```
class User(DjangoModelFactory):
    class Meta:
        model = 'auth.User'
        django_get_or_create = ('username',)

    first_name = 'Adam'
    last_name = 'Johnson'

    @lazy_attribute
    def username(self):
        return slugify(self.first_name + '.' +
                       self.last_name))

    @lazy_attribute
    def email(self):
        return self.username + "@example.com")

    @lazy_attribute
    def date_joined(self):
        return now() - timedelta(days=randint(5, 50))

    @lazy_attribute
    def last_login(self):
        return self.date_joined + dt.timedelta(days=4))
```

# Usage

```
class MyTests(TestCase):
    def setUp(self):
        # A single user, adam.johnson
        self.user = factories.User()

        # Another user, adam.smith
        self.user2 = factories.User(last_name='Smith')

        # A list of 10 users
        # (actually get_or_create -> 10 x adam.johnson)
        self.users = factories.User.create_batch(10)
```

# Adding fuzziness

# Adding fuzziness with faker

- A suite of functions for generating random test data

- http://www.joke2k.net/faker/

```
import faker

# separate to a factory boy Factory
faker = faker.Factory.create()

class User(DjangoModelFactory):
    first_name = lazy(lambda o: faker.first_name())
    last_name = lazy(lambda o: faker.last_name())
    # ...
```

# Controlling the randomness

- Great for testing multiple paths
- But need control - if a test fails due to the randomness, how do we re-run it?
- For nose, install "nose-randomize". Prints random seed and lets you control it via "–with-seed".

# One-to-many relationships

- No built-in structure for navigating in particular
- But flexible "post_generation" hook lets you add code

```
class Product(DjangoModelFactory):
    class Meta:
        model = 'app.Product'

    @lazy_attribute
    def name(self):
        return "Book by " + faker.name()

    @lazy_attribute
    def price(self):
        return Decimal(randint(5, 100))
```

```python
class ProductGroup(DjangoModelFactory):
    class Meta:
        model = 'app.ProductGroup'

    @lazy_attribute
    def name(self):
        return "Products from " + faker.company()

    @post_generation
    def products(self, create, count, **kwargs):
        if count is None:
            count = 3

        make_product = getattr(Product,
            'create' if create else 'build')
        products = [make_product(group=self)
                    for i in range(count)]

        if not create:
            # Fiddle with django internals so
            # self.product_set.all() works with build()
            self._prefetched_objects_cache = \
                {'product': products}
```

# Usage

```
class MyTests(TestCase):
    def setUp(self):
        # A group of 10
        self.g1 = factories.ProductGroup(products=10)
```

## Problem solved?

- Helped us make terser, clearer tests
- Lower maintenance
- Useful also for populating local database with example data

# Thank you

- me@adamj.eu