

LONGHORN  
PHP CONFERENCE

# DESIGNING TEST ARCHITECTURE THAT DOES NOT SUCK

MICHAEL BODNARCHUK @DAVERT

## **ABOUT ME**

- Michael Bodnarchuk **@davert**
- Web developer from Kyiv, Ukraine
- Lead developer of **Codeception** testing framework
- Also author of **CodeceptJS**, **Robo** and others
- Tech **Consultant**, CTO at **SDCLabs**

# I'M FROM



# **WE STRUGGLE REAL RUSSIAN INTERVENTION**



## WHAT OUR IT IS LIKE

- Outsource companies
- Magento office
- We ❤️ Symfony, Yii, Phalcon, Laravel
- IT engineers are rich!

# **TESTING BUSINESS EXPECTATIONS**

## **WHY DO WE TEST**

- To ensure software works as expected
- To discover bugs in software (before users)
- To measure performance
- To seek for security issues

## **WHAT TESTS WE CAN AUTOMATE**

- To ensure software works as expected
- ~~To discover bugs in software (before users)~~
- To measure performance
- ~~To seek for security issues~~

*We automate tests to execute them at any time*

## **AUTOMATED TESTING**

- To establish trust
- For constant changes
- To stabilize current codebase

# TESTING IS TOLD US TO BE LIKE THIS:

How to draw an owl

1.



2.



1. Draw some circles

2. Draw the rest of the fucking owl

We talk about how to test but we don't say

**WHAT TO TEST**

## PRIORITY FIRST

- Crucial **business scenarios**
- Security cases
- Algorithms, functions with complex logic
- Everything that is hard to test manually

## **TESTS SHOULD BE**

- Independent - not affect each other
- Atomic - concentrated on one feature

# **START WITH GENERAL**

Feature: customer registration

Background:

Given I am unregistered customer

Scenario: registering successfully

When I register

Then I should be registered

# ADD DETAILS

Scenario: registering successfully

When I register with

Name	davert	
Email	davert@sdclabs.com	
Password	123456	

Then I should be registered

And I receive confirmation email

## **QUALITIES OF A TEST**

1. Readability
2. Stability
3. Speed

# **READABILITY**

- TEST SHOULD BE EASY TO FOLLOW
- TEST SHOULD BE SIMPLE TO UPDATE
- CODE CAN BE REUSED TO TEST SIMILAR CASES

```
$request = $this->getRequest()
    ->setRequestUri('/user/profile/1')
    ->setParams(array('user_id'=>1));

$controller = $this->getMock(
    'UserController',
    array('render'),
    array($request, $response, $request->getParams())
);
$controller->expects($this->once())
    ->method('render')
    ->will($this->returnValue(true));

$this->assertTrue($controller->profileAction());
$this->assertTrue($controller->view->user_id == 1);
```



# **STABILITY**

- TEST SHOULD BE STABLE BY EXECUTION
- TEST SHOULD BE STABLE TO CHANGES

```
$mock = $this->getMock('Client', array('getInputFilter'));
$mock->expects($this->once()) // is it important?
    ->method('getInputFilter') // hardcoded method name
    ->will($this->returnValue($preparedFilterObject));

$formFactory = $this->getMock('Symfony\Component\Form\FormFactoryInterface');
$formFactory
    ->expects($this->once())
    ->method('create')
    ->will($this->returnValue($form))
```



# Codeception + WebDriver

```
// what if HTML changes?  
$I->click('//body/div[3]/p[1]/div[2]/div/span');  
  
// what if browser will render it longer?  
$I->wait(1);
```



## **HOW TO WRITE STABLE TESTS**

- **Don't mix specification with implementation**
- Focus on result, not on the path
- Use interfaces for tests

# **WE NEED A TABLE**



**JUST BUY IT**



[Blogpost: Expectation vs Implementation](#)

## **HOW TO WRITE STABLE TESTS**

- Don't mix specification with implementation
- **Focus on result, not on the path**
- Use interfaces for tests

