Leveling Up With Unit Testing

Mark Niebergall
Dutch PHP 2024

👏 Thank you!

- Organizers
- Attendees
- Speakers
- Sponsors
  - Nucleus Security

🙋 Survey

- Experience with Unit Testing?

📋 Housekeeping

- Full-day tutorial
  - Concepts
  - Coding

📋 Housekeeping

- We’ll have breaks
- Stand up, stretch, move about as needed
- Ask questions!
- Open for discussions
- Opinions

📋 Housekeeping

- Schedule
  - 9:00am Unit testing presentation
  - 10:30am Break
  - 10:40am Writing tests
  - 12:30pm Lunch
  - 1:30pm TDD
  - 3:00pm Break
  - 3:10pm Group tests
  - 4:00pm Tests for Open Source projects
  - 5:00pm End


HALYARD
STERLING NITRILE
HALYARD
STERLING NITRILE

https://analyze.co.za/wp-content/uploads/2018/12/441-1170x500.jpg

✔ Objective

✓ Be familiar with how to setup PHPUnit
✓ Familiar with how to test existing code
✓ Know how to write unit tests using PHPUnit
✓ Convince team and management to leverage automated testing

👀 Overview

- 😀 Benefits of Unit Testing
- 🧪 Test Types
- ⚙️ PHPUnit Setup
- 👨‍💻 Writing Unit Tests
- ⌨️ Testing Existing Code

😀 Benefits of Unit Testing

😀 Benefits of Unit Testing

```php
public static function add($a, $b)
{
    return $a + $b;
}
```

😀 Benefits of Unit Testing

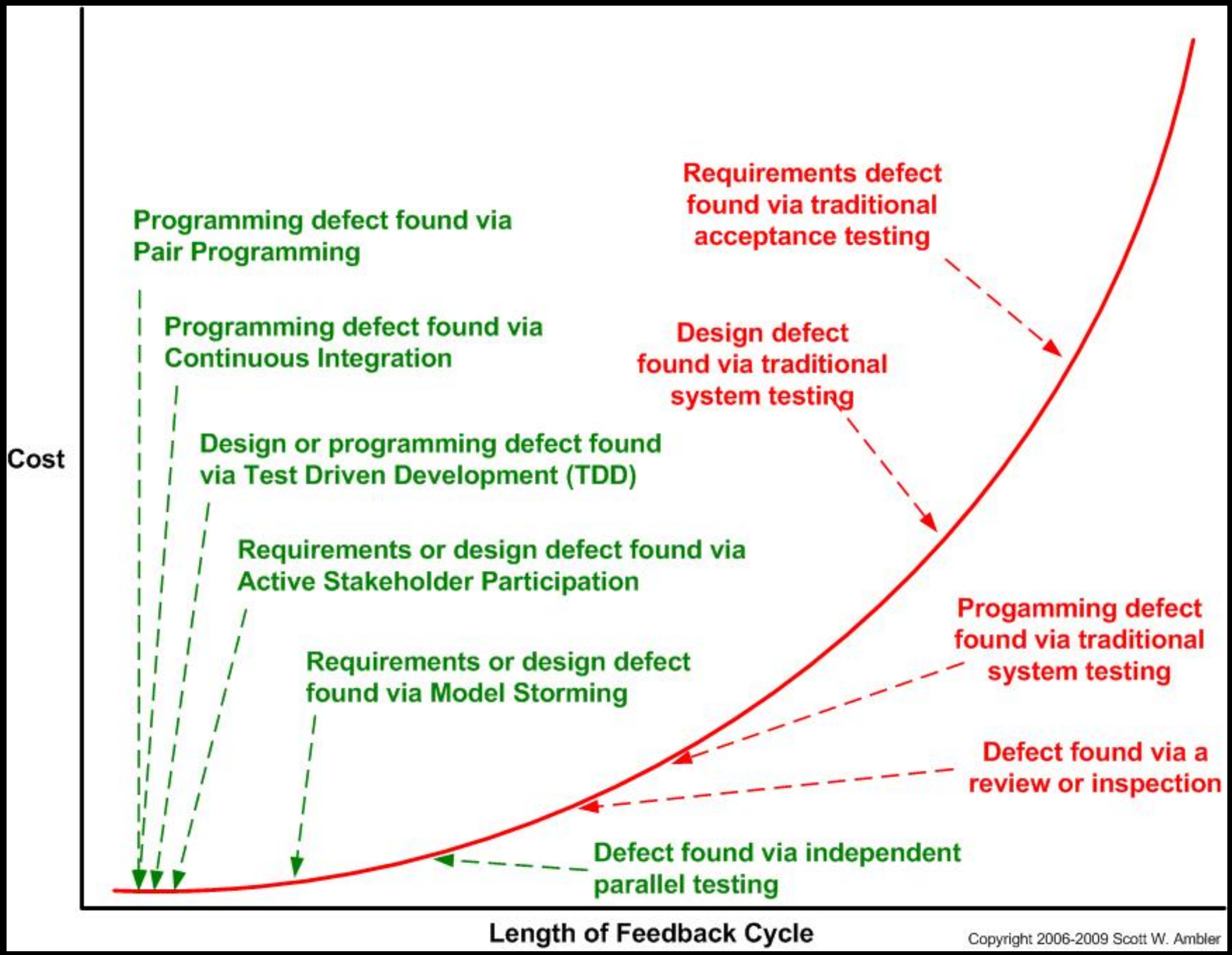
```php
public static function add($a, $b)
{
    return $a + $b;
}

public function add(float ...$numbers): float
{
    $return = 0;

    foreach ($numbers as $value) {
        $return = bcadd(
            (string) $return,
            (string) $value,
            10
        );
    }

    return (float) $return;
}
```

