Writing Better Tests

Anna Filina
TESTING AND LEGACY EXPERT
@afilina afilina.com

Summary

No redundancies

Isolated

Simple

No Redundancies

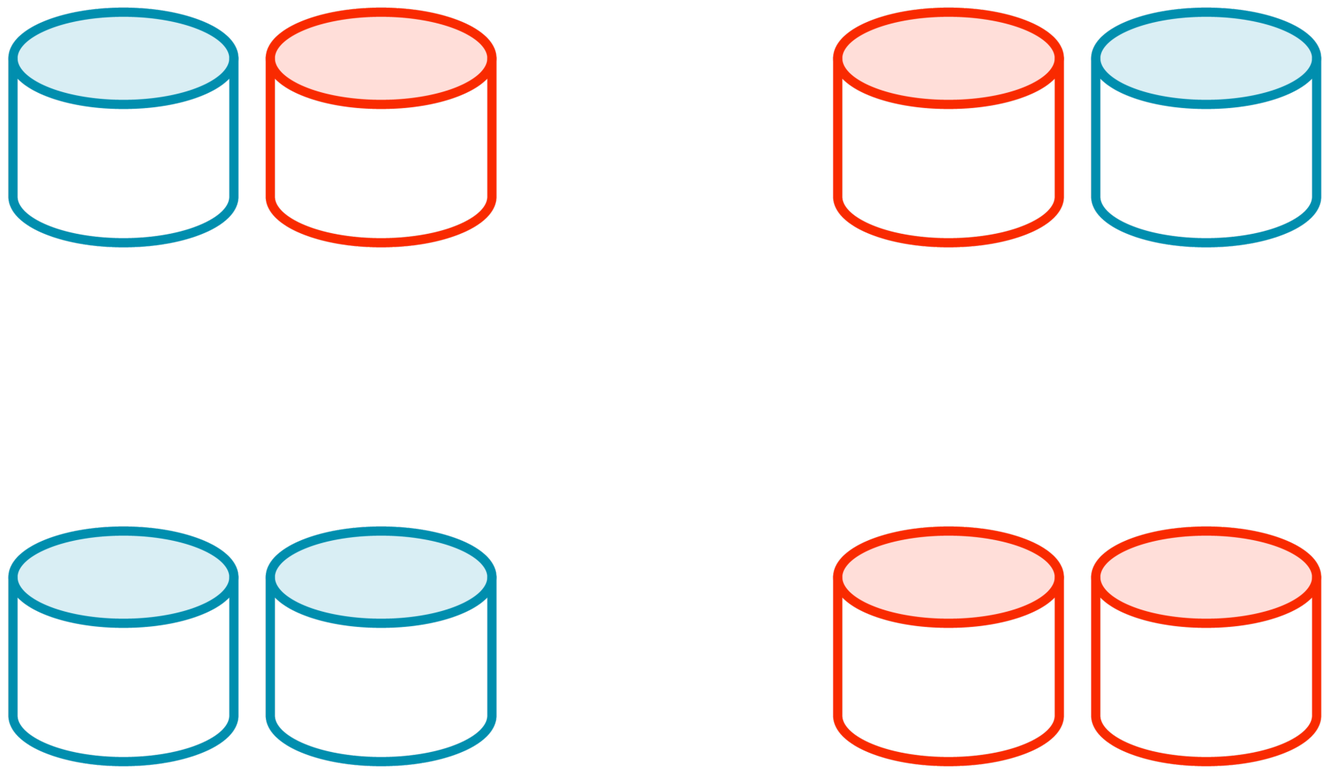
```php
final class AmountCalculator
{
    /**
     * @param Operation[] $operations
     */
    public function getResult(Amount $amount, array $operations): Amount
    {
        //...
    }
}
```

```
final class MarkupOperation implements Operation

final class DiscountOperation implements Operation
```

```php
$discountOperation = new DiscountOperation([
    //...
]);
$discountOperation->applyTo(new Amount(500));
```

```php
$discountOperation = new DiscountOperation([
    //...
]);
$discountOperation->applyTo(new Amount(500));
```

```php
$discountOperation = new DiscountOperation([
    //...
]);

$this->amountCalculator->getResult(
    new Amount(500),
    [$discountOperation]
);
```

