Testing with(out) dependencies

Sebastian Bergmann

I can help you

- adopt PHPUnit,
- optimize your tests,
- refine your development processes, and
- write more testable code

thePHP.cc

„An object-oriented system is a web of collaborating objects. A system is built by creating objects and plugging them together so that they can send messages to one another. The behavior of the system is an emergent property of the composition of the objects - the choice of objects and how they are connected.“

Steve Freeman and Nat Pryce

„The big idea is "messaging" [...] The key in making great and growable systems is much more to design how its modules communicate rather than what their internal properties and behaviors should be.“

Alan Kay

Services

- Encapsulate computational processes
- Should implement an interface

Business Objects

- (Immutable) Value Objects
- Entities

Services

- Encapsulate computational processes
- Should implement an interface

Newable Objects

- (Immutable) Value Objects
- Entities

„When we are writing a test in which we cannot (or choose not to) use a real [collaborating object], we can replace it with a Test Double.“

Gerard Meszaros

You would not risk the real Harrison Ford getting injured, right?

You would not test domain logic while talking to the database, right?

thePHP.cc

- Dummy Object
- Test Stub
- Test Spy
- Mock Object
- Fake Object

- Dummy Object
- Test Stub
- Test Spy
- Mock Object
- Fake Object

Test Stub

- Looks like a real collaborating object
- Can be configured to return a value or throw an exception
- Allows us to test an object that interacts with the doubled object

Mock Object

- Looks like a real collaborating object
- Can be configured to return a value or throw an exception
- Can be configured to expect messages (method calls)
- Allows us to test the communication between objects

```php
public function testEmitsGoodSoldEvent(): void
{
    $sourcer = new class implements MarketSourcer
    {
        public function source(): Market
        {
            return Market::from(5, 3, 3, 3, 3, 3);
        }
    };

    // ...

    $processor = new SellGoodCommandProcessor($emitter, $sourcer);

    $processor->process(new SellGoodCommand(Good::Bread, 1));

    // ...
}
```

```php
public function testWritesSubscribedEvents(): void
{
    // ...

    $writer = new class implements EventWriter {
        private array $events;

        public function write(Event $event): void {
            $this->events[] = $event;
        }

        public function events(): array {
            return $this->events;
        }
    };

    $subscriber = new WritingEventSubscriber($writer);
    $subscriber->notify($event);

    $this->assertSame([$event], $writer->events());
}
```

Test Spy (handwritten anonymous class)

PHPUnit can create test stubs and mock objects dynamically, so you do not need to implement them manually.

Using an API, you can specify how a test stub should behave as well as the expected communication between the object you test and a mock object.

I will show the most common use cases while ignoring edge cases.

```php
final class SellGoodCommandProcessorTest extends TestCase
{
    public function testEmitsGoodSoldEvent(): void
    {
        $sourcer = $this->createStub(MarketSourcer::class);

        $sourcer
            ->method('source')
            ->willReturn(Market::from(5, 3, 3, 3, 3, 3));

        // ...

        $processor = new SellGoodCommandProcessor($emitter, $sourcer);

        $processor->process(new SellGoodCommand(Good::Bread, 1));
    }
}
```

Test Stub (dynamically created using API)

```php
final class WritingEventSubscriberTest extends TestCase
{
    public function testWritesSubscribedEvents(): void
    {
        // ...

        $writer = $this->createMock(EventWriter::class);

        $writer
            ->expects($this->once())
            ->method('write')
            ->with($event);

        $subscriber = new WritingEventSubscriber($writer);

        $subscriber->notify($event);
    }
}
```

Mock Object (dynamically created using API)

```php
<?php declare(strict_types=1);
namespace PHPUnit\Framework;

abstract class TestCase extends Assert implements /* ... */
{
    // ...

    /**
     * @template RealInstanceType of object
     * @param class-string<RealInstanceType> $type
     * @return Stub&RealInstanceType
     */
    final protected static function createStub(string $type): Stub
    {
        // ...
```

`createStub($type)` returns an object of type **`$type&Stub`**

```php
$collaborator = $this->createStub(Collaborator::class);

$collaborator
    ->method('doSomething')
    ->willReturn('value');
```

Every call to **`doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createStub(Collaborator::class);

$collaborator
    ->method('doSomething')
    ->willReturn('value', 'another-value');
```

- The first call to **`doSomething()`** will return **`'value'`**
- The second call to **`doSomething()`** will return **`'another-value'`**
- Subsequent calls to **`doSomething()`** will trigger an error!