😀 Benefits of Unit Testing



http://www.ambysoft.com/artwork/comparingTechniques.jpg

😀 Benefits of Unit Testing

- 💻 Automated way to test code
  - Regression Testing

😀 Benefits of Unit Testing

- 💻 Automated way to test code
  - Continuous Integration (CI)
  - Continuous Deployment (CD)

😀 Benefits of Unit Testing

- 💻 Automated way to test code
  - Other ways to automatically test code
    - Behavioral (BDD): behat, phpspec
    - Functional
    - Acceptance: Selenium
    - Others?

😀 Benefits of Unit Testing

- 🐞 Decrease bugs introduced with code
  - Decreased time to deployment
  - Better use of QA team time

😀 Benefits of Unit Testing

- 🐞 Decrease bugs introduced with code
  - High confidence in delivered code

😀 Benefits of Unit Testing

- 💯 Confidence when refactoring
  - Tests covering code being refactored
  - TDD
    - Change tests
    - Tests fail
    - Change code
    - Tests pass

🧪 Test Types

🧪 Test Types

- Types
  - Manual
  - Automated

🧪 Test Types

- Manual
  - User clicking buttons
  - User typing input
  - User consuming an API

🧪 Test Types

- Manual
  - Catch business logic issues

🧪 Test Types

- Manual
  - Time consuming

🧪 Test Types

- Manual
  - Error prone
  - Miss errors

🧪 Test Types

- Manual
  - Overhead cost
  - Time

🧪 Test Types

- Automated
  - Repeatable

🧪 Test Types

- Automated
  - Much faster than manual interactions

🧪 Test Types

- Automated
  - Only as good as quality of tests written

🧪 Test Types

- Categories
  - Unit
  - Integration
  - Functional
  - End-to-end
  - Acceptance
  - Performance
  - Smoke

🧪 Test Types

- Unit
  - Test individual code components
  - Isolated with no external resources or calls
  - Run very fast

🧪 Test Types

- Integration
  - Internal resources
  - Test code interactions

🧪 Test Types

- Functional
  - Business requirements
  - API call responses

🧪 Test Types

- End-to-end
  - User interactions with the application

🧪 Test Types

- Acceptance
  - Verify requirements are met
    - QA
    - Stakeholder

🧪 Test Types

- Performance
  - System under load
  - Scalability

🧪 Test Types

- Smoke
  - Basic checks
  - Useful before large test suites or after refactors

⚙️ PHPUnit Setup

⚙️ PHPUnit Setup

- Install via composer
- Setup `phpunit.xml` for configuration (if needed)
- Run unit tests