## **FOCUS ON RESULT**

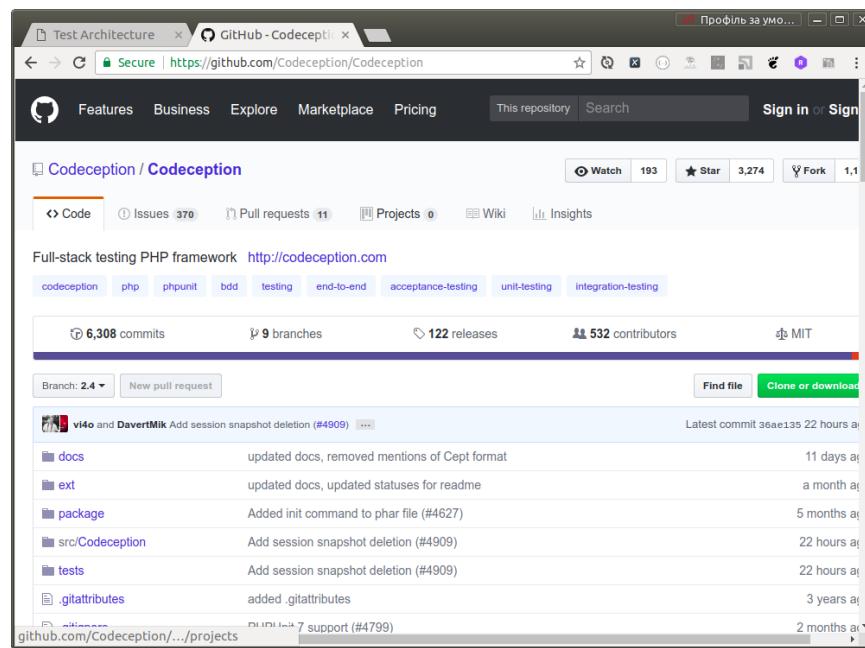
- Will the test have to duplicate exactly the application code?
- Will assertions in the test duplicate any behavior covered by library code?
- Is this detail important, or is it only an internal concern?

[Blogpost: The Right Way To Test React Components](#)

## **HOW TO WRITE STABLE TESTS**

- Don't mix specification with implementation
- Focus on result, not on the path
- Use interfaces for tests

# INTERFACES???



```
1 <?php
2 namespace App;
3
4 interface VeryImportantInterface
5 {
6     public function doSomethingCool();
7 }
8
9
```

## **WHAT ARE INTERFACES**

- Interface define rules to get things done
- Interfaces considered stable
- Interface is not just a keyword

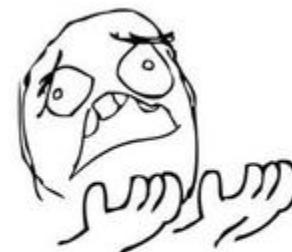
# 5 STAGES OF INTERFACE CHANGE



denial



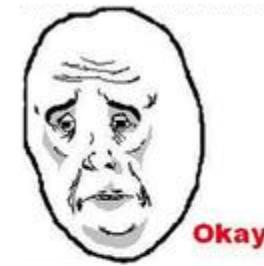
anger



bargaining



depression



acceptance

## **ANCHOR TESTS TO STABLE PARTS:**

- Web Interface
- Public API (REST, GraphQL, SOAP)
- PHP Interfaces
- Public Methods in Domain

**CONSIDER WHAT IS STABLE FOR YOU**

# CAN WE TEST PRIVATE METHODS?

- **Technically:** yes
- **Ideally:** no
- **Practically:** yes, if you consider them stable

**SPEED**

- **FOR ONE TEST CASE:**
  - fast enough for instant feedback
  - < 20 s
- **FOR ALL TESTS**
  - should be run on CI
  - easy to split into parallel processes
  - < 20 min

## **QUESTIONS TO BE ASKED**

- Should we sacrifice readability for speed?
- If so, why do you develop in PHP and not in C?