```php
$collaborator = $this->createStub(Collaborator::class);

$collaborator
    ->method('doSomething')
    ->willThrowException(new RuntimeException);
```

Every call to **`doSomething()`** will throw a **`RuntimeException`**

```
❯ ./phpunit --group test-doubles/test-stub --testdox --no-progress
PHPUnit 12.5.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.5.0
Configuration: /Users/sb/Work/OpenSource/phpunit/phpunit.xml

Time: 00:00.014, Memory: 12.00 MB

createStub()
 ✔ Creates test stub for interface
 ✔ Creates test stub for extendable class
 ✔ Creates test stub for extendable readonly class
 ✔ Cannot create test stub for final class
 ✔ Cannot create test stub for enumeration
 ✔ Cannot create test stub for unknown type
 ✔ Return value generation is enabled by default
 ✔ Return value generation can be disabled with attribute

createStubForIntersectionOfInterfaces()
 ✔ Creates test stub for intersection of interfaces
 ✔ Cannot create test stub for intersection of interfaces when less than two interfaces are specified
 ✔ Cannot create test stub for intersection of unknown interfaces
 ✔ Cannot create test stub for intersection of interfaces that declare the same method
 ✔ Return value generation is enabled by default
 ✔ Return value generation can be disabled with attribute

createConfiguredStub()
 ✔ Creates test stub for interface or extendable class with return value configuration for multiple methods
```

```
❯ ./phpunit --group test-doubles/test-stub --testdox --no-progress
PHPUnit 12.5.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.5.0
Configuration: /Users/sb/Work/OpenSource/phpunit/phpunit.xml

Time: 00:00.014, Memory: 12.00 MB

Test Stub
 ✔ Method returns null when return value is nullable and no return value is configured
 ✔ Method returns generated value when return value generation is enabled and no return value is configured
 ✔ __toString() method returns empty string when return value generation is disabled and no return value is
   configured
 ✔ Method does not return value when return value generation is disabled and no return value is configured
 ✔ Method returns configured value when return value is configured
 ✔ Configured return value must be compatible with return type declaration
 ✔ Objects passed as argument are not cloned by default
 ✔ Cloning of objects passed as argument can be enabled
 ✔ Method can be configured to return one of its arguments
 ✔ Method can be configured to return self reference
 ✔ Method can be configured to return reference
 ✔ Method can be configured to return values based on argument mapping
 ✔ Method with default parameter values can be configured to return values based on argument mapping
 ✔ Method with default parameter values can be configured to return values based on argument mapping that
   omits default values
 ✔ Method can be configured to return values using callback
 ✔ Method can be configured to return different values on consecutive calls
 ✔ Method can be configured to return different values and throw exceptions on consecutive calls
 ✔ Method can be configured to throw an exception
 ✔ Method with never return type declaration throws exception
 ✔ Original __clone() method is not called by default when test double object is cloned
 ✔ Original __clone() method can optionally be called when test double object is cloned
```

```
❯ ./phpunit --group test-doubles/test-stub --testdox --no-progress
PHPUnit 12.5.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.5.0
Configuration: /Users/sb/Work/OpenSource/phpunit/phpunit.xml

Time: 00:00.014, Memory: 12.00 MB

Return Value Generator (PHPUnit\Framework\MockObject\ReturnValueGenerator)
 ✔ Generates null for missing return type declaration
 ✔ Generates null for null
 ✔ Generates null for mixed
 ✔ Generates null for void
 ✔ Generates true for true
 ✔ Generates false for bool
 ✔ Generates false for false
 ✔ Generates 0.0 for float
 ✔ Generates 0 for int
 ✔ Generates empty string for string
 ✔ Generates empty array for array
 ✔ Generates stdClass object for object
 ✔ Generates callable for callable
 ✔ Generates callable for Closure
 ✔ Generates Generator for Generator
 ✔ Generates Generator for Traversable
 ✔ Generates Generator for iterable
```

```
❯ ./phpunit --group test-doubles/test-stub --testdox --no-progress
PHPUnit 12.5.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.5.0
Configuration: /Users/sb/Work/OpenSource/phpunit/phpunit.xml