⚙️ PHPUnit Setup

- phpunit/phpunit

⚙️ PHPUnit Setup

```
composer require --dev phpunit/phpunit 11.0.5
```

⚙️ PHPUnit Setup

- File phpunit.xml
  - PHPUnit configuration for that project
  - Documentation: https://docs.phpunit.de/en/11.0/organizing-tests.html#composing-a-test-suite-using-xml-configuration
  - vendor/bin/phpunit --generate-configuration

⚙️ PHPUnit Setup

```xml
<?xml version="1.0" encoding="UTF-8"?>
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/11.0/
phpunit.xsd"
         bootstrap="vendor/autoload.php"
         cacheDirectory=".phpunit.cache"
         executionOrder="depends,defects"
         failOnRisky="true"
         failOnWarning="true"
         colors="true">
    <testsuites>
        <testsuite name="default">
            <directory>tests</directory>
        </testsuite>
    </testsuites>
    <source restrictDeprecations="true" restrictNotices="true"
restrictWarnings="true">
        <include>
            <directory>src</directory>
        </include>
    </source>
    <coverage>
        <report>
            <html outputDirectory="tests/report/"/>
        </report>
    </coverage>
</phpunit>
```

⚙️ PHPUnit Setup

- 💻 Running PHPUnit

vendor/bin/phpunit tests/

⚙️ PHPUnit Setup

- 💻 Running PHPUnit

```
vendor/bin/phpunit tests/
PHPUnit 11.0.3 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.8
Configuration: /Users/markniebergall/Documents/projects/phpunit/
phpunit.xml

...........                                       11 / 11 (100%)

Time: 00:00.015, Memory: 8.00 MB

OK (11 tests, 15 assertions)
```

⚙️ PHPUnit Setup

- 💻 Running PHPUnit
  - Within IDE
    - PhpStorm
      - https://www.jetbrains.com/help/phpstorm/using-phpunit-framework.html

⚙️ PHPUnit Setup

- 💻 Running PHPUnit
  - Code Coverage Report
    - Need Xdebug or PCOV
      - Xdebug
        - export XDEBUG_MODE=coverage
        - xdebug.mode=coverage

⚙️ PHPUnit Setup

- 📂 Directory Structure
  - PHP files in src/
    - Ex: src/Math/Adder.php
      - namespace Org\Project\Math\Adder;
  - tests in tests/src/, ‘Test’ at end of filename
    - Ex: tests/src/Math/AdderTest.php
      - namespace Org\Project\Tests\Math\Adder;

👨‍💻 Writing Unit Tests

👨‍💻 Writing Unit Tests

```php
public function add(float ...$numbers): float
{
    $return = 0;

    foreach ($numbers as $value) {
        $return = bcadd(
            (string) $return,
            (string) $value,
            10
        );
    }

    return (float) $return;
}
```

👨‍💻 Writing Unit Tests

```php
use PHPUnit\Framework\TestCase;

class AdderTest extends TestCase
{
    protected Adder $adder;

    public function setUp(): void
    {
        $this->adder = new Adder();
    }

    public function testAdderWithSetup(): void
    {
        $sum = $this->adder->add(3, 7);

        $this->assertSame(10.0, $sum);
    }
```

👨‍💻 Writing Unit Tests

```php
public function testAdderThrowsExceptionWhenNotANumber()
{
    $this->expectException(TypeError::class);

    $adder = new Adder();
    $adder->add(7, 'Can\'t add this');
}
```

👨‍💻 Writing Unit Tests

```php
public function testAdderAddsIntegers(): void
{
    $adder = new Adder();
    $sum = $adder->add(7, 3, 5, 5, 6, 4, 1, 9);

    $this->assertSame(40.0, $sum);
}
```

```php
public function testAdderAddsDecimals(): void
{
    $adder = new Adder();
    $sum = $adder->add(1.5, 0.5);

    $this->assertSame(2.0, $sum);
}
```