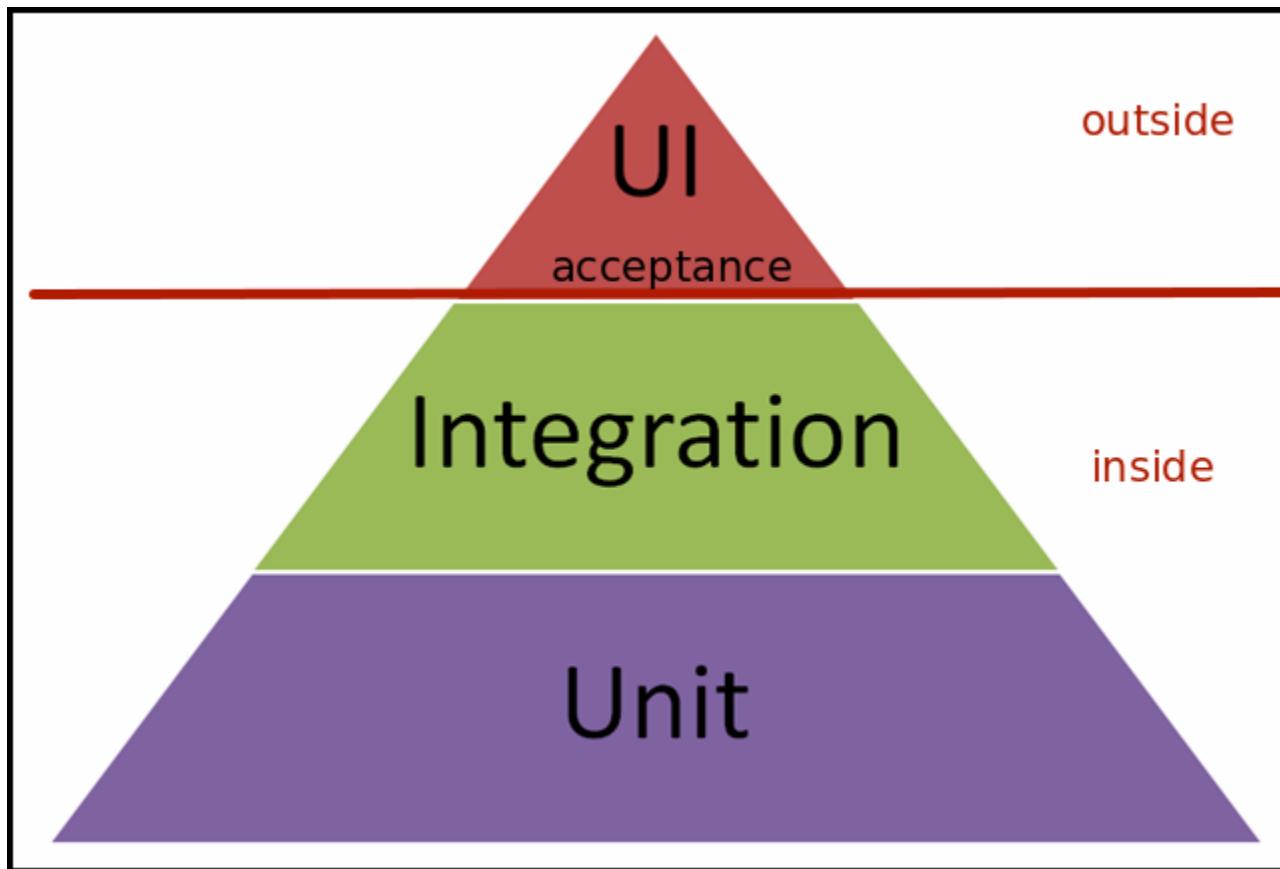
*Think how you can test a feature with minimal effort*

# **TEST INFRASTRUCTURE**

Let's talk about implementation

## **OUTER AND INNER TESTING**

- **Outer:** test from the public interface
- **Inner:** test from the source code

