Deploying Direct to Production:

A Scientist approach with Experiments

MessageBird

Who am I?

Mark Baker

Senior Software Engineer
MessageBird BV, Amsterdam

Coordinator and Developer of:

Open Source PHPOffice library
PHPExcel, PHPWord, PHPPowerPoint, PHPProject, PHPVisio
Minor contributor to PHP core (SPL Datastructures)
Other small open source libraries available on github

@Mark_Baker

https://github.com/MarkBaker

http://uk.linkedin.com/pub/mark-baker/b/572/171

http://markbakeruk.net

Why am I Here?

Why am I Here?

Deploying Direct to Production

Deploying Direct to Production

Deploying Direct to Production

Deploying Direct to Production

Deploying Direct to Production

- If Releasing is Risky, then you should fix your processes to reduce that risk.

  This isn't about Eliminating Errors

  It's about making the process resilient

Deploying Direct to Production – Ownership

- Developers are also Software Owners
  - Take responsibility for their code
  - Can deploy and rollback their own code
  - Work as part of a team
  - Peer Reviews

Deploying Direct to Production – Deployment

- Smaller and more Frequent changes are safer than large infrequent changes
- Good CI pipeline with an extensive unit test suite

Deploying Direct to Production – Deployment

- Fail Fast, Fix Fast
  - Verify the Changes immediately after Deployment
  - Ability to turn changes on and off (Feature Flags)

Deploying Direct to Production – Observability

- Observability-Driven Development
  - Code so that any Changes can be Verified
  - Good metrics to warn of problems

Deploying Direct to Production – Testing

Safe Testing in Production

A Scientist Approach

Scientist is an experimentation framework that will allow you to refactor and improve upon existing code in a live environment, without incurring risk or breakages.

A Scientist Approach

Scientist

Originally developed at Github

https://github.blog/2016-02-03-scientist/

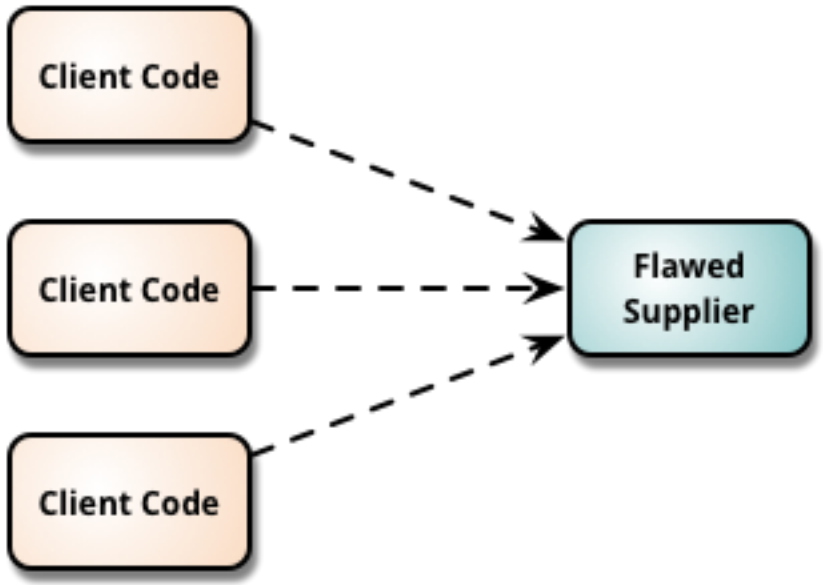
Branch by Abstraction

https://martinfowler.com/bliki/BranchByAbstraction.html

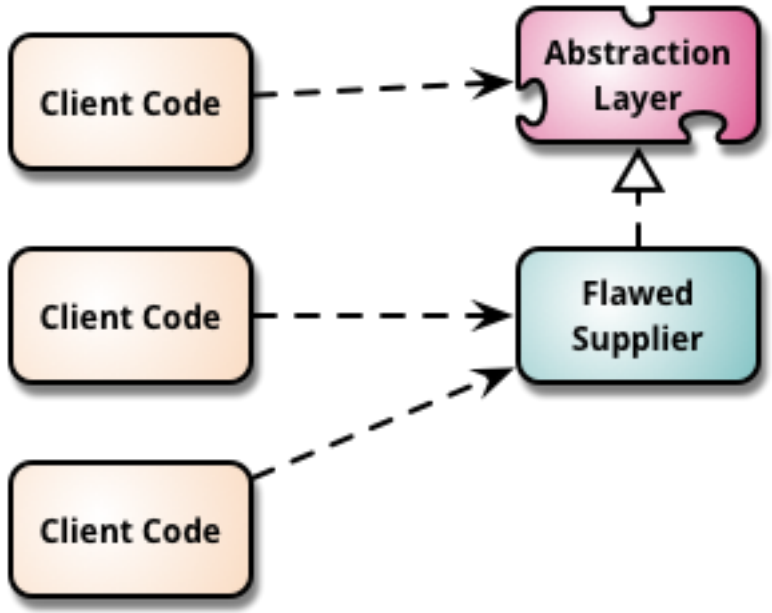
Branch by Abstraction

"Branch by Abstraction" is a technique for making a large-scale change to a software system in gradual way that allows you to release the system regularly while the change is still in-progress.

