Refactoring Legacy Code

IPC Spring Edition 2017

Benjamin Eberlei & Kore Nordmann (@qafoo)

2nd June, 2017

Hi

Benjamin Eberlei
@beberlei

Kore Nordmann
@koredn

@qafoo

@tideways

Business Requirements Will Never Converge

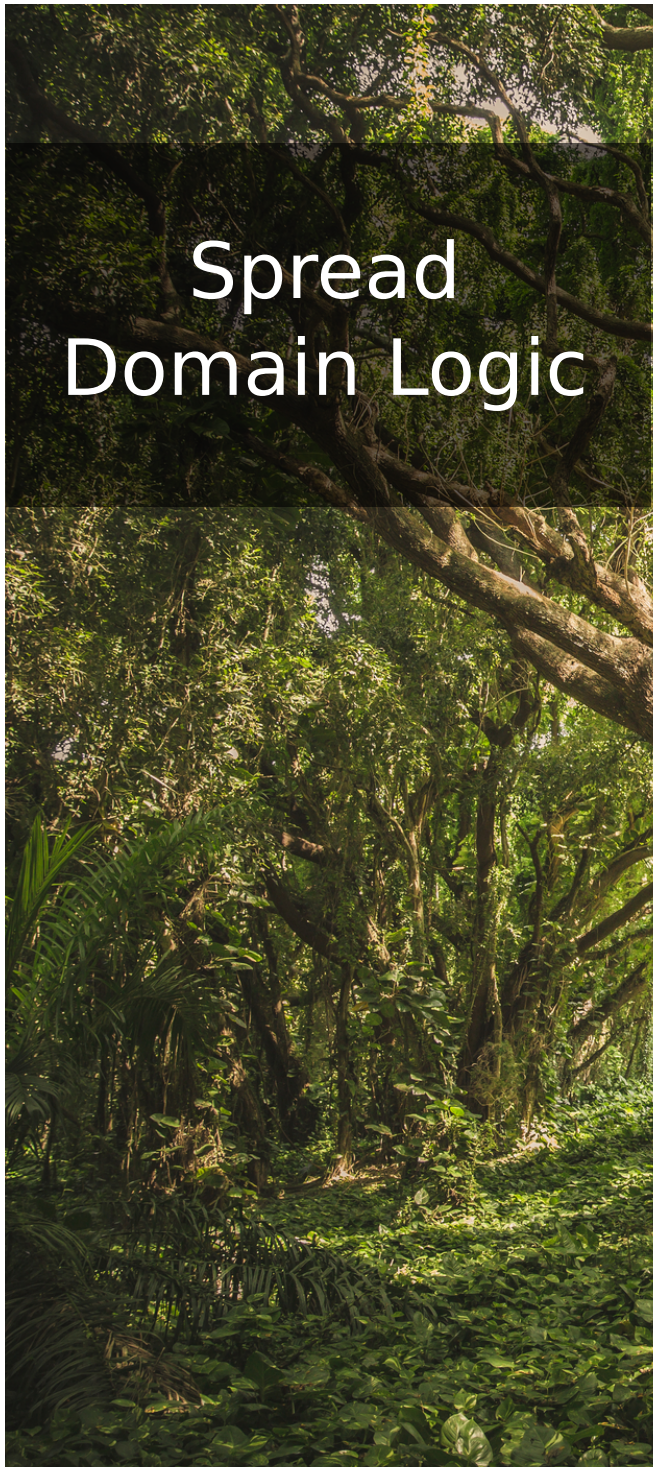
Refactoring

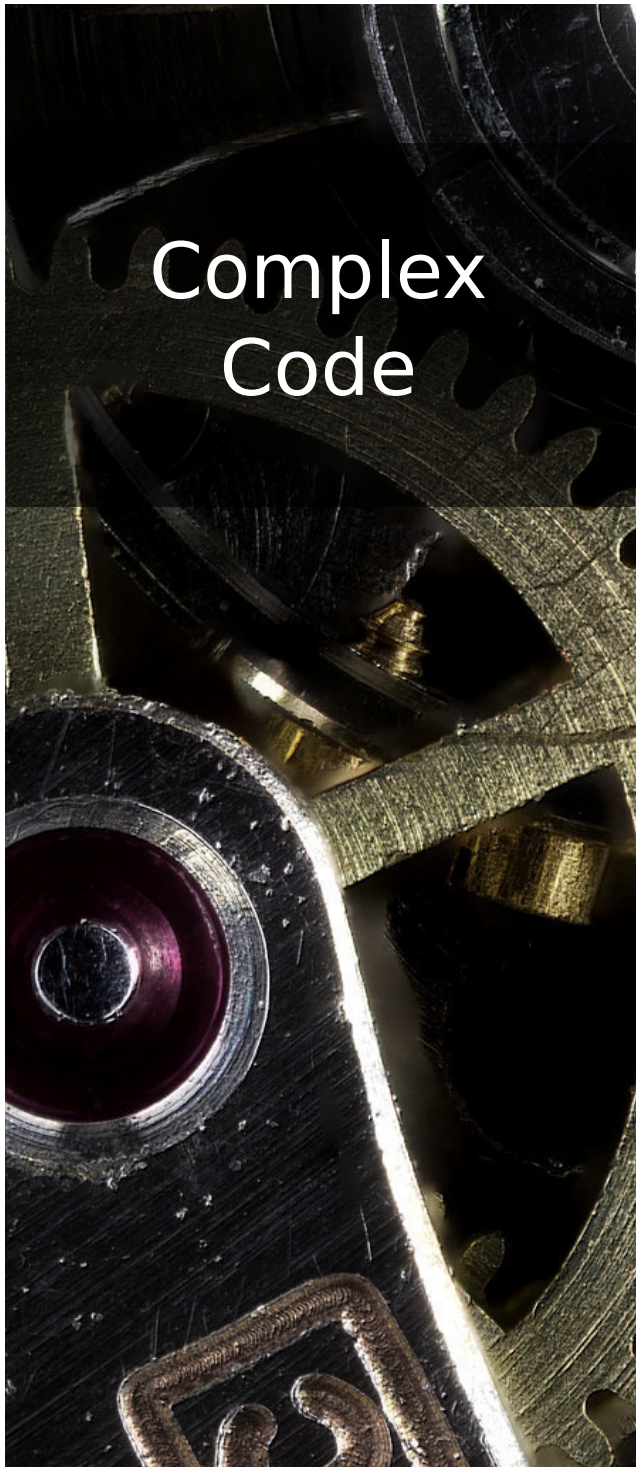
"Change is the only constant"

