factory_boy

Konrad Hałas
@konradhalas
konradhalas.blogspot.com

Django fixtures

```
[
  {
    "pk": 1,
    "model": "auth.user",
    "fields": {
      "username": "john",
      "first_name": "John",
      "last_name": "Lennon",
      "is_active": true,
      "is_superuser": false,
      "is_staff": false,
      "last_login": "2012-04-15T14:13:43.304Z",
      "groups": [],
      "user_permissions": [],
      "password": "",
      "email": "john@beatles.com",
      "date_joined": "2012-04-15T14:13:43.304Z"
    }
  }
]
```

users.json

Django fixtures

```python
class ExampleTest(TestCase):
    fixtures = ['users.json']

    def test_sth(self):
        user = User.objects.get(username='john')
        (...)
```

tests.py

Wady Django fixtures

- trudne w utrzymaniu
- zwiększają powiązanie testów
- wolne (baza danych)
- w innym miejscu niż testy
- JSON!

Przykładowy model

```python
class Post(models.Model):
    author = models.ForeignKey(User, related_name='posts')
    title = models.CharField(max_length=60)
    body = models.TextField()
    pub_date = models.DateTimeField(auto_now=True)
    rating = models.IntegerField()
```

models.py

Rozwiązanie 0 - ręczne tworzenie

```python
class PostTest(TestCase):

    def test_is_good(self):
        user = User.objects.create(username='john', password='johnpass')
        post = Post.objects.create(author=user, title='Post title', body='Post body', rating=10)
        self.assertTrue(post.is_good)
```

tests.py

Rozwiązanie 1 - własna fabryka

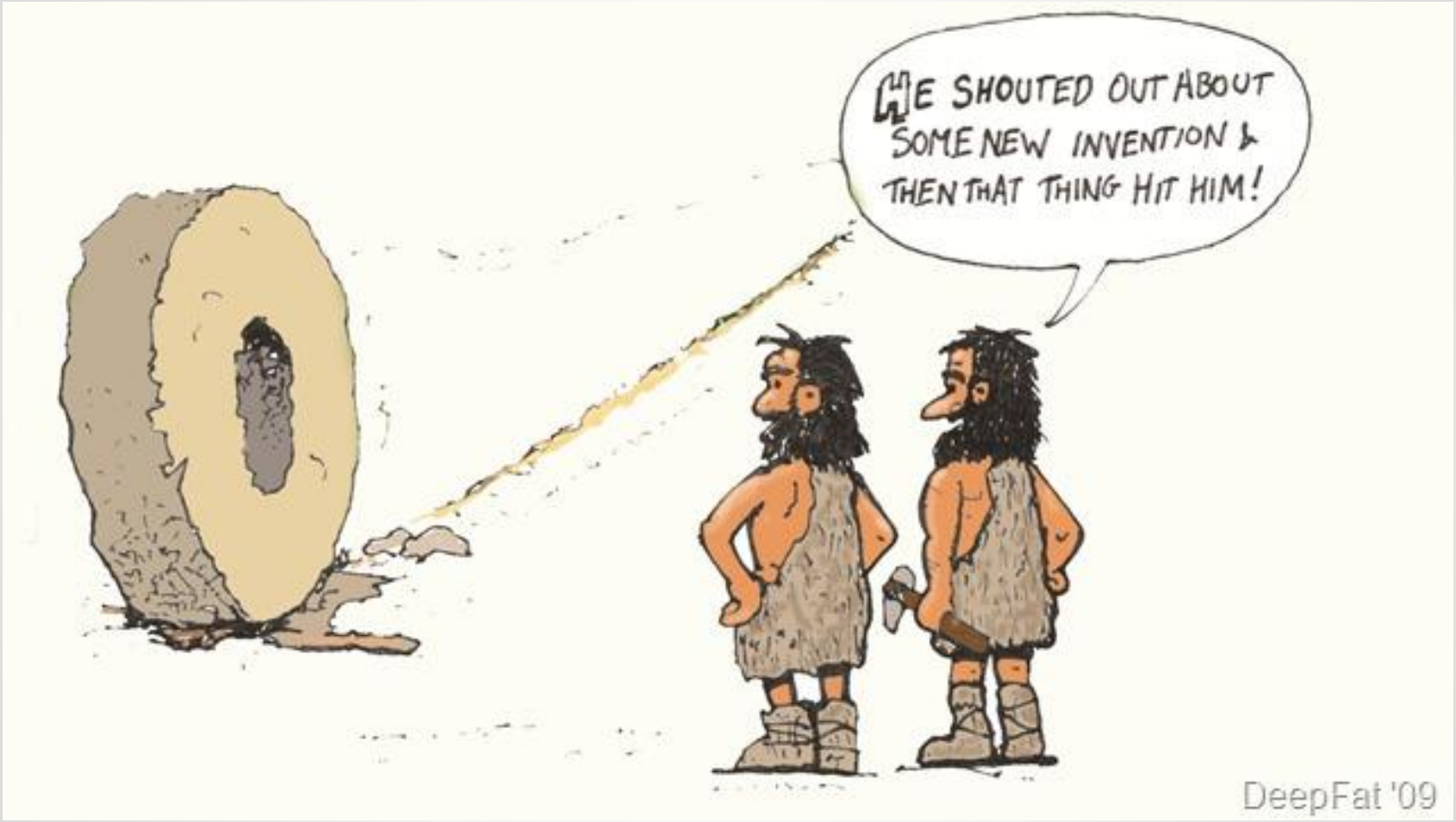
```python
def create_post(**kwargs):
    defaults = {
        'title': 'Post title',
        'body': 'Lorem ipsum dolor sit amet',
        'rating': 5
    }
    defaults.update(kwargs)
    if 'author' not in kwargs:
        defaults['author'] = create_user()
    return Post.objects.create(**defaults)
```

