

Почему стоит понижать культуру производства

На что обращать внимание, создавая приложения для Django

Дмитрий Лебедев

<http://course.ryba4.com>

skype: siberianoNsk

+7 923 732 1337



<http://www.devconf.ru>

Культура производства

“Программист должен не забыть. Пусть будет высокая культура производства.”

Культура производства

совокупность нормативных требований к технико-экономическому, организационному и эстетическому уровню производства.



solarledwind.en.alibaba.com

- С защитным кожухом гильотина требует меньшей культуры производства от рабочего.
- Там, где не вводят мер, облегчающих работу, случаются ошибки.
- В программировании эти ошибки не фатальны, и мы продолжаем их совершать.

Примеры завышенной культуры производства

- Что должно быть синхронным, лежит далеко
- Слово “прописывать”
- Огромное количество настроек, которые надо запомнить
- Сложные протоколы

Пример №1. Документация

Ещё один пример данных, которые **должны** быть синхронными.

Код – Документация

Пример №1. Документация

- Недостатки документации в Wiki
 - Неохота исправлять
 - Неточности незаметны
 - Размыта ответственность
 - Не интегрируется с IDE или консолью

Docstrings

```
In [65]: zip?
```

```
Type:      builtin_function_or_method
```

```
Base Class: <type 'builtin_function_or_method'>
```

```
String Form: <built-in function zip>
```

```
Namespace: Python builtin
```

```
Docstring:
```

```
zip(seq1 [, seq2 [...]]) -> [(seq1[0], seq2[0] ...), (...)]
```

Return a list of tuples, where each tuple contains the i-th element from each of the argument sequences. The returned list is truncated in length to the length of the shortest argument sequence.

Docstrings

- Вместе с кодом
 - Исправить — дело нескольких секунд
 - Неточности заметны сразу
 - Ответственность — на авторе, проверяется на обзоре кода
 - Легко получить документацию из консоли или IDE
- Сделать документацию в виде HTML можно при помощи инструментов
- В Wiki храним то, что никуда не пристраивается

Антипример №1, Debug Toolbar

```
INSTALLED_APPS = (  
    ...  
    'debug_toolbar',
```

Антипример №1, Debug Toolbar

```
MIDDLEWARE_CLASSES = (  
    ...  
    'debug_toolbar.middleware.DebugToolbarMiddleware',  
    ...  
)
```

Антипример №1, Debug Toolbar

```
INTERNAL_IPS = ('127.0.0.1',)
```

Сборка проекта

```
$ hg clone ssh://bitbucket.org/siberiano/course.ryba4.com course
$ cd course
$ make run
...
Development server is running at http://0.0.0.0:8001/
Quit the server with CONTROL-C.
...
$ make rebuild
Rebuilding the database...
```

Сборка проекта

- Большая инструкция приводит к ошибкам
- Обновившиеся зависимости — это отрезанные пальцы

Сборка проекта

- Нужен инструмент сборки, чтобы
 - собрать зависимости проекта на новом месте
 - развернуть базу данных
 - пересоздать базу данных
 - зафиксировать версии зависимостей
- Существующие решения:
 - django-fab-deploy
 - fastdev-django
 - самодельное решение на virtualenv или bootstrap.py

Пример №2, fixtures

- Файлы fixtures нужно положить по таким путям:

apps/polls/dev_fixtures/initial_data.json

apps/messages/dev_fixtures/initial_data.json

- В настройках придется указать

```
FIXTURE_DIRS = (  
    'apps/polls/dev_fixtures/',  
    'apps/messages/dev_fixtures/',  
)
```

Пример №2, fixtures

- Пути этих файлов строгие и уже содержат всю необходимую информацию, чтобы их найти.
- Решение проблемы:

```
FIXTURES_DIRS = tuple(ln.rtrim() for ln in os.popen('find . -path  
"*apps*dev_fixtures"))
```

Пример №3, спрайты



```
.folder.opened {
  width: 16px;
  height: 16px;
  background-image: url("icons.gif");
  background-position: -64px -16px;
}
```



Пример №3, спрайты

Настройки для приложения, делающего спрайты:

```
sprite=Sprite.create_from_local_files(['/path/to/first/file',  
'/path/to/second/file'])
```

Пример №3, спрайты

```
# settings.py  
SPRITES_DIR = 'static/img/sprites'
```

```
sprites/  
  sprite_one/  
    icon1.png  
    icon2.png  
  sprite_two/  
    icon3.png  
    icon4.png
```

Пример №4, urls и javascript

Статический файл js:

```
$.ajax({url: '/path/to/API/', ...})
```

Urls.py:

```
url(r'^api/$', api_view, name='api')
```

Связка разваливается, когда изменяют одну из этих 2 строк.