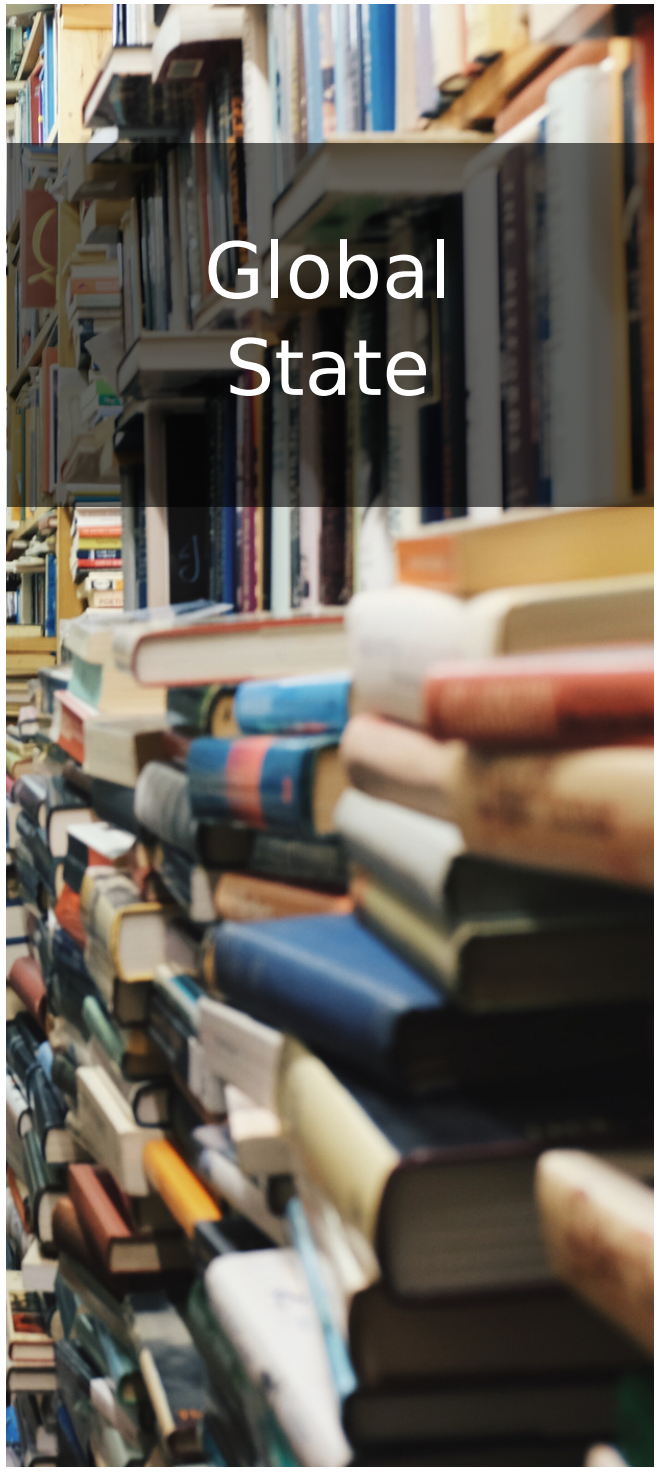
Refactoring

"Refactoring should never only be a dedicated task on your board – it should be an essential part of every other task you work on."

- Refactoring and testing are default behaviours
- Larger refactorings may require dedicated work


Spread
Domain Logic


Complex
Code


Global
State


Spread Domain Logic

Spread Domain Logic

- Reason for complexity
- Problematic to maintain or change
- Hard to understand

The Problem

```
1   class EventController
2   {
3       public function listAction(Request $request)
4       {
5           $start = new \DateTime($request->query->get('start', '-60 minute'));
6           $end = new \DateTime($request->query->get('end', 'now'));
7
8           if ($start > $end) {
9               $tmp = $end;
10              $end = $start;
11              $start = $tmp;
12          }
13
14          return [
15              'events' => $this->eventRepository->findBetween($start, $end),
16          ];
17      }
18
19      public function listTodayAction()
20      {
21          $start = new DateTime('today 00:00:00');
22          $end = new DateTime('today 23:59:59');
23
24          return [
25              'events' => $this->eventRepository->findBetween($start, $end),
26          ];
27      }
28  }
```

The Value Object

```
1   class DateRange
2   {
3       public function __construct(DateTime $start, DateTime $end)
4       {
5           if ($start > $end) {
6               $tmp = $end;
7               $end = $start;
8               $start = $tmp;
9           }
10
11          $this->start = $start;
12          $this->end = $end;
13      }
14
15      public static function today()
16      {
17          return new self(
18              new DateTime('today 00:00:00'),
19              new DateTime('today 23:59:59')
20          );
21      }
22
23      // @TODO: More utility factory methods and methods
24  }
```

The Resulting Code

```
1   class EventController
2   {
3       public function listAction(Request $request)
4       {
5           $range = DateRange::fromStrings(
6               $request->query->get('start', '-60 minute'),
7               $request->query->get('end', 'now')
8           );
9
10          return [
11              'events' => $this->eventRepository->findBetween($range),
12          ];
13      }
14
15      public function listTodayAction()
16      {
17          return [
18              'events' => $this->eventRepository->findBetween(DateRange::today()),
19          ];
20      }
21  }
```

Introduce Value Objects

- Convert all hash maps (arrays) into objects
- Design interfaces from a usage perspective
- Add validation of “eternal truth”