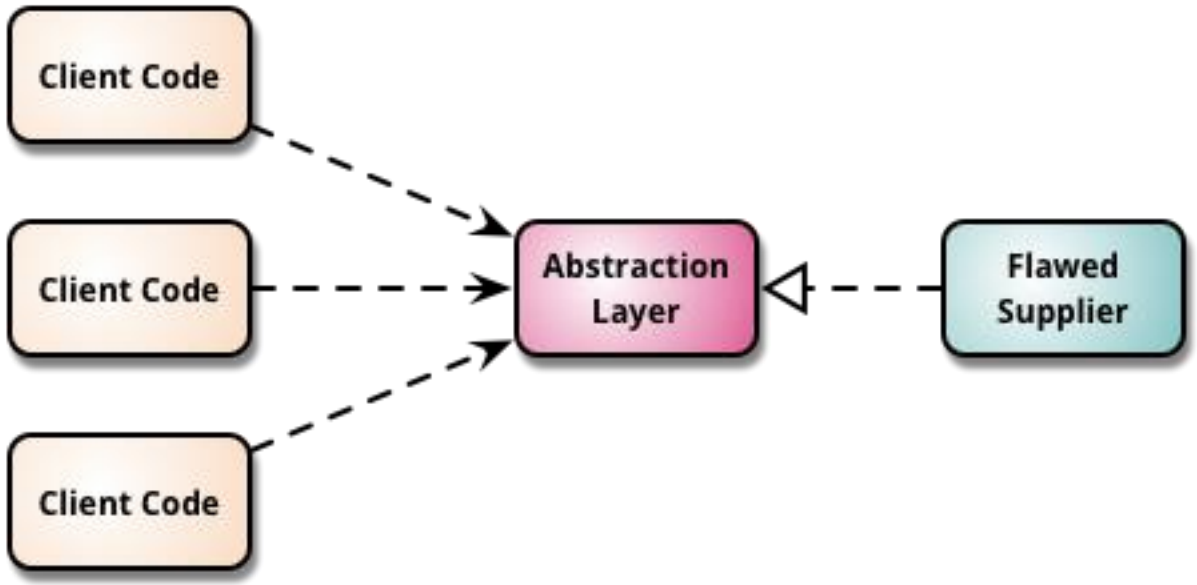
Branch by Abstraction



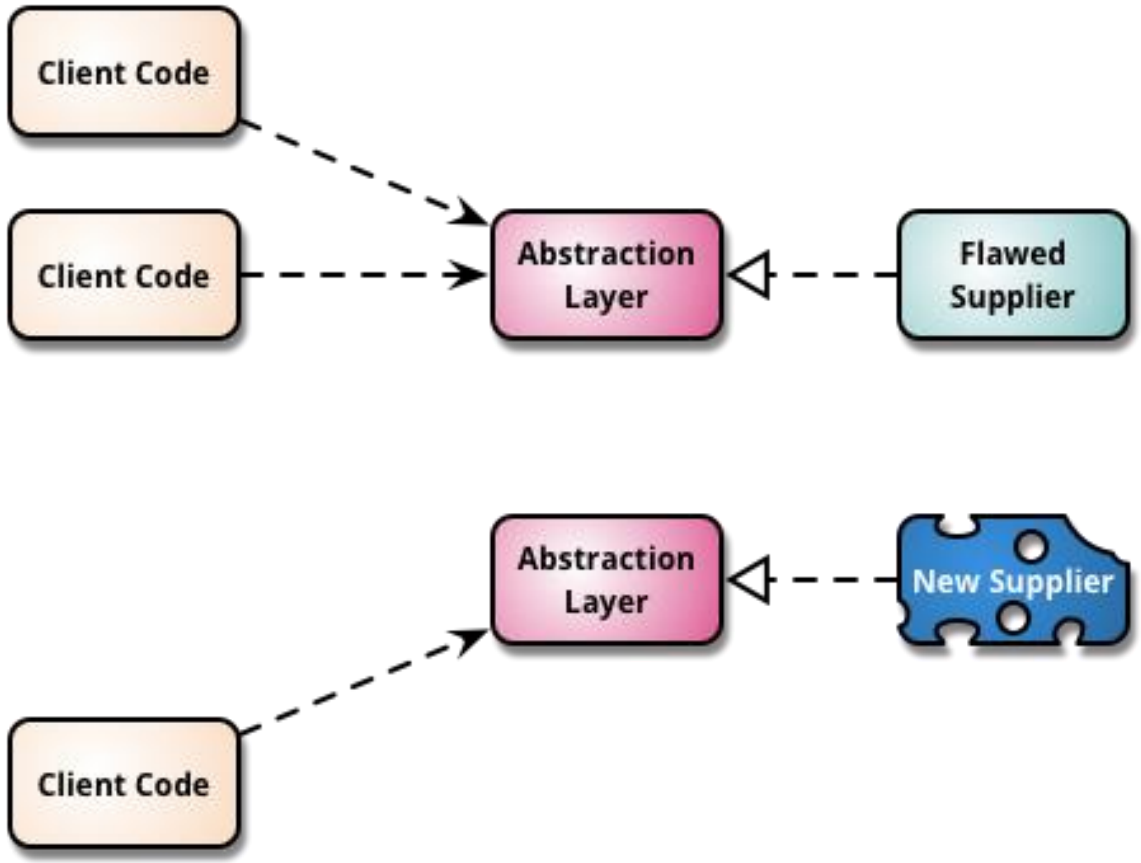
Branch by Abstraction