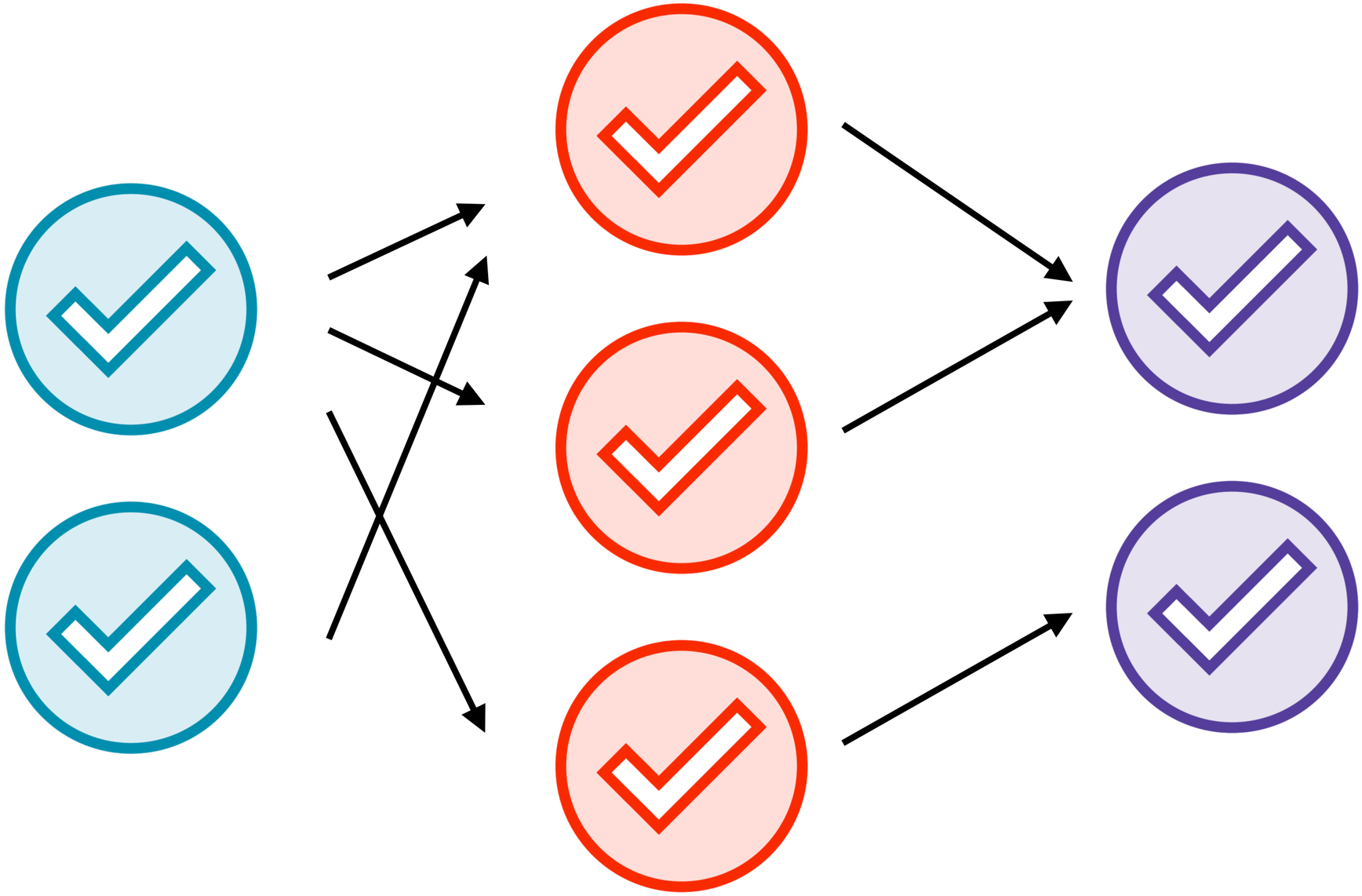


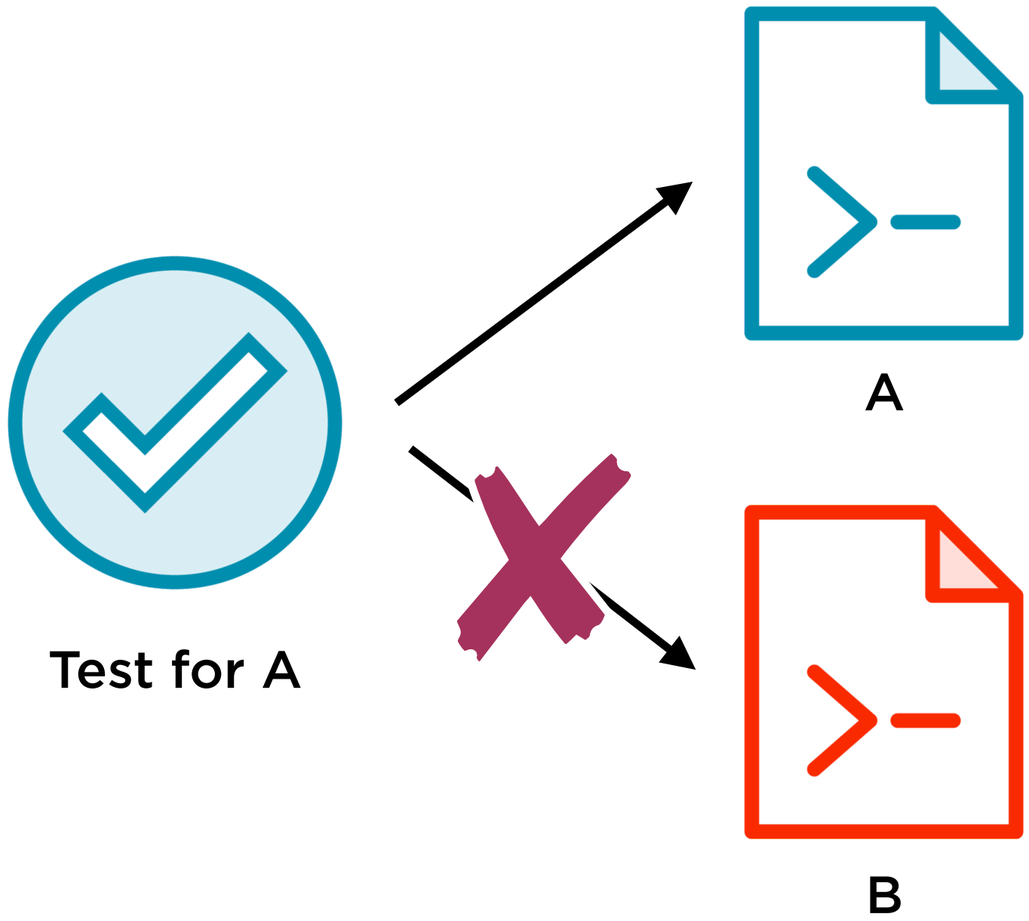
```php
$operation1 = new DiscountOperation([
    Discount::fromAmount(200),
    Discount::fromPercent(100),
]);

$operation2 = new DiscountOperation([
    Discount::fromAmount(200),
    Discount::fromAmount(200),
]);

$operation3 = new DiscountOperation([
    Discount::fromPercent(200),
    Discount::fromPercent(200),
]);

$operation4 = new DiscountOperation([
    Discount::fromPercent(100),
    Discount::fromAmount(200),
]);
```

```php
$operation1 = new DiscountOperation([
    Discount::fromAmount(200),
    Discount::fromPercent(100),
]);

$operation2 = new DiscountOperation([
    Discount::fromAmount(200),
    Discount::fromAmount(200),
]);

$operation3 = new DiscountOperation([
    Discount::fromPercent(200),
    Discount::fromPercent(200),
]);

$operation4 = new DiscountOperation([
    Discount::fromPercent(100),
    Discount::fromAmount(200),
]);
```