Complex Code

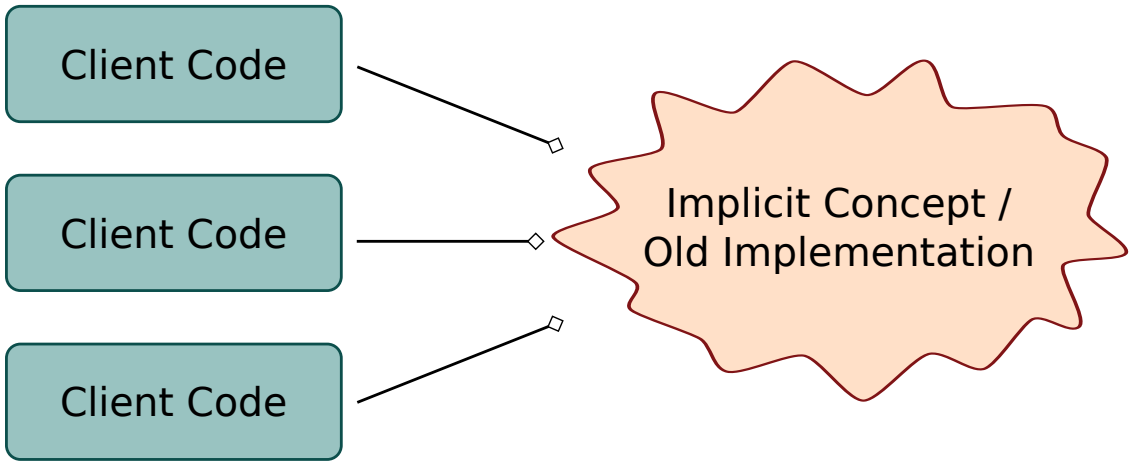
Complex Code

- Combination of logic with persistance, services etc.
- Hard to test or change

Branch By Abstraction

Technique to gradually replace larger sub systems in running software

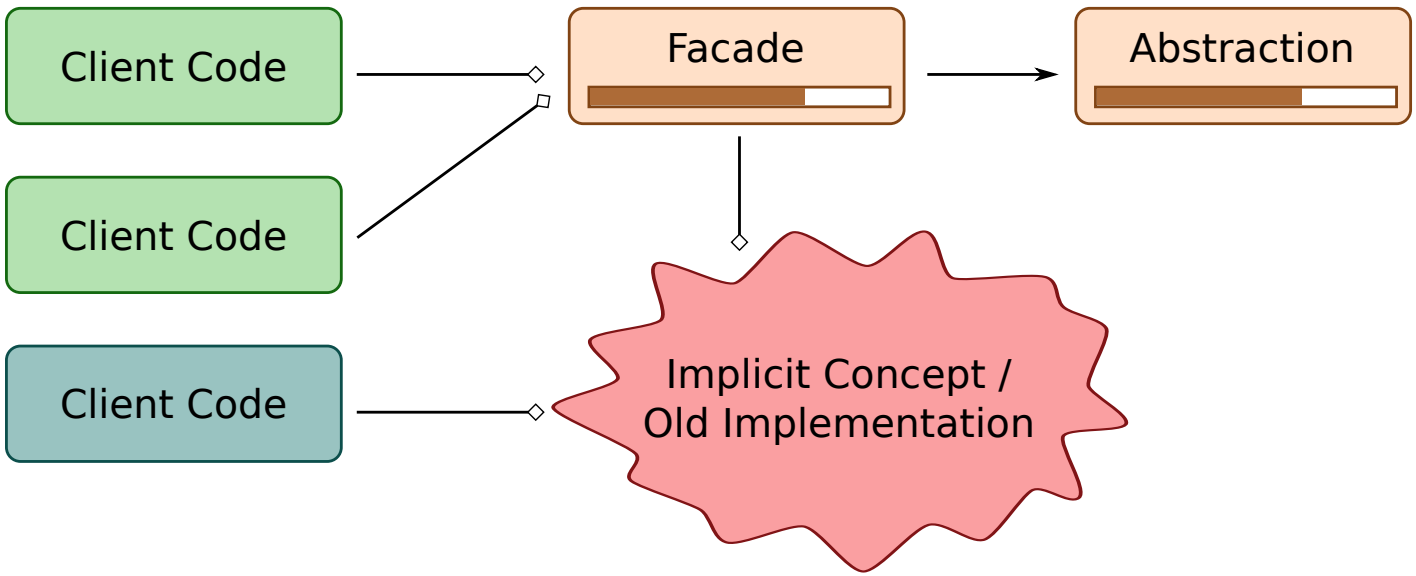
Initial State



Identify Usage in Clients

- Find out how concept is used
- Introduce new abstraction / facade step by step
- Adapt clients to use new Facade
  - Extract Method (Excellent tool support)
  - Extract Class (No tool support)