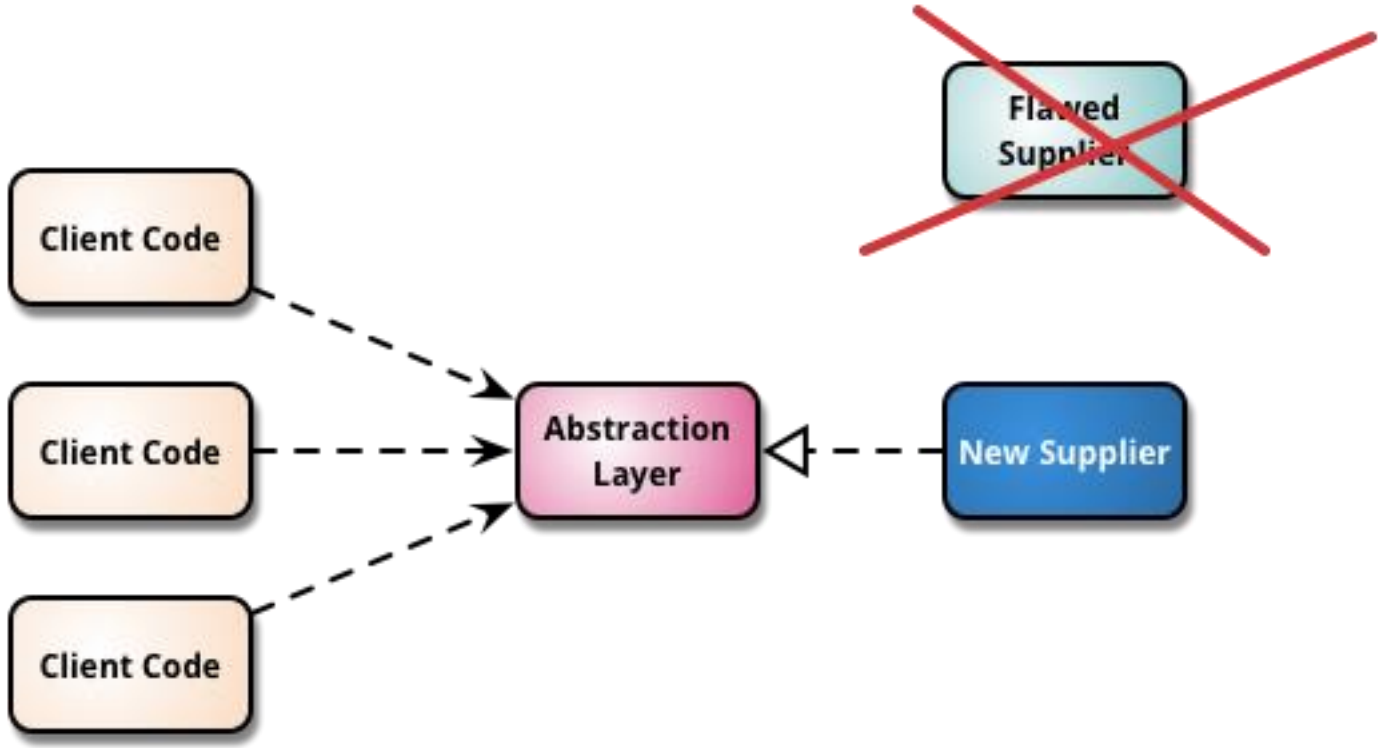
Branch by Abstraction



Branch by Abstraction



Branch by Abstraction



Scientist

Scientist

Scientist

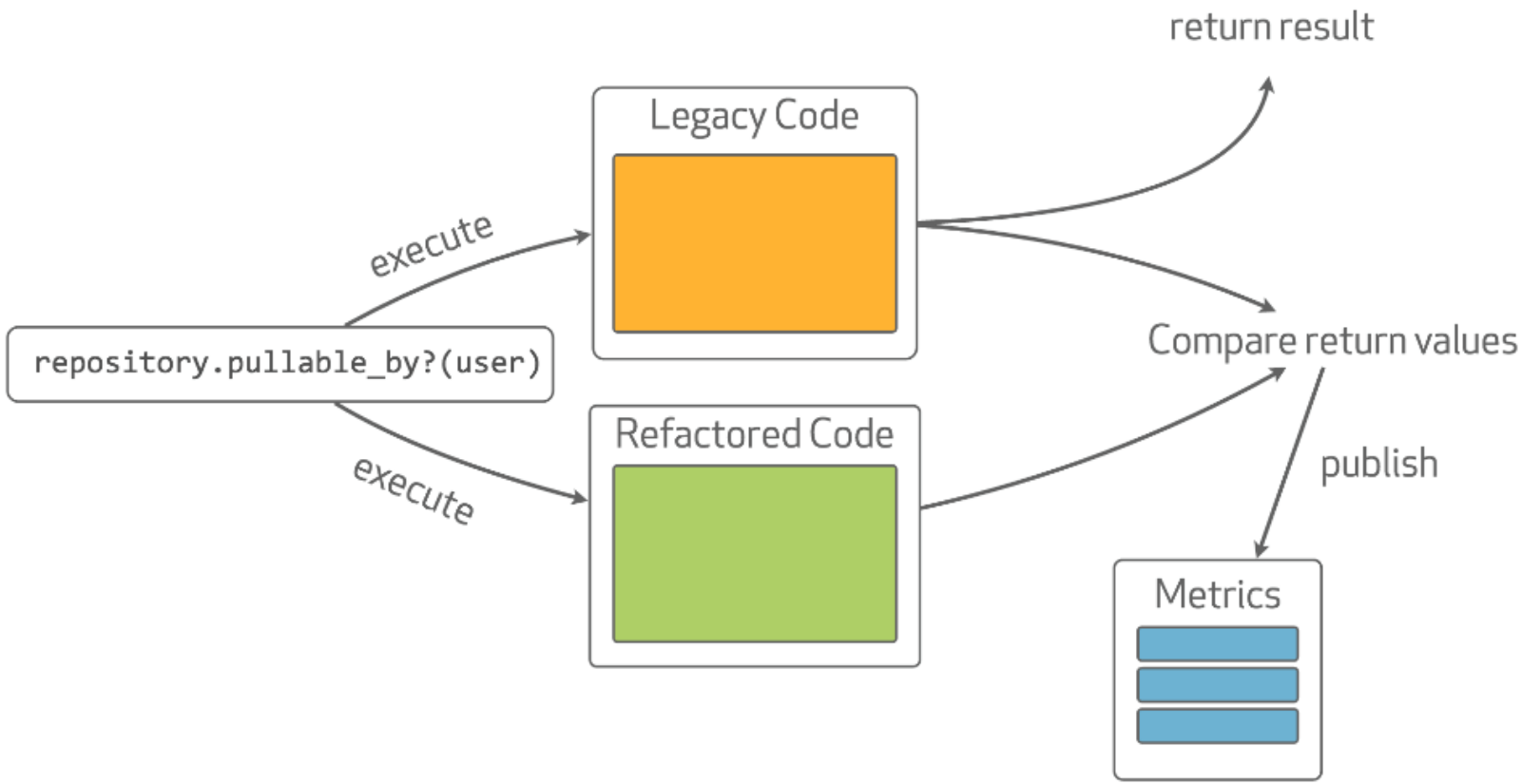
A Ruby library for carefully refactoring critical paths.

https://github.com/github/scientist

A PHP library inspired by Scientist.