Tests Needed < Combinations

Isolated




A
Test for A
B

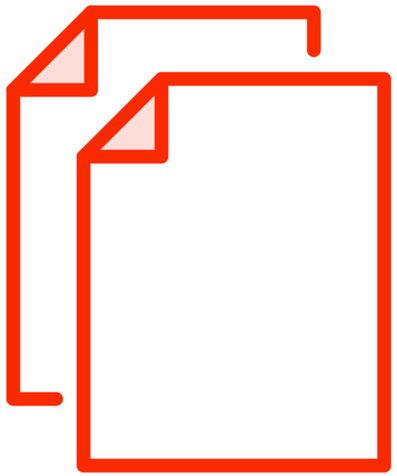
```php
public function __construct(ProductRepository $repo)
{
    $this->repo = $repo;
}
```

```php
//..
$entityManager = EntityManager::create($connection, $config);
$repo = new DoctrineProductRepository($entityManager);

$testSubject = new ProductListHandler($repo);
```

```php
//..
$entityManager = EntityManager::create($connection, $config);
$repo = new DoctrineProductRepository($entityManager);

$testSubject = new ProductListHandler($repo);
```

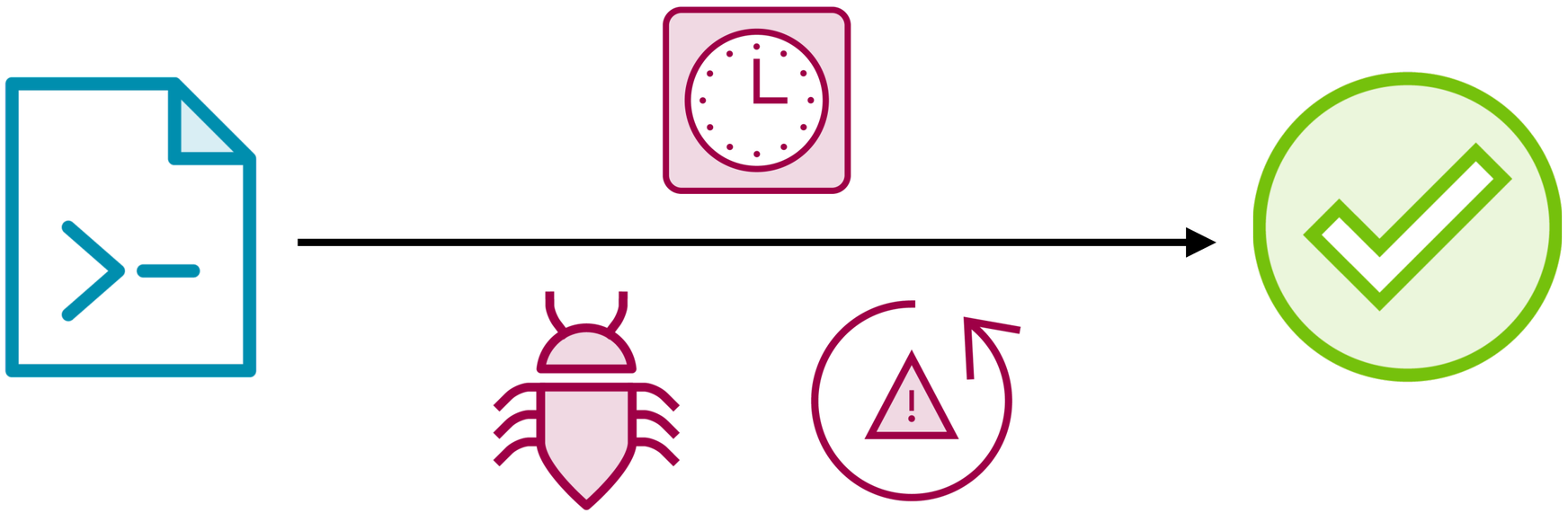
```php
$repo = $this->createStub(ProductRepository::class);

$testSubject = new ProductListHandler($repo);
```

