Prophecy

– Mocking Framework –

@bicpi

@bicpi

```
{
    "name": "Philipp Rieber",
    "twitter": "@bicpi",
    "url": "http://philipp-rieber.net",
    "location": "Munich",
    "skills": ["PHP", "Symfony", "DevOps", "Writer"],
    "job": "PAYMILL",
    "status": "DDD"
}
```

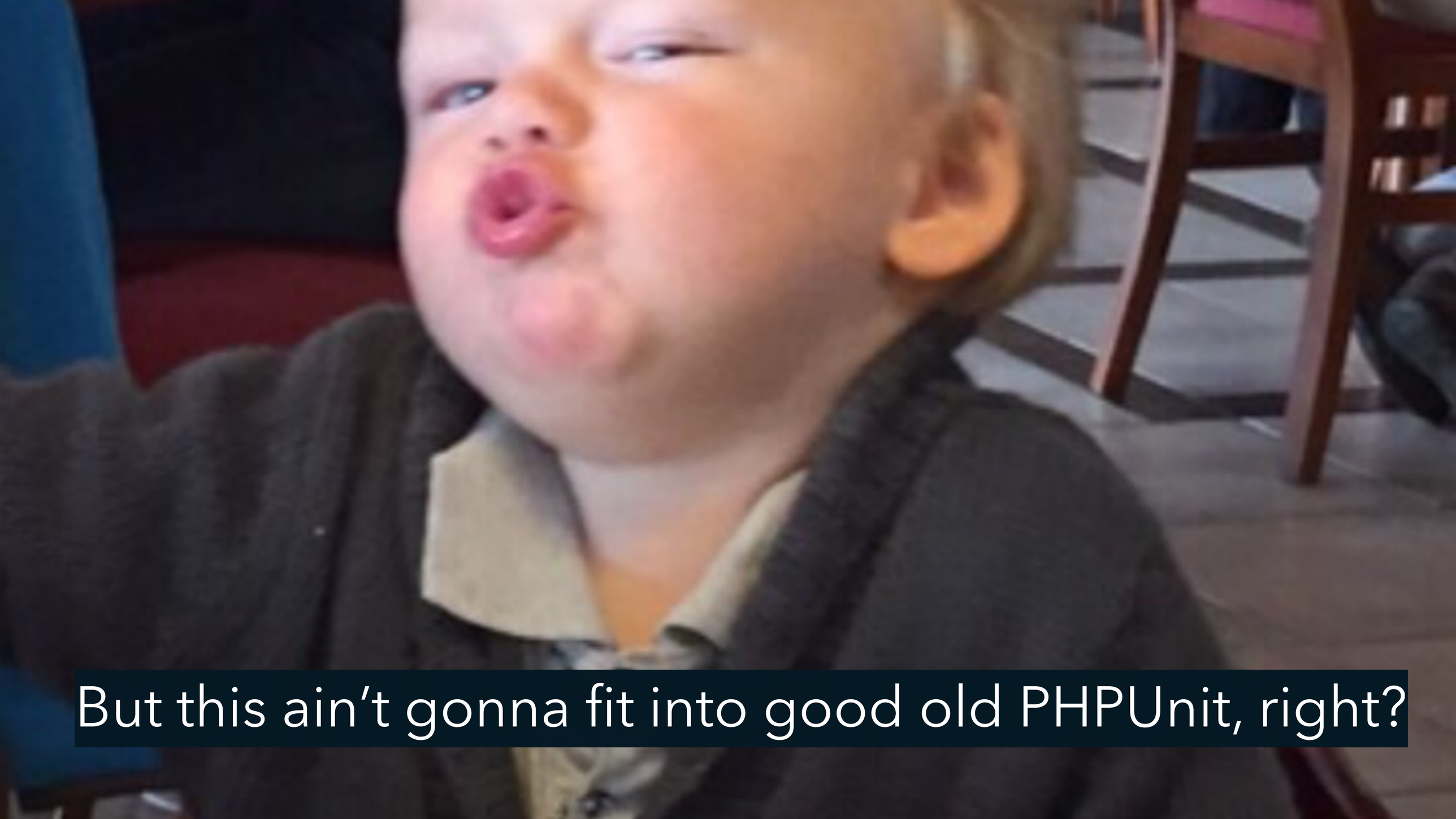
phpspec/prophecy

– Mocking framework –

@everzet • @_md • @CiaranMcNulty • @phpspec

Why?

“Modern PHP”


But this ain’t gonna fit into good old PHPUnit, right?

Well, yes.