https://github.com/daylerees/scientist
https://scientist.readme.io/

Scientist


return result
Legacy Code
execute
repository.pullable_by?(user)
Compare return values
Refactored Code
execute
publish
Metrics

Experiments

Experiments

```
$experiment = (new Scientist\Laboratory)
    ->experiment('To take life on earth to the second birth')
    ->control($controlCallback)
    ->trial('A theme she had', $trialCallback1)
    ->trial('On a scheme he had', $trialCallback2);

$value = $experiment->run();
```

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        $valid = true;

        foreach($products as $product) {
            $fields = array_keys($product);
            foreach ($this->requiredFields as $requiredField) {
                if (!in_array($requiredField, $fields)) {
                    $valid = false;
                }
            }
        }

        return $valid;
    }
}
```

Experiments

```
$productList = [
    [
        'name' => 'Build APIs You Won\'t Hate',
        'description' => 'Available',
        'type' => 'e-Book',
        'price' => 25.99
    ],
    [
        'name' => 'Surviving Other People\'s APIs',
        'description' => 'Pre-order',
        'type' => 'e-Book',
        'price' => 21.99
    ],
];

$productValidator = new ProductValidator();
$productListIsValid = $productValidator->validateProductList($productList);
```

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        $invalidProducts = array_filter($products, [$this, 'isInvalidProduct']);

        return count($invalidProducts) === 0;
    }

    private function isInvalidProduct(array $product): bool {
        $fields = array_keys($product);
        $missingFields = array_diff($this->requiredFields, $fields);

        return count($missingFields) > 0;
    }
}
```

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        $valid = true;

        foreach($products as $product) {
            $fields = array_keys($product);
            foreach ($this->requiredFields as $requiredField) {
                if (!in_array($requiredField, $fields)) {
                    $valid = false;
                }
            }
        }

        return $valid;
    }
}
```

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    private function oldValidateProductList(array $products): bool {
        $valid = true;

        foreach($products as $product) {
            $fields = array_keys($product);
            foreach ($this->requiredFields as $requiredField) {
                if (!in_array($requiredField, $fields)) {
                    $valid = false;
                }
            }
        }

        return $valid;
    }
}
```

Experiments

```
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        return $this->oldvalidateProductList($products);
    }

    private function oldValidateProductList(array $products): bool {
        ...
    }
}
```

Experiments

- The `validateProductList()` method is now our Abstraction Layer

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        return $this->oldvalidateProductList($products);
    }

    private function oldValidateProductList(array $products): bool {
        ...
    }

    private function newValidateProductList(array $products): bool {
        ...
    }
}
```

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        $experiment = (new Scientist\Laboratory)
            ->experiment('Verify Validation of Product List')
            ->control([$this, 'oldValidateProductList'])
            ->trial('Experimental New Version', [$this, 'newValidateProductList']);

        return $experiment->run($products);
    }

    private function oldValidateProductList(array $products): bool {
        ...
    }

    private function newValidateProductList(array $products): bool {
        ...
    }
}
```

Experiments

- Publish the Results, Analyse the data comparisons, and refine `newValidateProductList()` code if necessary

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        return $this->newValidateProductList($products);
    }

    private function oldValidateProductList(array $products): bool {
        ...
    }

    private function newValidateProductList(array $products): bool {
        ...
    }
}
```

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        return $this->newValidateProductList($products);
    }

    private function newValidateProductList(array $products): bool {
        ...
    }
}
```

Experiments

```php
class ProductValidator {
    private $requiredFields = ['name', 'description', 'type', 'price'];

    public function validateProductList(array $products): bool {
        $invalidProducts = array_filter($products, [$this, 'isInvalidProduct']);

        return count($invalidProducts) === 0;
    }

    private function isInvalidProduct(array $product): bool {
        $fields = array_keys($product);
        $missingFields = array_diff($this->requiredFields, $fields);

        return count($missingFields) > 0;
    }
}
```

Experiments

chance()

Allows us to run the experiment only on a random percentage of experiments

matcher()

Enables us to define custom rules for comparing the control result with the trial result

Experiments

Journals

Journals allow experiment data to be sent to data stores for later inspection. (PSR-3)

Reports

For viewing the results of Experiments

Experiments

Results

- Time
- Memory
- Exceptions
- Data Value Match/Non-match

Other Processes

Feature Flags

- Easily Enable/Disable Changes in Code

Canary Deployments

- Controlled Requests to Changed Code
- Promote Releases Gracefully

A Scientist Approach

Dank u wel!

A Scientist Approach

Merci beaucoup!