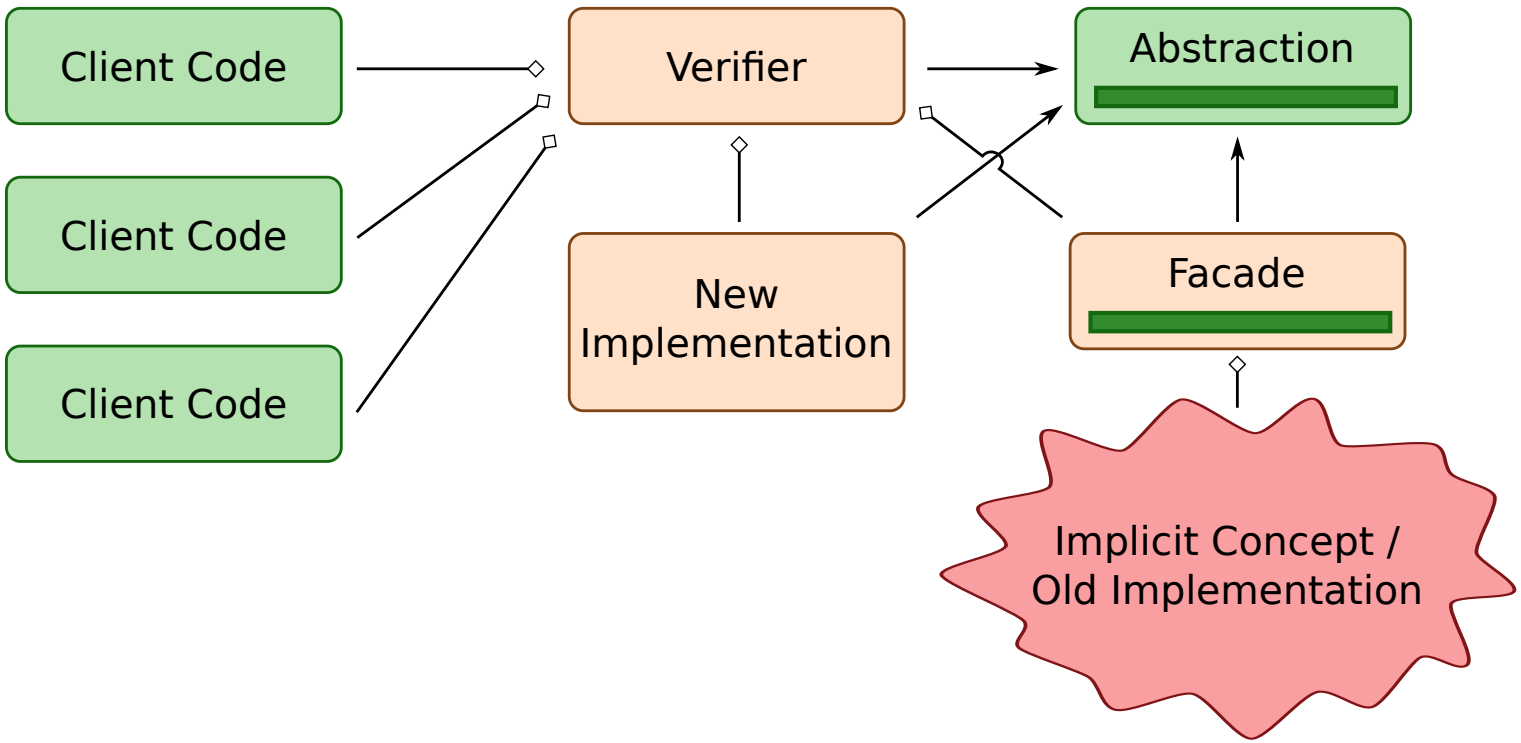
Adapt Clients



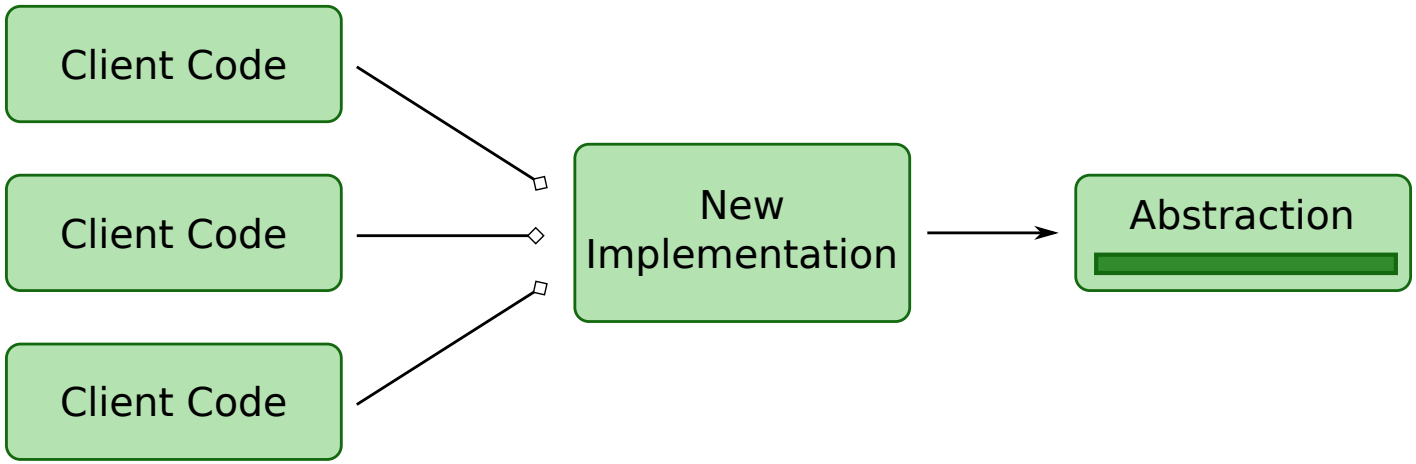
Start New Implementation

- Once facade is complete start new implementation (test driven)
- Verify new implementation with decorator

Introduce New Implementation



Delete Old Code



Branch By Abstraction

- Extract larger subsystems in running software
  - Persistence layer
  - Complex domain concepts
- Design interfaces from a usage perspective
- Validate refactoring in production

Global State

Global State

- Almost impossible to test
- Isolation of Bounded Contexts impossible (Microservices)

The Problem

```php
class SearchService
{
    public function searchAction($queryString, $type)
    {
        /** @var $solarium \Solarium_Client */
        $solarium = Solarium::getInstance();
        $select = $solarium->createSelect();

        // More and complex filtering logic to test

        $result = $solarium->query($select);

        return $result;
    }
}
```

Start By Extracting Construction

```
1   class SearchService
2   {
3       public function searchAction($queryString, $type)
4       {
5           /** @var $solarium \Solarium_Client */
6           $solarium = $this->getSolarium();
7           // ...
8       }
9
10      protected function getSolarium()
11      {
12          return Solarium::getInstance();
13      }
14  }
```

Use Optional Constructor Injection

```php
class SearchService
{
    private $solarium;

    public function __construct(\Solarium_Client $solarium = null)
    {
        $this->solarium = $solarium ?: Solarium::getInstance();
    }

    protected function getSolarium()
    {
        return $this->solarium;
    }
}
```