👨‍💻 Writing Unit Tests

```php
#[DataProvider('dataProviderNumbers')]
public function testAdderAddsNumbers(
    float $expectedSum,
    ...$numbers
): void {
    $adder = new Adder();
    $sum = $adder->add(...$numbers);
    $this->assertSame($expectedSum, $sum);
}

public static function dataProviderNumbers(): array
{
    return [
        [2, 1, 1],
        [2, 1.5, 0.5],
    ];
}
```

👨‍💻 Writing Unit Tests

```php
#[DataProvider('dataProviderNumbers')]
public function testAdderAddsNumbers(
    float $expectedSum,
    ...$numbers
): void {
    $adder = new Adder();
    $sum = $adder->add(...$numbers);
    $this->assertSame($expectedSum, $sum);
}

public static function dataProviderNumbers(): iterable
{
    yield 'integer' => [2, 1, 1];

    yield 'integers with decimals' => [2, 1.5, 0.5];
}
```

👨‍💻 Writing Unit Tests

```php
/**
 * @given an Adder
 * @when adding numbers together
 * @then sum is accurately calculated
 *
 * @param float $expectedSum
 * @param ...$numbers
 *
 * @return void
 */
#[DataProvider('dataProviderNumbers')]
public function testAdderAddsNumbers(
    float $expectedSum,
    ...$numbers
): void {
```

👨‍💻 Writing Unit Tests

- 📏 Test Coverage
  - Percent of code covered by tests
  - Not aiming for 100%
  - No need to test language constructs
  - Open in browser: tests\report\index.html

🤦 Writing Unit Tests

	Code Coverage									
	Lines			Functions and Methods				Classes and Traits		
Total		65.12%	28 / 43		50.00%	4 / 8	CRAP		0.00%	0 / 1
UserEntity		65.12%	28 / 43		50.00%	4 / 8	39.72		0.00%	0 / 1
__construct		100.00%	7 / 7		100.00%	1 / 1	1			
validateUserId		100.00%	4 / 4		100.00%	1 / 1	3			
validateUsername		100.00%	4 / 4		100.00%	1 / 1	2			
validateEmailAddress		100.00%	4 / 4		100.00%	1 / 1	2			
validateFirstName		25.00%	1 / 4		0.00%	0 / 1	6.80			
validateLastName		25.00%	1 / 4		0.00%	0 / 1	6.80			
validatePhoneNumber		20.00%	2 / 10		0.00%	0 / 1	12.19			
validateLastActivity		83.33%	5 / 6		0.00%	0 / 1	3.04			

```php
<?php

declare(strict_types=1);

namespace User;

use DateTime;
use InvalidArgumentException;

class UserEntity
{
    public function __construct(
        public readonly ?int $userId,
        public readonly string $username,
        public readonly string $emailAddress,
        public readonly ?string $firstName = null,
        public readonly ?string $lastName = null,
        public readonly ?string $phoneNumber = null,
        public readonly bool $isEnabled = false,
        public readonly ?DateTime $lastActivity = null,
    ) {
        $this->validateUserId($this->userId);
        $this->validateUsername($this->username);
        $this->validateEmailAddress($this->emailAddress);
        $this->validateFirstName($this->firstName);
        $this->validateLastName($this->lastName);
        $this->validatePhoneNumber($this->phoneNumber);
        $this->validateLastActivity($this->lastActivity);
    }

    protected function validateUserId(?int $userId): void
    {
        if ($userId !== null && $userId < 1) {
            throw new InvalidArgumentException(
                'userId must be a non-negative int'
            );
        }
```

👨‍💻 Writing Unit Tests

- ⛔ Self-contained
  - No actual database connections
  - No API calls should occur
  - No external code should be called
    - Use testing framework

