

Stabile Software in volatilen Umgebungen

code.talks

Kore Nordmann (@koredn / @qafoo)
27th April 2017



Hi, I'm Kore (@koredn)



Well Aged Online Shop



Our Project



- ▶ Development worked like a charm
- ▶ Reasons to change something:
 - ▶ New (security) release of shop software
 - ▶ New feature requirements
 - ▶ New scaling requirements (Microservices!11!!)



After just one change

Why Is There a Problem?

- ▶ Life times of code:
 - ▶ Frontend Code (HTML, JavaScript)
 - ▶ Half-life period: **6 Months**
 - ▶ Service Integrations (Payment, Mailing, ...)
 - ▶ Half-life period: **2 Years**
 - ▶ Infrastructure Code (Framework, Database, Logging, ...)
 - ▶ Half-life period: **3 Years**
 - ▶ Domain Code (Basket calculations, discounts, ...)
 - ▶ Half-life period: **10 Years**

Maintenance



Cory House 🏠

@housecor



Folgen

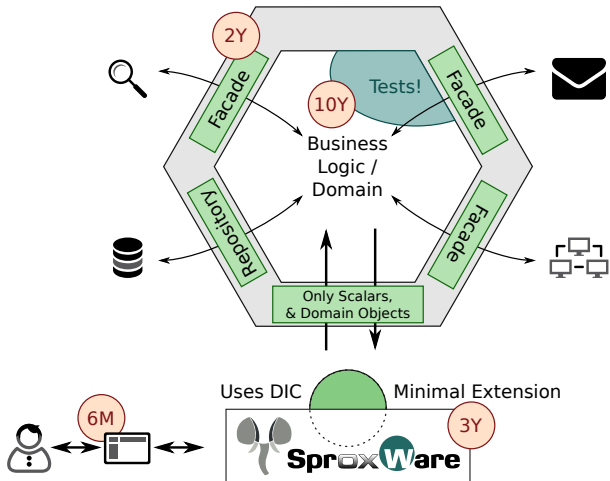


Software maintenance is not "keep it working like before." It is "keep being useful in a changing world".

Why Don't Updates Just Work?

- ▶ Changing APIs in vendor software
- ▶ Side effects (session, global scope, class scope) between “modules”
- ▶ Wrong abstractions – not embracing the next change

Design Stable Modules / Extensions



The Entry Point

- ▶ May be an ugly mess
- ▶ **Must not** contain logic, beside:
 - ▶ Basic input conversion
 - ▶ Exception handling
 - ▶ Basic out preparation
- ▶ Access your Dependency Injection Container (Application configuration) statically, if necessary
- ▶ Do not test with Unit Tests (unless it is a certification requirement)

Oxid Example

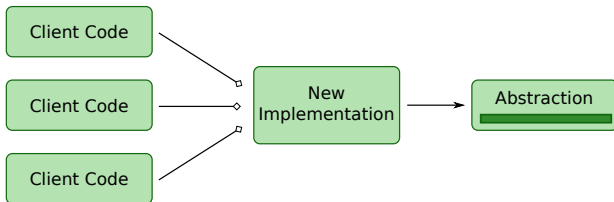
```
1 class acme_dev_magic_magic extends oxAdminDetails
2 {
3   public function render()
4   {
5     parent::render();
6     try {
7       $dic = oxRegistry::get('acme_dic');
8
9       if (isset($_POST['push'])) {
10        $magic = $dic['acme_dev_magic_magic'];
11        $magic->import($this->getLangauge());
12      }
13
14      $configuration = $dic['acme_dev_magic_configuration'];
15      $this->addTplParam('hasUrl', (bool) $configuration->url);
16      $this->addTplParam('hasSecret', (bool) $configuration->secret);
17      $this->addTplParam('isProduction', (bool) oxRegistry::getConfig()->
18        isProductiveMode());
19
20      return 'acme_dev_magic_magic.tpl';
21    } catch (\Exception $e) {
22      $this->addTplParam('exception', $e);
23      return 'acme_dev_magic_error.tpl';
24    }
25 }
```

The Domain

- ▶ Find your agreed-upon patterns:
 - ▶ Separate between “Newables” (Data Objects) and “Injectables” (Services, Transaction Script)
 - ▶ Only add “Eternal Truth” to Newables – all other logic goes into Injectables
 - ▶ No Newable may aggregate an Injectable
 - ▶ No Injectable may aggregate a Newable – only use as parameters

How do we find this Domain in existing code?

Finding The Domain (Branch By Abstraction)

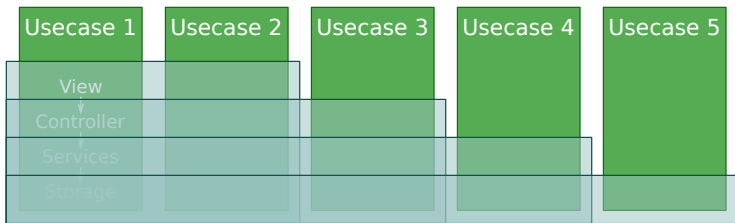


<https://qa.fo/bba>

Do Not Abstract (Embrace Change)

Split vertically instead of horizontally

- ▶ How much code can be re-used for that new user registration through Facebook?



Do **Not** Abstract

- ▶ **Never** let fellow developers come up with technical abstractions
 - ▶ **No** custom Object Relational Mapper
 - ▶ **No** custom Request, Routing
 - ▶ **No** custom Form Handler
 - ▶ **No** custom Configuration Handlers
 - ▶ **No** custom Template Systems
 - ▶ **No** custom Logger
 - ▶ **No** custom ...
- ▶ Use the amazing components which are out there and **tested**

Your developers do not develop SproxWare

Summary

- ▶ Extract a framework independent domain
- ▶ Test your decoupled domain
- ▶ Sensible Domain Driven Design (just) consists of sensible Domain Objects in 99% (no CQRS, no Event Sourcing, ...)
- ▶ Embrace change
- ▶ Define an Extended Definition Of Done with design rules¹

¹https://qafoo.com/blog/097_extended_definition_of_done.html



THANK YOU

Rent a quality expert
qafoo.com