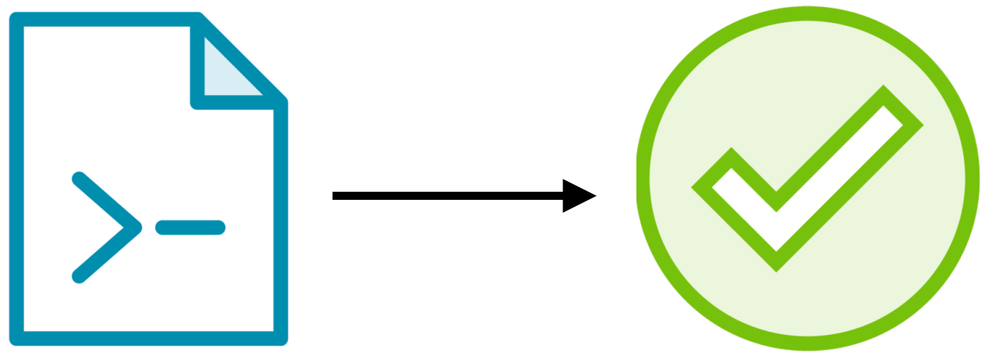
Database

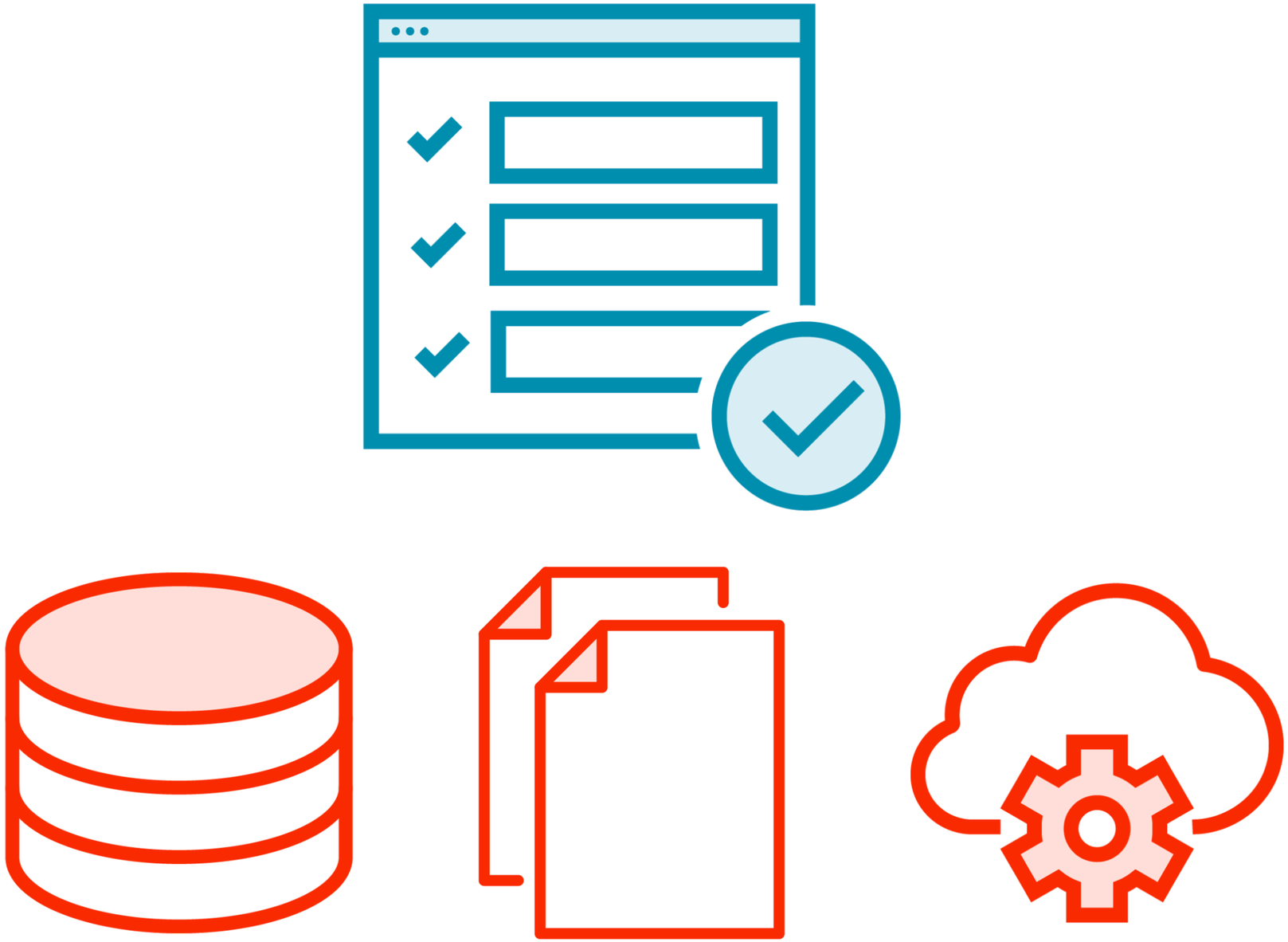


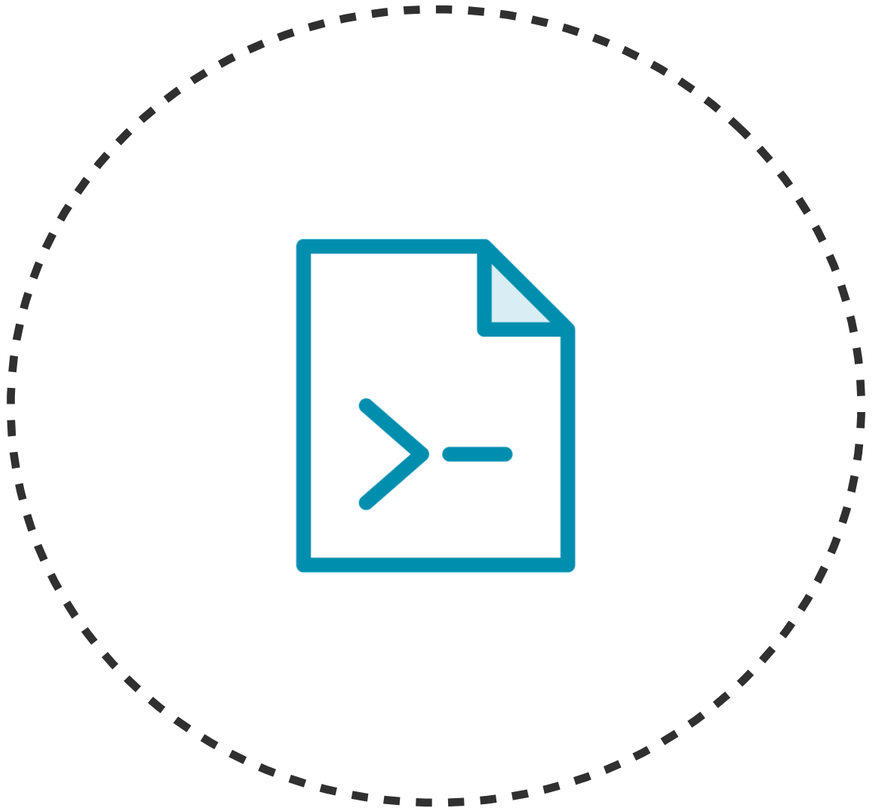
Filesystem

Web Service

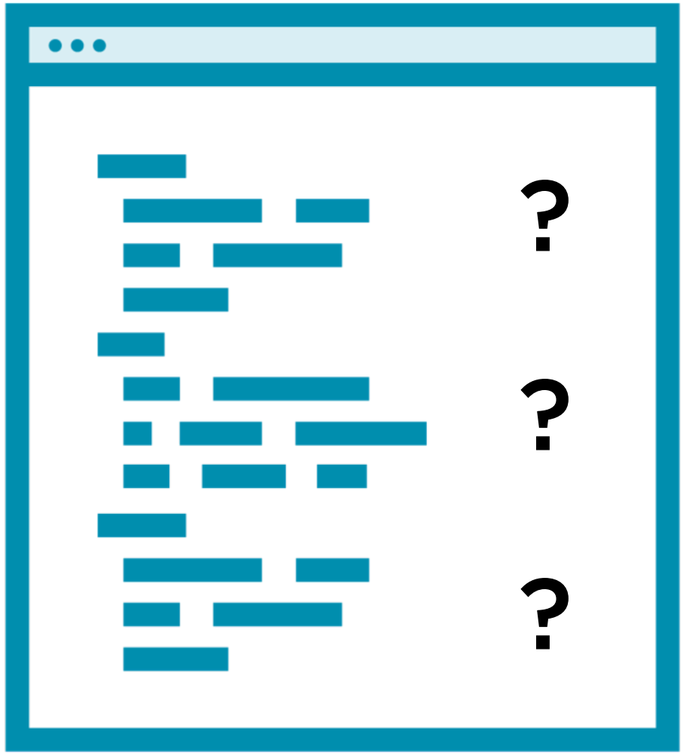








Simple



```php
$transaction = new Transaction(
    new Amount(200),
    Transaction::TYPE_REFUND,
    new AccountNumber('1020AB')
);
```

```php
$transaction = new Transaction(
    new Amount(200),
    Transaction::TYPE_REFUND,
    new AccountNumber('1020AB')
);
```

```php
$transaction = $this->createRefund();
```

```php
$this->product = new Product(
    'To the Moon and Back',
    new Amount(200)
);

self::assertEquals(
    'To the Moon and Back',
    $this->product->getName()
);
```

```php
$this->product = new Product(
    'To the Moon and Back',
    new Amount(200)
);

self::assertEquals(
    'To the Moon and Back',
    $this->product->getName()
);
```

```php
$product = new Product(
    self::NAME,
    new Amount(self::PRICE)
);

self::assertEquals(
    self::NAME,
    $product->getName()
);
```