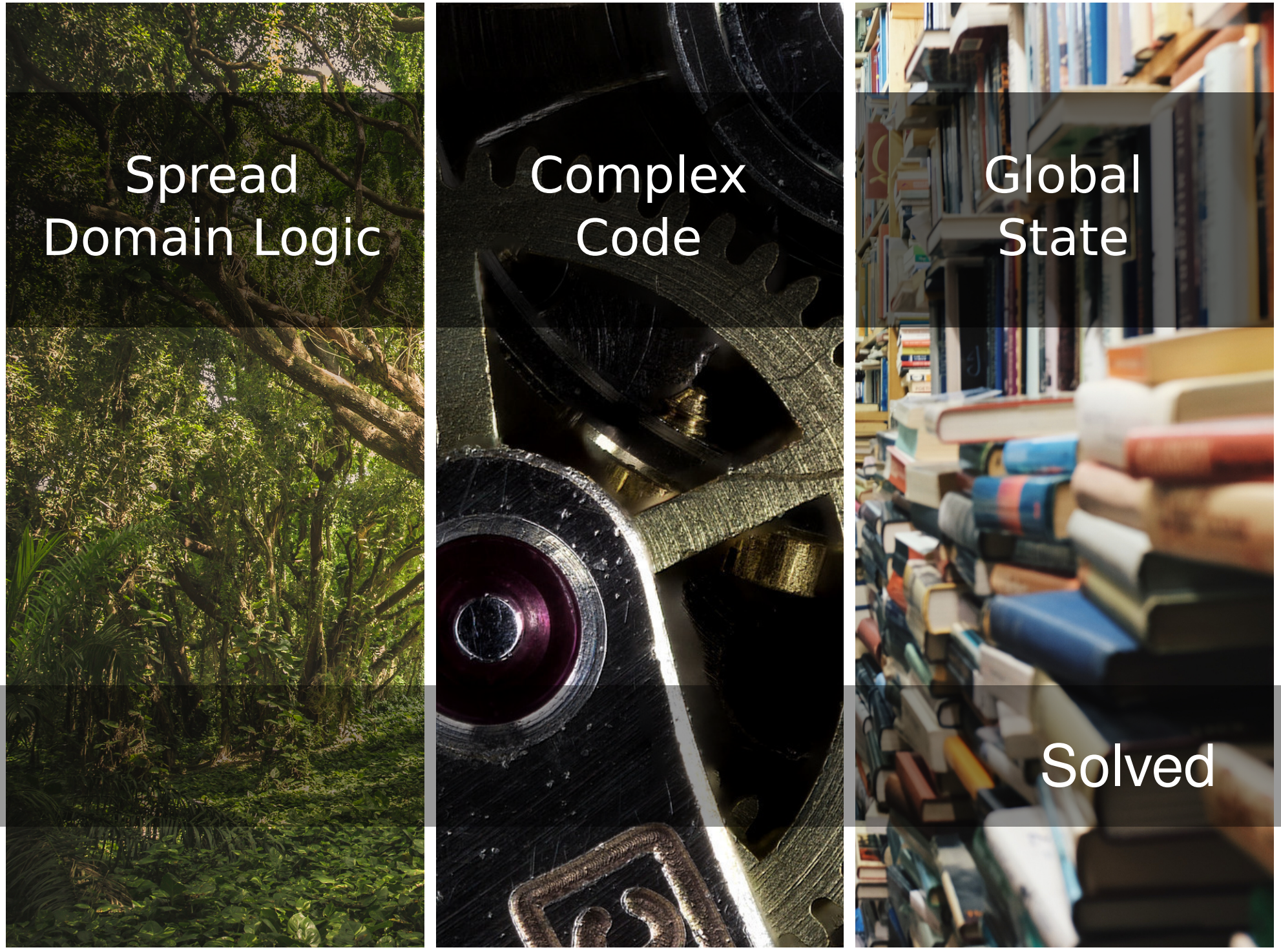
Oxid Entry Point (Controller) Example

```php
class acme_product_details extends oxProductDetails
{
    public function render()
    {
        parent::render();
        try {
            $dic = oxRegistry::get('acme_dic');
            $solarium = $dic->get('acme.bundle.solarium');
            $result = $solarium->searchProduct($_GET['product_id']);


            $this->addTplParam('product', $result->product);
            $this->addTplParam('related', $result->relatedProducts);

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

- Push **all** dependencies into services
- Use some dependency injection container to manage your dependency graph (application configuration)
- Use dependency injection containter (DIC) in legacy entry points
  - Might temporarily use `static` access
  - In the end it should only be used in the `index.php` dynamically


Spread
Domain Logic
Complex
Code
Global
State
Solved

Refactoring

Optimize for removal & replacement – **not** for re-use.

Refactoring

Execute in baby steps. Commit early and often.

https://qafoo.com/newsletter

Qafoo
passion for software quality

THANK YOU

Rent a quality expert
qafoo.com