👨‍💻 Writing Unit Tests

- 🏗️ Fixtures
  - setUp
    - Called once per function
  - tearDown
    - Called once per function

👨‍💻 Writing Unit Tests

- 🏗️ Fixtures
  - setUpBeforeClass
    - Called once per class
  - tearDownAfterClass
    - Called once per class

👨‍💻 Writing Unit Tests

- ✅ Assertions

```php
$this->assertInstanceOf(Response::class, $response);

$this->assertEquals(200, $response->getStatusCode());

$this->assertSame(401, $responseActual->getStatusCode());

$this->assertTrue($dispatched);

$this->assertFalse($sent);
```

👨‍💻 Writing Unit Tests

- ✅ Assertions

```php
$this->expectException(RuntimeException::class);

$this->expectExceptionCode(403);

$this->expectExceptionMessage('Configuration not found.');
```

👨‍💻 Writing Unit Tests

- 🔀 Mocking
  - Built-in Mock Objects
  - Prophecy
  - Mockery

👨‍💻 Writing Unit Tests

- ⚠️ Scratching the surface
  - Dive deep into each
  - See what you like, what works
  - Leverage a mix

🤦 Writing Unit Tests

- Built-in Mock Objects
  - Only mock certain methods, let others run as-is
  - Pattern:

```
->expects($this->once())
->method('sendEmail')
->with($email, $message)
->willReturn($value)
```

```php
use PHPUnit\Framework\TestCase;

class RectangleTest extends TestCase
```

👨‍💻 Writing Unit Tests

- Built-in Mock Objects

```php
$adderMock = $this->createMock(Adder::class);
$adderMock
    ->expects($this->once())
    ->method('add')
    ->with($length, $width)
    ->willReturn(10.34001);

$multiplierMock = $this->createMock(Multiplier::class);
$multiplierMock
    ->expects($this->once())
    ->method('multiply')
    ->with(2, 10.34001)
    ->willReturn(20.68002);

$rectangle = new Rectangle(
    $length,
    $width,
    $adderMock,
    $multiplierMock,
);
```

👨‍💻 Writing Unit Tests

- Prophecy
  - Pattern: ->shouldBeCalled()->willReturn($value)

👨‍💻 Writing Unit Tests

- Prophecy
  - Mock objects
  - Expected method calls
  - Reveal object when injecting

```php
use PHPUnit\Framework\TestCase;
use Prophecy\PhpUnit\ProphecyTrait;

class RectangleTest extends TestCase
{
    use ProphecyTrait;
```

👨‍💻 Writing Unit Tests

- Prophecy

```
$adderMock = $this->prophesize(Adder::class);
$multiplierMock = $this->prophesize(Multiplier::class);

$adderMock
    ->add($length, $width)
    ->shouldBeCalled()
    ->willReturn(10.34001);

$multiplierMock
    ->multiply(2, 10.34001)
    ->shouldBeCalled()
    ->willReturn(20.68002);

$rectangle = new Rectangle(
    $length,
    $width,
    $adderMock->reveal(),
    $multiplierMock->reveal()
);
```

👨‍💻 Writing Unit Tests

- Prophecy

```
$dbMock->fetchRow(Argument::any())
    ->shouldBeCalled()
    ->willReturn([]);

$asyncBusMock
    ->dispatch(
        Argument::type(DoSomethingCmd::class),
        Argument::type('array')
    )
    ->shouldBeCalled()
    ->willReturn((new Envelope(new \stdClass())));
```

👨‍💻 Writing Unit Tests

- Mockery
  - Both built-in and Prophecy styles

👨‍💻 Writing Unit Tests

- Mockery

```
$adderMock = Mockery::mock(Adder::class);
$multiplierMock = Mockery::mock(Multiplier::class);

$adderMock
    ->shouldReceive('add')
    ->with($length, $width)
    ->andReturn(10.34001);

$multiplierMock
    ->shouldReceive(‘multiply')
    ->with(2, 10.34001)
    ->andReturn(20.68002);

$rectangle = new Rectangle(
    $length,
    $width,
    $adderMock,
    $multiplierMock
);
```