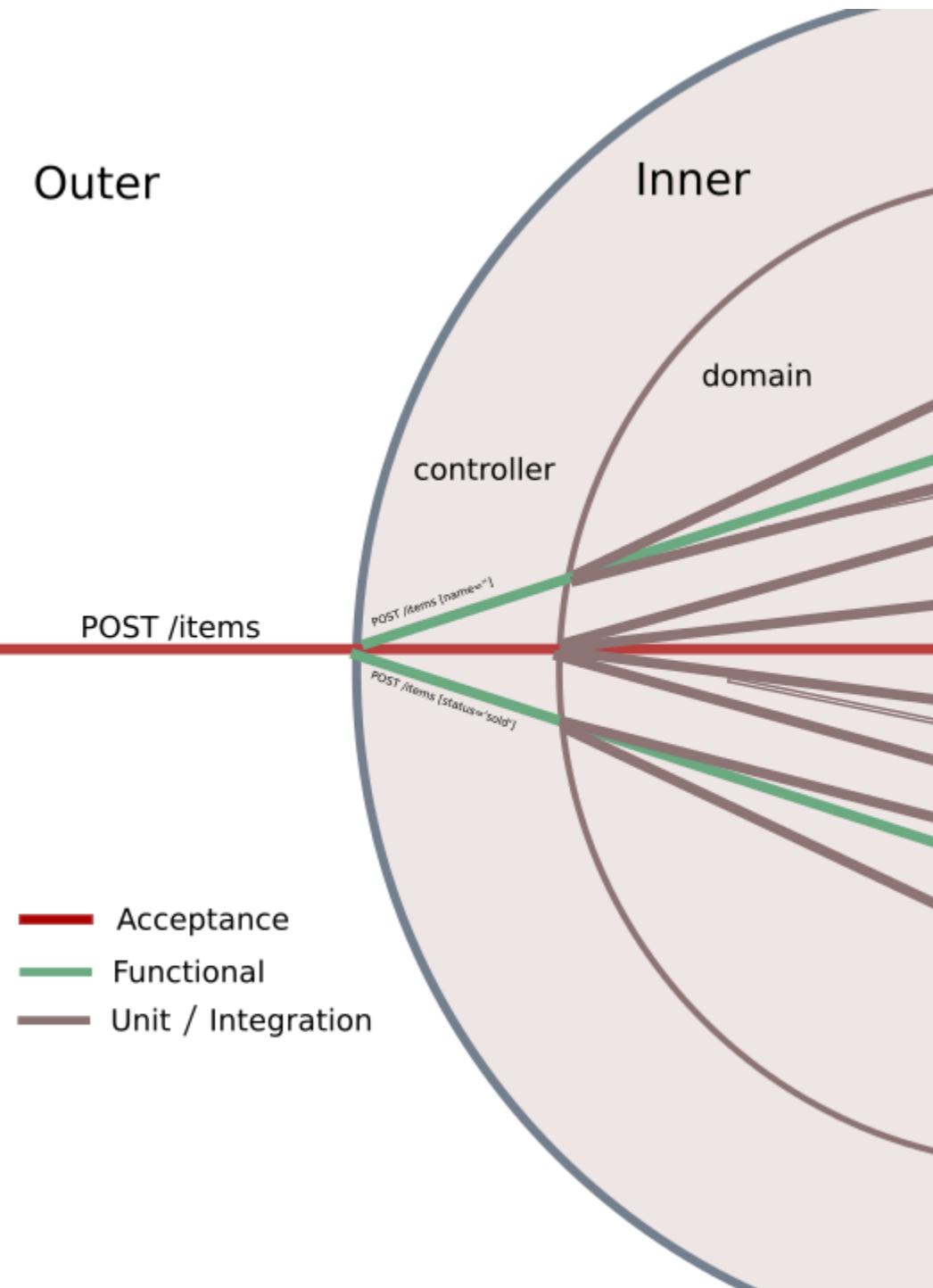


# **TEST TYPES**

- Outer
  - Acceptance: Browser-based UI tests
  - Characterization: CURL-based request/response
- Inner
  - Functional: Request/response emulation
  - Integration: Service with its dependencies
  - Unit: Service in pure isolation

		<h2>Know Pros and Cons</h2>	
<b>Stability to Changes</b>	<b>Wide Coverage</b>	<b>Invest into Infrastructure</b>	<b>Speed of Development</b>
	<pre>graph TD; UI[UI] --- Integration[Integration]; Integration --- Unit[Unit]</pre>		
<b>Stability of Execution</b>	<b>Detailed Coverage</b>	<b>Invest into Architecture</b>	<b>Speed of Execution</b>





# **HOW TO BUILD TEST ARCHITECTURE?**

# **WHAT TO TEST**

- Write down specifications
- Choose specifications which should be tested
- Write examples for specification
- Choose the testing layer

*The more specific example we need to test the more detailed layer we choose.*

## **ACCEPTANCE VS FUNCTIONAL VS UNIT**

- 1. Choose a testing layer where test would be**
  - Readable
  - Stable
  - Fast enough
- 2. Write a test**
- 3. Repeat**
- 4. Refactor!**

# UNIT VS INTEGRATION TESTS

- Unit Tests for
  - pure functions
  - algorithms
  - complex data
  - dozen execution paths
- Integration tests for
  - everything else

# MOCKS

- Are dangerous:
  - Affect readability
  - Affect stability
- Should be used for
  - Async services
  - 3rd-party services
  - Remote services

*Even you can write a unit test with mocks it doesn't mean  
you should*

## **TDD || !TDD**

- Hard to start (nothing is stable)
- Build on top of interfaces
- Use TDD to discover specifications

## BDD || !BDD

- Writing tests in English is not about BDD at all
- BDD transforms specification to tests
- BDD has its cost (additional abstraction layer)
- Use BDD when non-technical mates involved
  - (*when management is actually going to read your tests*)

# **TEST ARCHITECTURE TEMPLATES**

## **NEW PROJECT. HOW TO TEST?**

- Domain Layer should have unit / integration tests
- Application layer should have integration / functional tests
- UI should have acceptance tests with positive scenarios

# EARLY STAGES STARTUP. HOW TO TEST?

- Uncertainty Problem:
  - We don't have strict requirements
  - We can do a pivot any day
  - We are unsure of EVERYTHING 😠
- Solution:
  - Test only when you stabilize the code
  - Start with Public API, Domain Logic

# **LEGACY PROJECT. HOW TO TEST?**

- Detect the critical parts of a system
- Write acceptance tests for them
- Refactor old code
- Cover the new code with unit tests

# **CONCLUSIONS**

1. Discover what to test
2. Find a suitable level of testing
3. Write readable+stable+fast tests!

# QUESTIONS!

- Michael Bodnarchuk @davert
- Author of [Codeception](#) Testing Framework
- Consultant & Trainer at [SDCLabs](#)