Time: 00:00.014, Memory: 12.00 MB

Return Value Generator (PHPUnit\Framework\MockObject\ReturnValueGenerator)
 ✔ Generates test stub for class or interface name
 ✔ Generates test stub for intersection of interfaces
 ✔ Generates new instance of test stub for static
 ✔ Generates new instance of test stub for static when used recursively
 ✔ Generates null for null|true|float|int|string|array|object
 ✔ Generates null for null|false|float|int|string|array|object
 ✔ Generates null for null|bool|float|int|string|array|object
 ✔ Generates true for true|float|int|string|array|object
 ✔ Generates false for false|float|int|string|array|object
 ✔ Generates false for bool|float|int|string|array|object
 ✔ Generates 0 for int|string|array|object
 ✔ Generates empty for string|array|object
 ✔ Generates array for array|object
 ✔ Generates stdClass object for union that contains object and unknown type
 ✔ Generates test stub for first intersection of interfaces found in union of intersections
 ✔ Generates test stub for unknown type
```

```php
<?php declare(strict_types=1);
namespace PHPUnit\Framework;

abstract class TestCase extends Assert implements /* ... */
{
    // ...

    /**
     * @template RealInstanceType of object
     * @param class-string<RealInstanceType> $type
     * @return MockObject&RealInstanceType
     */
    final protected function createMock(string $type): MockObject
    {
        // ...
```

`createMock($type)` returns an object of type **`$type&MockObject`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->once())
    ->method('doSomething')
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must be called exactly once
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->exactly(1))
    ->method('doSomething')
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must be called exactly once
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->exactly(0))
    ->method('doSomething')
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must not be called
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->never())
    ->method('doSomething')
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must not be called
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->atLeastOnce())
    ->method('doSomething')
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must be called at least once
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->atLeast(1))
    ->method('doSomething')
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must be called at least once
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->atMost(1))
    ->method('doSomething')
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must be called at most once
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->once())
    ->method('doSomething')
    ->with('argument')
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must be called exactly once
- The first argument must be equal to **`'argument'`**
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->once())
    ->method('doSomething')
    ->with($this->equalTo('argument'))
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must be called exactly once
- The first argument must be equal to **`'argument'`**
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```php
$collaborator = $this->createMock(Collaborator::class);

$collaborator
    ->expects($this->once())
    ->method('doSomething')
    ->with($this->identicalTo('argument'))
    ->willReturn('value');
```