factories.py

Rozwiązanie 1 - własna fabryka

```
1 class PostTest(TestCase):
2
3     def test_is_good(self):
4         post = create_post(rating=10)
5         self.assertTrue(post.is_good)
```

tests.py


HE SHOUTED OUT ABOUT SOME NEW INVENTION & THEN THAT THING HIT HIM!
DeepFat '09

Rozwiązanie 2 - factory_boy

```python
class PostFactory(factory.Factory):
    FACTORY_FOR = Post

    title = 'Post title'
    body = 'Lorem ipsum dolor sit amet'
    rating = 5
    author = factory.SubFactory(UserFactory)
```

factories.py

Rozwiązanie 2 - factory_boy

```python
class PostTest(TestCase):

    def test_is_good(self):
        post = PostFactory.build(rating=1, author__username='khalas')
        self.assertTrue(post.is_good)
```

tests.py

factory_boy - tworzenie obiektów

```
1 PostFactory.build()
2 PostFactory.create()
3 PostFactory.attributes()
4 PostFactory.stub()
```

factory_boy - LazyAttribute

```python
class UserFactory(factory.Factory):
    FACTORY_FOR = User

    first_name = 'John'
    last_name = 'Lennon'
    email = factory.LazyAttribute(lambda u: '{0}.{1}@email.com'.format(u.first_name, u.last_name))
```

factories.py

factory_boy - Sequence

```python
class UserFactory(factory.Factory):
    FACTORY_FOR = User

    username = factory.Sequence(lambda n: 'user-{0}'.format(n))
```

factories.py

factory_boy - oraz...

- post generation hook
- rozszerzanie mechanizmu tworzenia obiektu
- zmiana strategii budowanie obiektu

factory_boy - zalety

- łatwe w zarządzaniu
- lokalne
- kontrola zapisów do bazy danych
- Python!

Dziękuję!

factory_boy

- instalacja: `pip install factory_boy`
- github: github.com/dnerdy/factory_boy
- docs: readthedocs.org/docs/factoryboy

Inspiracja:

Carl Meyer, PyCon US 2012, *Testing and Django*

video: pyvideo.org/video/699/testing-and-django