Пример №4, urls и javascript

Функция в JS:

```
reverse_url = function(url_name, params){  
    params = params || {};  
    $.extend(params, {url_name: url_name})  
    return '/reverse_url/?' + $.param(params);  
};
```

Пример №4, urls и javascript

Middleware:

```
class UrlReverseMiddleware(object):
    def process_request(self, request):
        if request.path_info == '/reverse_url/':
            query_dict = request.GET.copy()
            url_name = query_dict.pop('url_name')[0]

            args = query_dict.pop('args', [])
            if args == [""]:
                args = []
```


Пример №4, urls и javascript

try:

```
    request.path = request.path_info = reverse(url_name,  
args=args)
```

except NoReverseMatch:

```
    raise Http404
```

```
request.GET = QueryDict(query_dict.urlencode())
```

Пример №4, urls и javascript

Использование:

```
reverse_url('complicated_url', {args: [1, 2], param1: 3});  
/reverse_url/?url_name=complicated_url&args=[1, 2]&param1=3
```

Преимущества:

- Адрес можно перемещать
- Можно искать использование адреса по его имени

Пример №5, протоколы

Так можно:

```
def view1(request, arg1, arg2):
```

```
    pass
```

```
def view2(request, arg1, arg2):
```

```
    pass
```

Пример №5, протоколы

Такого стоит избегать:

```
data ={  
    'location': place.name,  
    'date': visit.date,  
    'event': 'visit',  
    'type_of_' + place.__class__.__name__: place.place_type,  
}
```

- Изменения всегда нужно делать в паре
- Не отлаживается статическим анализатором
- Не отлаживается дебаггером

Было переписано так:

```
data = {  
    'event': visit, # объект класса Event  
}
```

visit.place — место

visit.user — человек

Пример №6, меню приложений

- Много файлов `apps/app/menu.html`
- Много файлов `templates/app/url.py`

```
<div class="submenu">
  <a href="{% url photos %}">Фотоальбом</a>
  <a href="{% url calendar %}">Календарь</a>
  {% if request.user.is_authenticated() %}
    <a href="{% url inbox %}">Входящие сообщения
      {% if request.user.new_msg %}
        <b>({{ request.user.new_msg }} новых)</b>
      {% endif %}
    </a>
  {% endif %}
  {% if request.user.is_authenticated() %}
    <a href="{% url create_request %}">
      Подать заявку</a>
  {% endif %}
</div>
```


Сложная вёрстка повторяется в шаблонах меню

```
urlpatterns = patterns('views',  
    item(url(r'^photos$', 'photos', name='sport_photos'),  
        caption='Фотографии'),  
  
    item(url(r'^calendar$', 'calendar', name='sport_calendar'),  
        caption='Календарь'),  
  
    item(url(r'^inbox$', 'inbox', name='sport_inbox'),  
        caption='Входящие', template='sport/menu_inbox.html'),  
  
    item(url(r'^request$', 'create', name='sport_create_request'),  
        caption='Подать заявку'),  
  
    url(r'^ajax_request$', 'ajax', name='sport_ajax'),  
)
```

```
<div class="submenu">  
  {% for item in submenu %} {# права уже проверены #}  
    {{ item }}  
  {% endfor %}  
</div>
```

Пример №6, меню приложений, ИТОГИ

Было	Стало
<ul style="list-style-type: none">.Много шаблонов с меню.Много файлов с url.Проверка прав доступа в view.Проверка прав доступа в шаблонах	<ul style="list-style-type: none">.Много файлов с url.Проверка прав доступа в view.Функция регистрации пункта меню.Функция выдачи меню в контекст

Как можно делать новое приложение

- Проверить, нет ли готовых решений
- Определить
 - кто им пользуется из разработчиков
 - как ему удобнее им пользоваться
 - какие данные (шаблоны, статические файлы, адреса) потребуются
- Поместить минимальный набор данных на нужные места
- Написать приложение под эти данные

Резюме. Принципы.

- Не делайте ненужных настроек
- Настройки должны быть там, где ими пользуются
- Документация — в коде
- Проект должен собираться автоматически