Stabile Software in volatilen Umgebungen

International PHP Conference 2017

Tobias Schlitt (@tobySen) & Kore Nordmann (@koredn)

24th October 2017

Hi, we're Toby and Kore

Tobias Schlitt
@tobySen

Kore Nordmann
@koredn

Frontastic

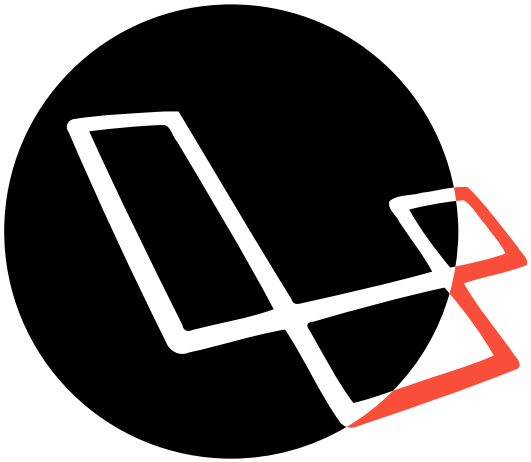
@qafoo

@tideways

Qafoo
passion for software quality


Well Aged Project

Our Project



Symvel Framework

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


After just one change

Why Updates Don’t Just Work?

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

Maintenance

Cory House 🏠
@housecor

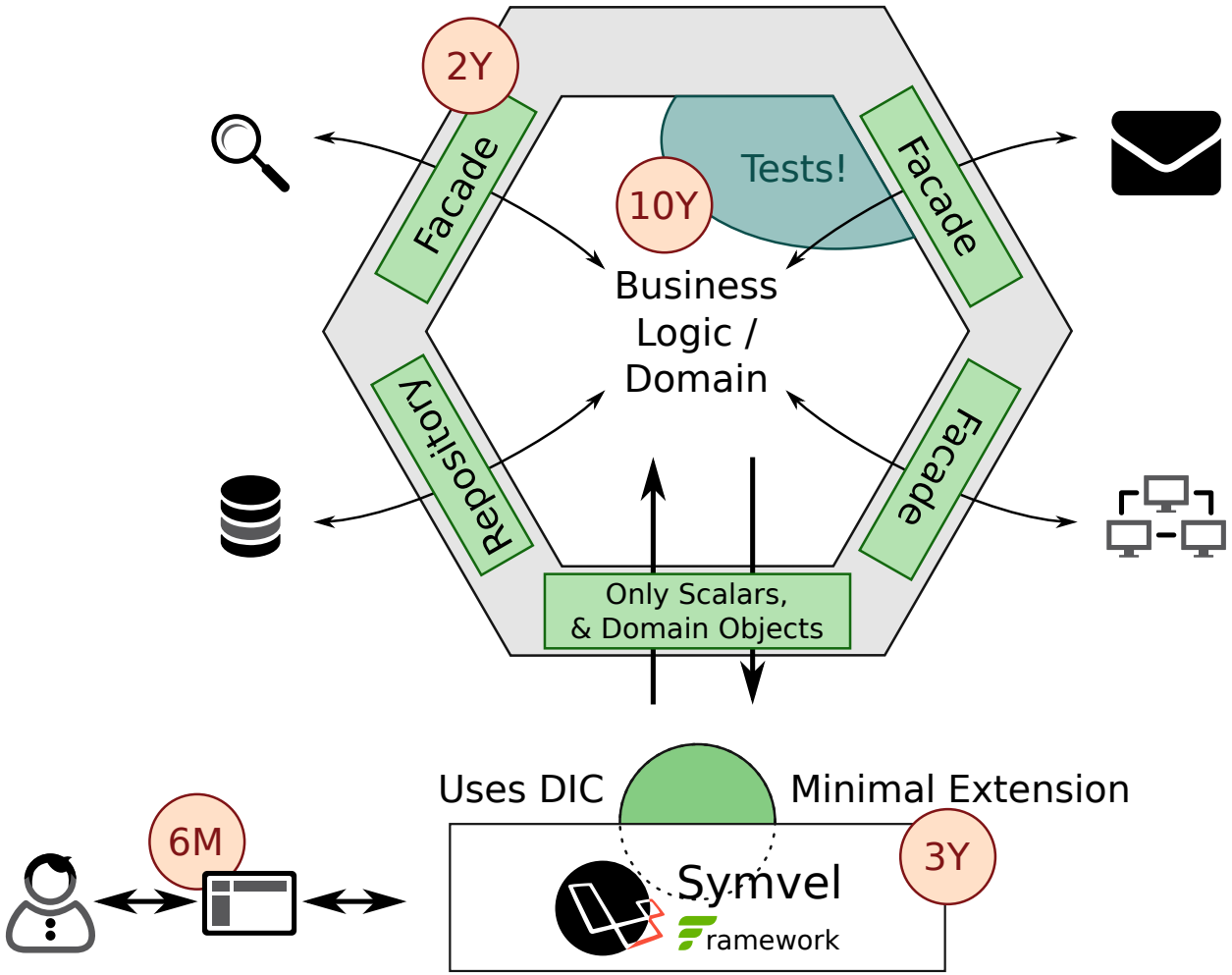
Folgen

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

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**

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

```php
class acme_dev_magic_magic extends oxAdminDetails
{
    public function render()
    {
        parent::render();
        try {
            $dic = oxRegistry::get('acme_dic');

            if (isset($_POST['push'])) {
                $magic = $dic['acme.dev-magic.magic'];
