Здравствуйте!

Отправляю домашнее задание.
Для меня это был очень полезный курс, потому что он перевернул моё представление о том как нужно писать программы. Без курса вряд ли в ближайшее время я сам смог бы глубоко погрузиться в тему паттернов, за что Вам большое спасибо, Леонид!

Итак, вот такие антипаттерны я нашел в своем проекте:
- Magic Number

- Hard Code

- Copy-Paste

- Golden hammer

- Big ball of mud

- Mutillation

- Reinventing the wheel

Наверняка, можно найти еще больше, но я выделю эти особенно и объясню почему.

Магические числа
в модуле registry_import.py:

```
def save_data_to_db(self, record):
    """save every data record to DB using RegistryRecordAdapter"""
    args = {
        'date_created': record['date_create'],
        'title': record['fio']+'-'+record['vid_d']+'-'+record['stamp']+'-'+record['date_create'],
        'org': record['company'],
        'typeof': 'Аттестация персонала',
        'params': json.dumps(record),
        'status': 0
    }
```

Устанавливаю статус 0, который вряд ли понятен кому-то кроме меня. Ну и, наверняка, неправильно устанавливать с помощью адаптера конкретные значения полей (typeof — Аттестация персонала) при записи в базу данных, потому что в дальнейшем, чтобы изменить эти устанавливаемые адаптером значения, нужно будет по всему коду вносить изменения и тратить на это время. Нужно будет продумать систему конфигурирования адаптеров. Думаю, можно попробовать, применив паттерн строитель собирать, разные адаптеры под разные нужды. Эта функциональность потребуется для импорта не только данных об аттестации персонала, но еще и для направлений аттестации технологий, оборудования и материалов.

Хардкод
Хардкод ссылки для импорта данных в модуле registry_import.py:

```
data_url = 'http://ac.naks.ru/curl/json.php?url=reestr_personal&token=NtdRAUoEtsiwrew73UWyASw0wsYa&type=personal&popov=Y'
```

Так как я делаю типовой сайт аттестационного центра, который будет загружать разные данные для разных аттестационных центров (сайтов будет много), то нужно сделать настройку токенов для разных подсистем импорта и для разных АЦ.

Вообще хардкода много:

```
<a class="main-content__org_info--attention" href="{{ menu_urls.1.url }}#page_details_top"><i class="fas fa-exclamation-circ
    для заявителей
    <div>Профессиональные стандарты в области сварки</div>
</a>

<a class="main-content__org_info--org-activities" href="{{ menu_urls.2.url }}#page_details_top">Аттестация специалистов свар
    производства</a>
<a class="main-content__org_info--org-activities" href="{{ menu_urls.3.url }}#page_details_top">Аттестация сварщиков</a>
<a class="main-content__org_info--org-activities" href="{{ menu_urls.4.url }}#page_details_top">Аттестация сварочного оборуд
<a class="main-content__org_info--org-activities" href="{{ menu_urls.5.url }}#page_details_top">Аттестация сварочных техноло
<a class="main-content__org_info--org-activities" href="{{ menu_urls.6.url }}#page_details_top">Оценка квалификации в област
```

Тут я захрдкодил ссылки на пункты меню на главной странице. Пока не придумал как сделать более гибкую и настраиваемую систему. Что пойдет в первую ссылку, что во вторую понятно только мне. Очень жесткая система, которая, конечно, должна быть переделана. Скорее всего надо будет переделать сам принцип формирования ссылок на главной странице — выводить из БД ссылки, наименования пунктов, управлять последовательностью пунктов меню.

Велосипед, костыль и soft-code

Изобретение велосипеда и сложная для понимания конструкция в mainapp.views для формирования списка новостей с картинками:

```
post_list = [dict({'post': post, 'picture': PostPhoto.objects.filter(
    post__pk=post.pk).first()}) for post in all_news]
```

Когда я создавал модели я не продумал вариант, при котором фотографии будут нужны в заголовке новости. Однако, я создал модель для фотографии, которая связывается внешним ключом с новостью. Сделал я это для большей гибкости — чтобы можно было фотографии использовать не только в тексте новостей или статей, но еще и для того, чтобы фотографии можно было использовать, например, в галерее на сайте. В итоге создал сам себе проблем тем, что приходится формировать вот такой массив из словарей, в котором для каждого поста выбирается соответствующая ему фотография. Возможно, это еще и пример реализации антипаттерна soft code — моё стремление сделать систему более гибкой порождает дополнительные проблемы и заставляет меня придумывать костыли.