- **`$collaborator->doSomething()`** must be called exactly once
- The first argument must be identical to **`'argument'`**
- Every call to **`$collaborator->doSomething()`** will return **`'value'`**

```
❯ ./phpunit --group test-doubles/mock-object --testdox --no-progress
PHPUnit 12.5.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.5.0
Configuration: /Users/sb/Work/OpenSource/phpunit/phpunit.xml

Time: 00:00.014, Memory: 12.00 MB

createMock()
 ✔ Creates mock object for interface
 ✔ Creates mock object for extendable class
 ✔ Creates mock object for extendable readonly class
 ✔ Cannot create mock object for final class
 ✔ Cannot create mock object for enumeration
 ✔ Cannot create mock object for unknown type
 ✔ Return value generation is enabled by default
 ✔ Return value generation can be disabled with attribute

createMockForIntersectionOfInterfaces()
 ✔ Creates mock object for intersection of interfaces
 ✔ Cannot create mock object for intersection of interfaces when less than two interfaces are specified
 ✔ Cannot create mock object for intersection of unknown interfaces
 ✔ Cannot create mock object for intersection of interfaces that declare the same method
 ✔ Return value generation is enabled by default
 ✔ Return value generation can be disabled with attribute

createConfiguredMock()
 ✔ Creates mock object for interface or extendable class with return value configuration for multiple methods
```

```
❯ ./phpunit --group test-doubles/mock-object --testdox --no-progress
PHPUnit 12.5.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.5.0
Configuration: /Users/sb/Work/OpenSource/phpunit/phpunit.xml

Time: 00:00.014, Memory: 12.00 MB

Mock Object
 ✔ Method returns null when return value is nullable and no return value is configured
 ✔ Method returns generated value when return value generation is enabled and no return value is configured
 ✔ __toString() method returns empty string when return value generation is disabled and no return value is
   configured
 ✔ Method does not return value when return value generation is disabled and no return value is configured
 ✔ Method returns configured value when return value is configured
 ✔ Configured return value must be compatible with return type declaration
 ✔ Objects passed as argument are not cloned by default
 ✔ Cloning of objects passed as argument can be enabled
 ✔ Method can be configured to return one of its arguments
 ✔ Method can be configured to return self reference
 ✔ Method can be configured to return reference
 ✔ Method can be configured to return values based on argument mapping
 ✔ Method with default parameter values can be configured to return values based on argument mapping
 ✔ Method with default parameter values can be configured to return values based on argument mapping that
   omits default values
 ✔ Method can be configured to return values using callback
 ✔ Method can be configured to return different values on consecutive calls
 ✔ Method can be configured to return different values and throw exceptions on consecutive calls
 ✔ Method can be configured to throw an exception
 ✔ Method with never return type declaration throws exception
 ✔ Original __clone() method is not called by default when test double object is cloned
 ✔ Original __clone() method can optionally be called when test double object is cloned
```

```
❯ ./phpunit --group test-doubles/mock-object --testdox --no-progress
PHPUnit 12.5.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.5.0
Configuration: /Users/sb/Work/OpenSource/phpunit/phpunit.xml

Time: 00:00.014, Memory: 12.00 MB

Mock Object
 ✔ Expectation that method is never called succeeds when method is not called
 ✔ Expectation that method is never called fails when method is called
 ✔ Expectation that method is called zero or more times succeeds when method is not called
 ✔ Expectation that method is called zero or more times succeeds when method is called once
 ✔ Expectation that method is called once succeeds when method is called once
 ✔ Expectation that method is called once fails when method is never called
 ✔ Expectation that method is called once fails when method is called more than once
 ✔ Expectation that method is called at least once succeeds when method is called once
 ✔ Expectation that method is called at least once succeeds when method is called twice
 ✔ Expectation that method is called at least twice succeeds when method is called twice
 ✔ Expectation that method is called at least twice succeeds when method is called three times
 ✔ Expectation that method is called at least once fails when method is not called
 ✔ Expectation that method is called at least twice fails when method is called once
 ✔ Expectation that method is called twice succeeds when method is called twice
 ✔ Expectation that method is called twice fails when method is never called
 ✔ Expectation that method is called twice fails when method is called once
 ✔ Expectation that method is called twice fails when method is called three times
 ✔ Expectation that method is called at most once succeeds when method is never called
 ✔ Expectation that method is called at most once succeeds when method is called once
 ✔ Expectation that method is called at most once fails when method is called twice
 ✔ Expectation that method is called with any parameter succeeds when method is called with parameter
 ✔ Expectation that method is called with parameter succeeds when method is called with expected parameter
 ✔ Expectation that method is called with parameter fails when method is called but with unexpected parameter
```

```
❯ ./phpunit --group test-doubles/mock-object --testdox --no-progress
PHPUnit 12.5.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.5.0
Configuration: /Users/sb/Work/OpenSource/phpunit/phpunit.xml

Time: 00:00.014, Memory: 12.00 MB

Mock Object
 ✔ Method call can be expected contingent on whether another method was previously called
 ✔ Contingent expectations are not evaluated until their condition is met
 ✔ Contingent expectations are evaluated when their condition is met
 ✔ Expectation cannot be contingent on expectation that has not been configured
 ✔ Expectations cannot have duplicate ids
 ✔ Expectations can be configured on test stubs
 ✔ Will return callback with variadic variables
 ✔ Expectations are cloned when test double is cloned
```

Services

- Implement an interface that can be doubled
- We use a real implementation of the interface when we want to test this implementation
- We use a test stub of the interface when we want to test a component that depends on the interface
- We use a mock object of the interface when we want to test the communication between collaborating objects

Value Objects

- Are immutable
- Cannot be doubled because they are `final`
- This is not a problem: doubling them would be pointless