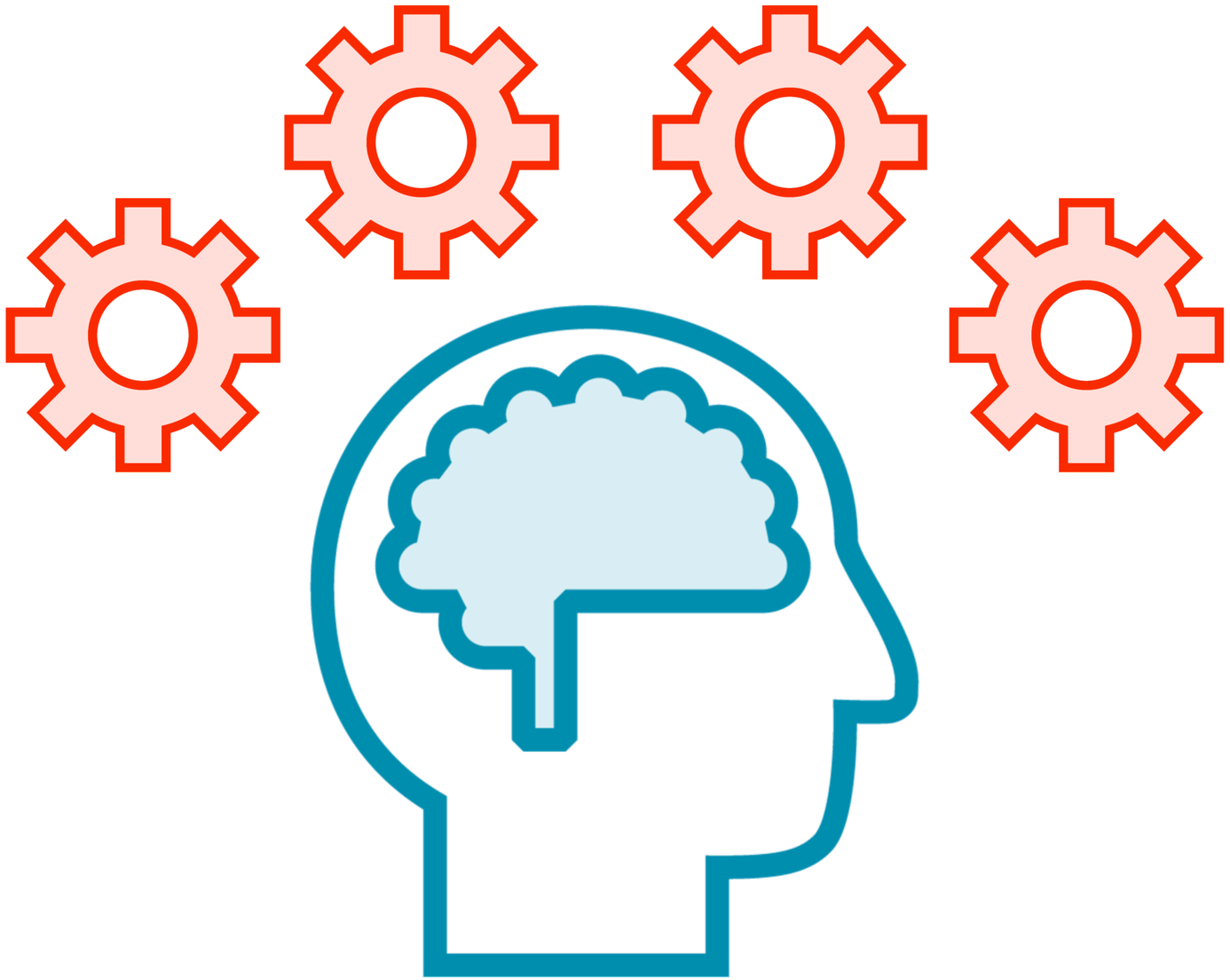
```php
$data = simplexml_load_file('tests-bad/data/discounts.xml');
$discounts = [];

foreach ($data->discount as $item) {
    if ((string)$item['type'] === 'percent') {
        $discounts[] = Discount::fromPercent((float)$item['value']);
    }
    if ((string)$item['type'] === 'amount') {
        $discounts[] = Discount::fromAmount((int)$item['value']);
    }
}

self::assertEquals(
    new Amount(850),
    $this->calculator->getResult(
        new Amount(1000),
        [
            new DiscountOperation($discounts)
        ])
);
```

```
<root>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="percent" value="0.01"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
    <discount type="amount" value="5"/>
</root>
```

```php
$operation = new DiscountOperation([
    Discount::fromAmount(100)
]);

$modifiedAmount = $this->amountCalculator->getResult(
    new Amount(1000),
    [$operation]
);

self::assertEquals(
    new Amount(900),
    $modifiedAmount
);
```

```php
$operation = new DiscountOperation([
    Discount::fromAmount(100)
]);

$modifiedAmount = $this->amountCalculator->getResult(
    new Amount(1000),
    [$operation]
);

self::assertEquals(
    new Amount(900),
    $modifiedAmount
);
```

```php
self::assertEquals(
    new Amount(900),
    $modifiedAmount
);
```



Up Next:
Using Mocks