Комок грязи и copy-paste

Попытку прикрутить доменную модель считаю полностью проваленной. Я использовал антипаттерн copy-paste, вместо того, чтобы начать с внедрения идей и концепций доменной модели моей предметной области и понемногу наращивать функциональность, проверяя ее работоспособность. Вместо этого я потратил время на попытку связать готовый код со своим и упоролся в итоге в стену.

```
class Field(models.Field):
    """base class for custom fields"""

    def __init__(self, *args, **kwargs):
        kwargs['max_length'] = 100
        super().__init__(*args, **kwargs)

    def bind_name(self, name):
        self.name = name
        self.storage_name = ''.join(('_', self.name))
        return self
```

Неправильной идеей было унаследоваться от model.Field, чтобы создать метакласс для моих полей, неправильной идеей было изобрести велосипед с хранилищем объектов с помощью модуля django-pickle - скорее всего надо было пробовать встроенный в джанго JSONField. В целом это такой комок грязи, который можно исправить только если переписать с нуля.

Золотой молоток

Мне настолько понравилась идея фабричного метода, что я начал ее использовать везде где только можно. Не знаю насколько это считается правильным подходом, но чувствую, что фабричный метод станет для меня тем самым золотым молотком.

Фабрика писем с уведомлением пользователей:

```
def send_email(self, message):
    mail_selector = {
        'Заявка': {
            'subject': 'Сообщение получено',
            'message': 'Уважаемый {}, информируем вас, что ваше сообщение \
            получено и передано в работу'
        },
        'Подписка': {
            'subject': 'Спасибо за подписку!',
            'message': '{}, благодарим Вас за подписку'
        },
        'Вопрос': {
            'subject': 'Вопрос получен',
            'message': '{}, мы получили ваш вопрос, в ближайшее время вы \
            получите ответ'
        }
    }
```

Фабрика заголовков страницы, фабрика форм:

```
form_name_select = {
    'post': 'новость',
    'article': 'статью',
    'document': 'документ'
}
title = 'Создать {}'.format(form_name_select[content_type])

forms = {
    'post': PostForm,
    'article': ArticleForm,
    'document': DocumentForm
}

if request.method == "POST":

    form_Class = forms[content_type]
```

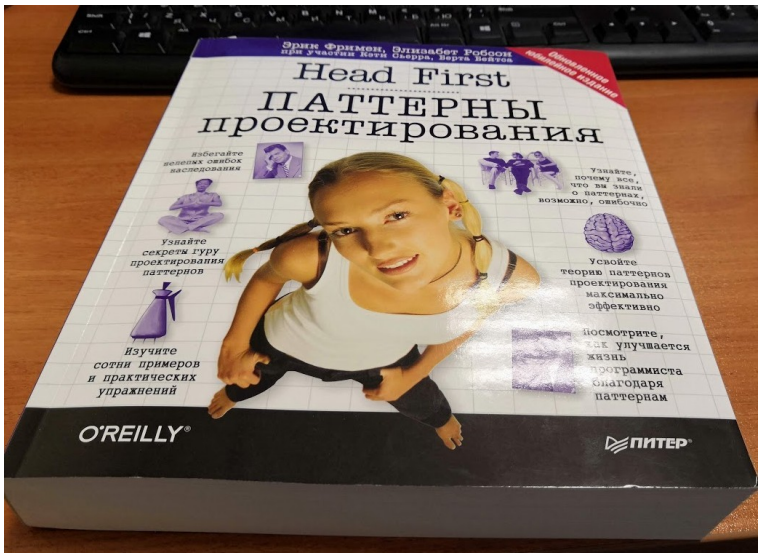
Излишнее затачивание под определенную задачу - антипаттерн Mutilation

В моем проекте есть необходимость вывода детальной информации объекта Post, который как и объект Article наследуется от ContentMixin класса. В попытке создать универсальный контроллер детального вида объектов Post и Article мне пришлось слишком много времени потратить на адаптацию кода просмотра страницы новостей под просмотр статей. Излишнее затачивание под определенную задачу в итоге привело к тому, что мне не нравится результат ни того ни другого. Надо было просто унаследовать вид статьи от новости и сделать небольшие изменения в обоих вариантах.

```
if content == 'post':
    attached_images = PostPhoto.objects.filter(post__pk=pk)
    attached_documents = Document.objects.filter(post__pk=pk)
    post_content = {
        'post': obj,
        'images': attached_images,
        'documents': attached_documents,
        'bottom_related': Article.objects.all().order_by(
            '-created_date')[:3]
    }
if content == 'article':
    tags_pk_list = [tag.pk for tag in obj.tags.all()]
    related_articles = Article.objects.filter(
        tags__in=tags_pk_list).exclude(pk=pk).distinct()
    post_content = {
        'post': obj,
        'related': related_articles,
        'bottom_related': related_articles.order_by('-created_date')[:3]
    }
```

Я не совсем уверен, но чувствую, что можно было сделать по-другому.

Выводы:
Мой следующий проект совершенно точно будет лишен всех этих недостатков! Никакого хардкода, волшебных чисел и копипасты, только чистый, выверенный, протестированный код, наполненный философией и глубоким пониманием классических и современных паттернов.



← Моя новая настольная книга (до творения банды четырех я еще доберусь)

Еще раз благодарю за курс!