👨‍💻 Writing Unit Tests

- Mockery

```
$adderMock = Mockery::mock(Adder::class);
$multiplierMock = Mockery::mock(Multiplier::class);

$adderMock
    ->expects()
    ->add($length, $width)
    ->andReturns(10.34001);

$multiplierMock
    ->expects()
    ->multiply(2, 10.34001)
    ->andReturn(20.68002);

$rectangle = new Rectangle(
    $length,
    $width,
    $adderMock,
    $multiplierMock
);
```

⌨️ Testing Existing Code

⌨️ Testing Existing Code

- ⚠️ Problematic Patterns
  - Long and complex functions

⌨️ Testing Existing Code

- ⚠️ Problematic Patterns
  - Missing Dependency Injection (DI)
    - `new Thing();` in functions to be tested

⌨️ Testing Existing Code

- ⚠️ Problematic Patterns
  - exit
  - die
  - print_r
  - var_dump
  - echo
  - other outputs in-line
    - Hint: use expectOutputString

⌨️ Testing Existing Code

- ⚠️ Problematic Patterns
  - Time-sensitive
  - sleep

⌨️ Testing Existing Code

- ⚠️ Problematic Patterns
  - Database interactions
  - Resources

⌨️ Testing Existing Code

- ⚠️ Problematic Patterns
  - Out of class code execution
  - Functional code

⌨️ Testing Existing Code

- ✅ Helpful Patterns
  - Unit testing promotes good code patterns

⌨️ Testing Existing Code

- ✅ Helpful Patterns
  - Dependency Injection
  - Classes and functions focused on one thing
  - Abstraction
  - Interfaces
  - Clean code

⌨️ Testing Existing Code

- ✅ Helpful Patterns
  - Code that is SOLID
    - Single-responsibility: every class should have only one responsibility
    - Open-closed: should be open for extension, but closed for modification
    - Liskov substitution: in PHP, use interface/abstract
    - Interface segregation: Many client-specific interfaces are better than one general-purpose interface
    - Dependency inversion: Depend upon abstractions, [not] concretions

⌨️ Testing Existing Code

- ✅ Helpful Patterns
  - DDD

⌨️ Testing Existing Code

- ✅ Helpful Patterns
  - KISS
  - DRY
  - YAGNI

⌨️ Testing Existing Code

```php
class ShapeService
{
    public function create(string $shape): int
    {
        $db = new Db();
        return $db->insert('shape', ['shape' => $shape]);
    }

    public function smsArea(Rectangle $shape, string $toNumber): bool
    {
        $sms = new Sms([
            'api_uri' => 'https://example.com/sms',
            'api_key' => 'alkdjfoasifj0392lkdsjf',
        ]);

        $sent = $sms->send($toNumber, 'Area is ' . $shape->area());

        (new Logger())
            ->log('Sms sent to ' . $toNumber . ': Area is ' . $shape->area());

        return $sent;
    }
}
```

⌨️ Testing Existing Code

```php
class ShapeService
{
    public function create(string $shape): int
    {
        $db = new Db();
        return $db->insert('shape', ['shape' => $shape]);
    }

    public function smsArea(Rectangle $shape, string $toNumber): bool
    {
        $sms = new Sms([
            'api_uri' => 'https://example.com/sms',
            'api_key' => 'alkdjfoasifj0392lkdsjf',
        ]);

        $sent = $sms->send($toNumber, 'Area is ' . $shape->area());

        (new Logger())
            ->log('Sms sent to ' . $toNumber . ': Area is ' . $shape->area());

        return $sent;
    }
}
```

⌨️ Testing Existing Code

```php
class ShapeService
{
    public function __construct(
        protected Db $db,
        protected Sms $sms
    ) {}

    public function create(string $shape): int
    {
        return $this->db->insert('shape', ['shape' => $shape]);
    }

    public function smsArea(ShapeInterface $shape, string $toNumber): bool
    {
        $area = $shape->area();
        return $this->sms->send($toNumber, 'Area is ' . $area);
    }
```