PHPUnit >= 4.5

Some terminology first

A mocking framework creates

Test Doubles

Types of Doubles

Dummy · Stub · Mock · Spy

Dummy

– Doubles without behaviour –

Stub

– Doubles with behaviour, but no expectations –

Mock

– Doubles with expectations –

Spy

– Verify expected behaviour –

How does it look like?

```
$ composer req --dev phpspec/prophecy
```

Create a prophet

```php
$prophet = new \Prophecy\Prophet();
```

Create a prophecy

```php
$prophecy = $prophet->prophesize(ClassOrInterface::class);
```

Reveal the prophecy

```
$double = $prophecy->reveal();
```

All together

```php
$prophet = new \Prophecy\Prophet();

$prophecy = $prophet->prophesize(ClassOrInterface::class);

$double = $prophecy->reveal();
```

Examples please

Dummy – Example

Create an object that just doubles `\Psr\Log\LoggerInterface`

Dummy

```php
use Psr\Log\LoggerInterface;

$loggerProphecy = $prophet->prophesize(LoggerInterface::class);

$logger = $loggerProphecy->reveal();
```

Dummy

```php
use Psr\Log\LoggerInterface;

$loggerProphecy = $prophet->prophesize(LoggerInterface::class);

$logger = $loggerProphecy->reveal();

// All the mothods return null now
$logger->alert();
$logger->debug();
// ...
```

Stub – Example

Create an object that doubles `\Guzzle\HttpClient`

… and model the behaviour of a `GET` request to `/profile`

Stub

```
use GuzzleHttp\Client;

$prophecy = $prophet->prophesize(Client::class);
$prophecy
    ->get('/profile')
    ->willReturn('{"username": "bicpi"}');
```

Stub

```php
use GuzzleHttp\Client;

$prophecy = $prophet->prophesize(Client::class);
$prophecy
    ->get('/profile')
    ->willReturn('{"username": "bicpi"}');

$httpClient = $prophecy->reveal();

// Always returns {"username": "bicpi"} now
$httpClient->get('/profile');
```

Stub

Some more examples

Stub

```
$prophecy = $prophet->prophesize(Sequence::class);
$prophecy
    ->next()
    ->willReturn(1, 2, 3);

$sequence = $prophecy->reveal();

$sequence->next(); // 1
$sequence->next(); // 2
$sequence->next(); // 3
```

Stub

```
$prophecy = $prophet->prophesize(Validator::class);
$prophecy
    ->validate()
    ->willThrow(ValidationException::class);

$validator = $prophecy->reveal();

// throws ValidationException
$validator->validate();
```

Stub

```
$prophecy = $prophet->prophesize(ArrayValidator::class);
$prophecy
    ->validate(Argument::type('string'))
    ->willThrow(ValidationException::class);

$prophecy
    ->validate(Argument::type('array'))
    ->willReturn(null);
```

Stub – Complex behaviours

```
$prophecy
     ->method(…)
     ->will($callback);
```

Stub – Complex behaviours

```
$userProphecy
    ->setName('Philipp')
    ->will(function () {
        $this->getName()->willReturn('Philipp');
    });
```

Stub – Complex behaviours

```
$companyProphecy
    ->setName(Argument::type('string'))
    ->will(function ($args) {
        $this->getName()->willReturn($args[0]);
    });
```

Mock – Example

Create an object that doubles `\Doctrine\ORM\EntityManagerInterface` and expect the `persist()` method for a `User` entity to be called

Mock

```
use Doctrine\ORM\EntityManagerInterface;

$prophecy = $prophet->prophesize(EntityManagerInterface::class);
$prophecy
    ->persist(Argument::type(User::class))
    ->shouldBeCalled();

$entityManager = $prophecy->reveal();
```

Mock

```php
$prophecy = $prophet->prophesize(EntityManagerInterface::class);
$prophecy
    ->persist(Argument::type(User::class))
    ->shouldBeCalled();

$entityManager = $prophecy->reveal();

// … do the tests

// Check the predictions
$prophet->checkPredictions();
```

Mock

```
$prophecy = $prophet->prophesize(EntityManagerInterface::class);
$prophecy
    ->persist(Argument::type(User::class))
    ->shouldNotBeCalled();

$entityManager = $prophecy->reveal();

// … do the tests

// Check the predictions
$prophet->checkPredictions();
```

Mock

```
$prophecy = $prophet->prophesize(EntityManagerInterface::class);
$prophecy
    ->persist(Argument::type(User::class))
    ->shouldBeCalledTimes(3);

$entityManager = $prophecy->reveal();

// … do the tests

// Check the predictions
$prophet->checkPredictions();
```

Spy

Check recorded method calls of a double

Spy

```php
$prophecy = $prophet->prophesize(EntityManagerInterface::class);
$entityManager = $prophecy->reveal();

// During the tests ...
$entityManager->persist($someEntity);

$entityManager
    ->persist()
    ->shouldHaveBeenCalled();

// Check the predictions
$prophet->checkPredictions();
```

Put everything together

```php
$prophet = new \Prophecy\Prophet();

$prophecy = $prophet->prophesize(ClassOrInterface::class);

// Make predictions  ...

$double = $prophecy->reveal();

$prophet->checkPredictions();
```

Prophecy in PHPUnit

```
class SomeTest extends \PHPUnit_Framework_TestCase {
    private $prophet;

    protected function setUp() {
        $this->prophet = new \Prophecy\Prophet;
    }

    public function testSomething() {
        $prophecy = $this->prophet->prophesize(...);
    }

    protected function tearDown() {
        $this->prophet->checkPredictions();
    }
}
```

Prophecy in PHPUnit – built-in support

```
class SomeTest extends PHPUnit_Framework_TestCase
{
    public function testSomething()
    {
        $prophecy = $this->prophesize(...);
    }
}
```

Nesting prophecies

Prophecy is extensible

PHPUnit Autocomplete Assistant

– for PhpStorm –

https://github.com/maxfilatov/phpuaca

https://github.com/phpspec/prophecy