Services

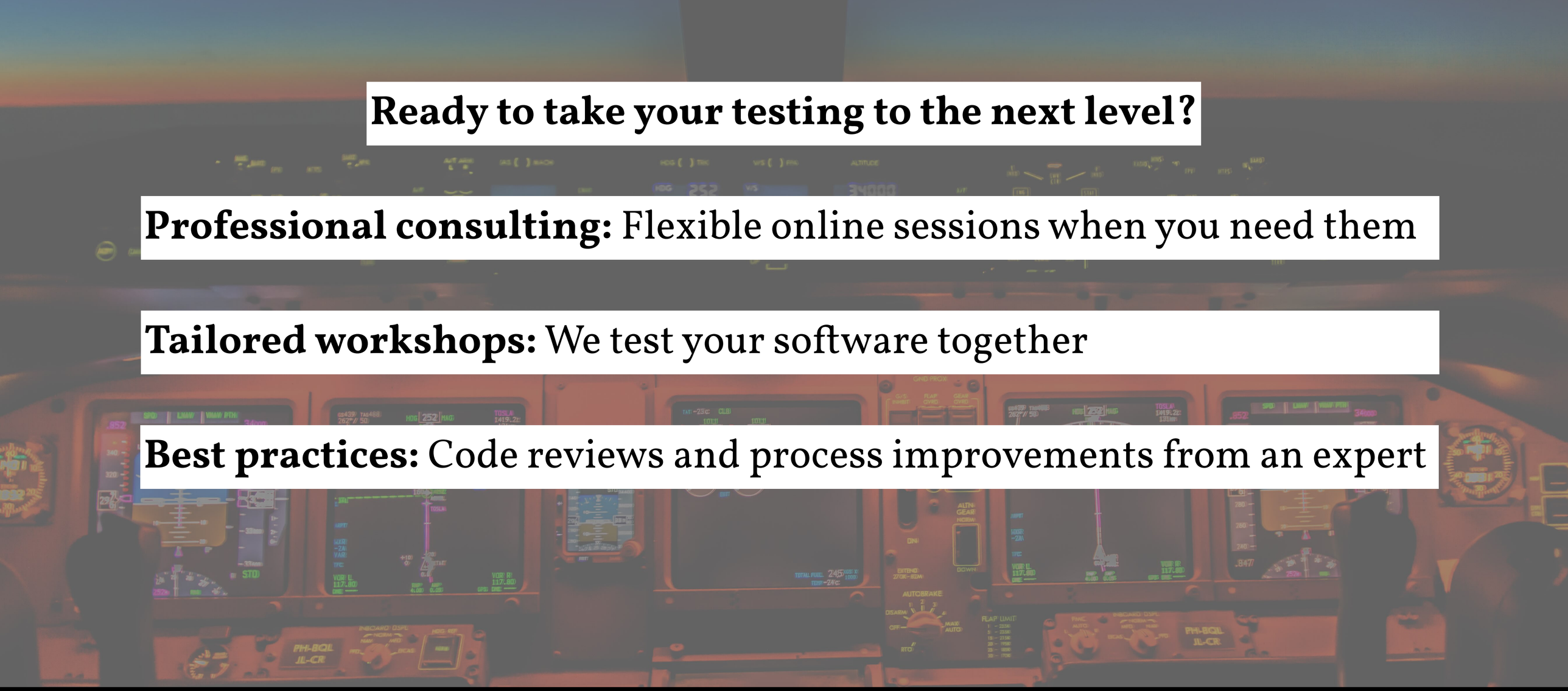
- Implement an interface that can be doubled
- We use a real implementation of the interface when we want to test this implementation
- We use a test stub of the interface when we want to test a component that depends on the interface
- We use a mock object of the interface when we want to test the communication between collaborating objects

Newable Objects

- Do not need to be doubled
- We double the repository etc. instead to isolate from infrastructure

This presentation has a home on the web:

thePHP.cc



Ready to take your testing to the next level?

Professional consulting: Flexible online sessions when you need them

Tailored workshops: We test your software together

Best practices: Code reviews and process improvements from an expert

https://phpunit.expert/consulting-and-coaching

Thank you!

https://phpunit.expert

sebastian@thephp.cc

@sebastian@phpc.social

thePHP.cc

Image Credits

- https://www.pexels.com/de-de/foto/foto-von-auf-dem-fluss-geparkten-booten-2031706
- https://www.pexels.com/photo/walnuts-near-walnut-cracker-3358735
- https://www.pexels.com/de-de/foto/foto-von-leuten-die-stehen-wahrend-sie-diskutieren-3182772