⌨️ Testing Existing Code

```php
/**
 * @given a shape
 * @when inserting the shape in the database
 * @then database insert is called for the shape
 *
 * @return void
 */
public function testCreate(): void
{
    $dbMock = $this->createMock(Db::class);
    $smsMock = $this->createMock(Sms::class);

    $shape = 'square';
    $dbMock
        ->expects($this->once())
        ->method('insert')
        ->with('shape', ['shape' => $shape])
        ->willReturn(1);

    $shapeServiceCleanedUp = new ShapeServiceCleanedUp(
        $dbMock,
        $smsMock
    );

    $shapeServiceCleanedUp->create($shape);
}
```

⌨️ Testing Existing Code

```php
public function testSmsArea(): void
{
    $dbMock = $this->createMock(Db::class);
    $smsMock = $this->createMock(Sms::class);
    $shapeMock = $this->createMock(ShapeInterface::class);

    $area = 16.4;
    $shapeMock
        ->expects($this->once())
        ->method('area')
        ->willReturn($area);

    $toNumber = '18005551234';
    $smsMock
        ->expects($this->once())
        ->method('send')
        ->with($toNumber, 'Area is ' . $area)
        ->willReturn(true);

    $shapeServiceCleanedUp = new ShapeServiceCleanedUp(
        $dbMock,
        $smsMock
    );

    $shapeServiceCleanedUp->smsArea(
        $shapeMock,
        $toNumber
    );
}
```

💬 Discussion Items

- Convincing Teammates
- Convincing Management

💬 Discussion Items

- Does unit testing slow development down?

💬 Discussion Items

- “I don’t see the benefit of unit testing”

💬 Discussion Items

- Unit tests for legacy code

💬 Discussion Items

- Other?

📝 Review

- 😀 Benefits of Unit Testing
- 🧪 Test Types
- ⚙️ PHPUnit Setup
- 👨‍💻 Writing Unit Tests
- ⌨️ Testing Existing Code

Leveling Up With Unit Testing

- 🙋 Questions?

Mark Niebergall @mbniebergall

- PHP since 2005
- Masters degree in MIS
- Senior Software Engineer
- Vulnerability Management project (security scans)
- Utah PHP Co-Organizer
- CSSLP, SSCP Certified and Exam Developer
- Endurance sports, outdoors

Mark Niebergall @mbniebergall

✏️ References

- https://docs.phpunit.de/en/11.0/index.html
- https://analyze.co.za/wp-content/uploads/2018/12/441-1170x500.jpg
- http://www.ambysoft.com/artwork/comparingTechniques.jpg
- https://en.wikipedia.org/wiki/SOLID
- https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Writing Tests

Writing Tests

- Scenario
  - User Entity

Writing Tests

- Scenario
  - MFA login
  - Send MFA code
    - SMS
    - Email

TDD

- Scenario
  - Is a number Fibonacci?

TDD

- Scenario
  - Password Checks
    - Not common
    - Not dictionary
      - https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10k-most-common.txt
    - At least 12 characters

Group Tests

- Simple scenarios to choose from
  - Circle circumference:
    - 2 * pi * radius
  - Fizz Buzz
    - 1-100, 3: Fizz, 5: Buzz, others: number
      - Ex: 1, 2, Fizz, 4, Buzz, 6, …, 14, Fizz Buzz, 16, …
  - Valid Date Checker

Group Tests

- Complex scenarios to choose from
  - Conference Schedule Builder
    - Attendee can select 1 session per time slot
  - Automatic driving car action decider
    - Based on what is seen, what should car do next
      - Stop? Accelerate? Yield? Turn?
  - Order Processor
  - Other?

Discussion

- Ideal code coverage
- TDD vs Tests after
- Unit vs Integration vs

Open Source Project Tests

Open Source Project Tests

- Write tests
- Improve code coverage

Open Source Project Tests

- GitHub
  - language:php
- Package
  - joindin
  - PHP-FIG
- Framework
- Tools
  - Guzzle
  - Faker