                $magic->import($this->getLangauge());
            }

            $configuration = $dic['acme.dev-magic.configuration'];
            $this->addTplParam('hasUrl', (bool) $configuration->url);
            $this->addTplParam('hasSecret', (bool) $configuration->secret);
            $this->addTplParam('isProduction', (bool) oxRegistry::getConfig()->
                isProductiveMode());

            return 'acme_dev_magic_magic.tpl';
        } catch (\Exception $e) {
            $this->addTplParam('exception', $e);
            return 'acme_dev_magic_error.tpl';
        }
    }
}
```

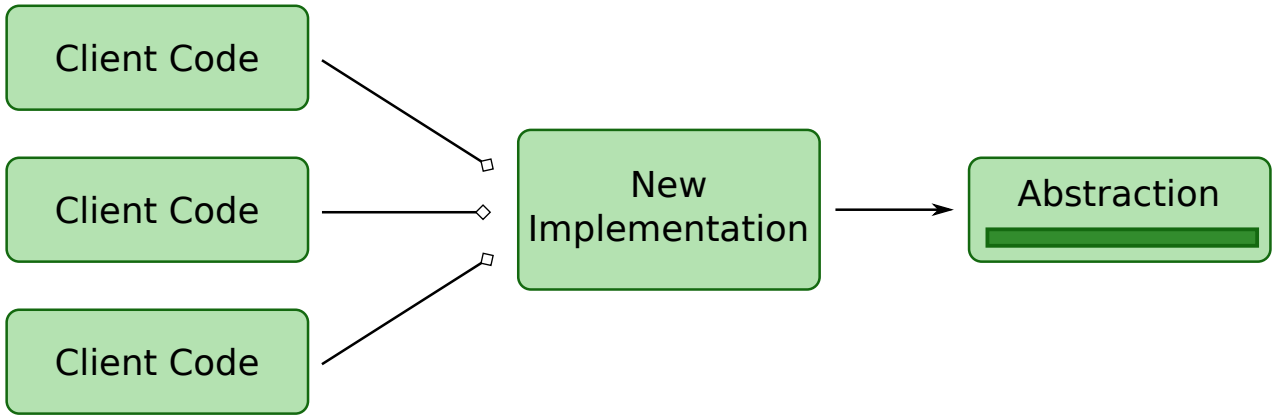
Qafoo
passion for software quality

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 an 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

- **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**

Do Not Abstract

Your developers do not develop Symvel Framework

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 rules[1]

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

THANK YOU

Rent a quality expert
qafoo.com