A czy Ty masz zainstalowane wszystkie potrzebne aplikacje?

- PHP 7.1
- Composer
- IDE (np. Trial PHP Storm)

Na komputerach z windowsem może być XAMPP :)

WIFI: Sch-Guest
Hasło: lublub23


PHP TRAINING
Enhance your PHP code with PHPSpec & Behat
powered by
SCHIBSTED
TECH POLSKA

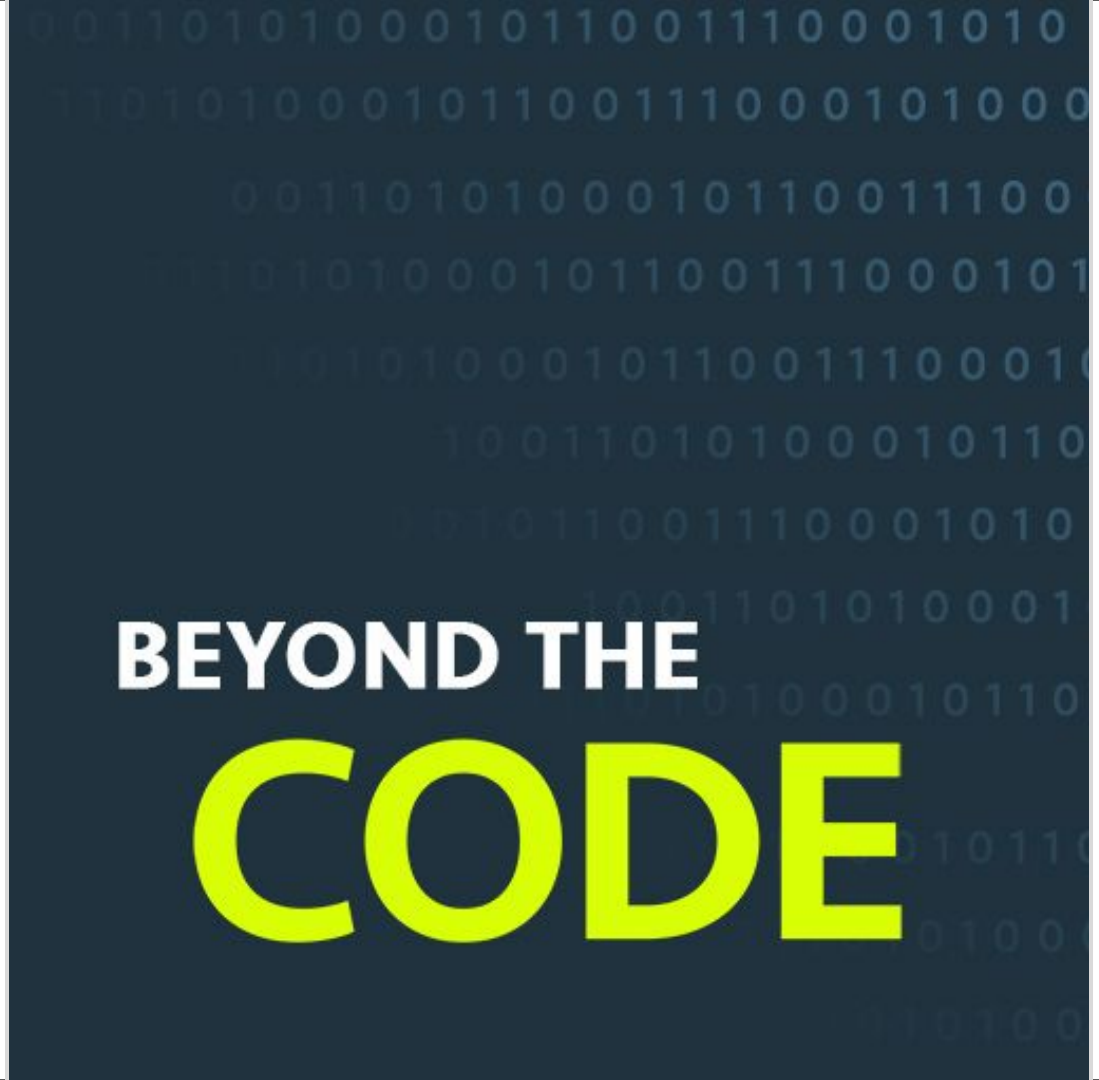
Kilka spraw organizacyjnych

SCHIBSTED
TECH POLSKA

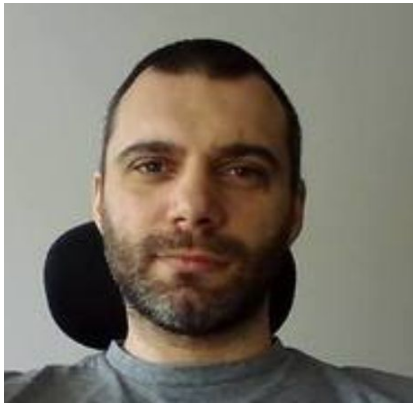
PHP Developer

Gdańsk

Apply now

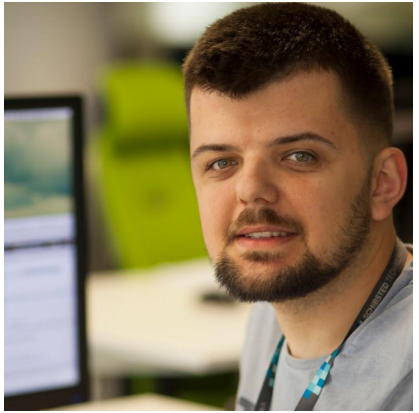

BEYOND THE
CODE

My !



Robert Burczak
Behat
3.

Maciej Kosiedowski
PhpSpec
2.



Kacper Sieradziński
Testy i takietam
1.

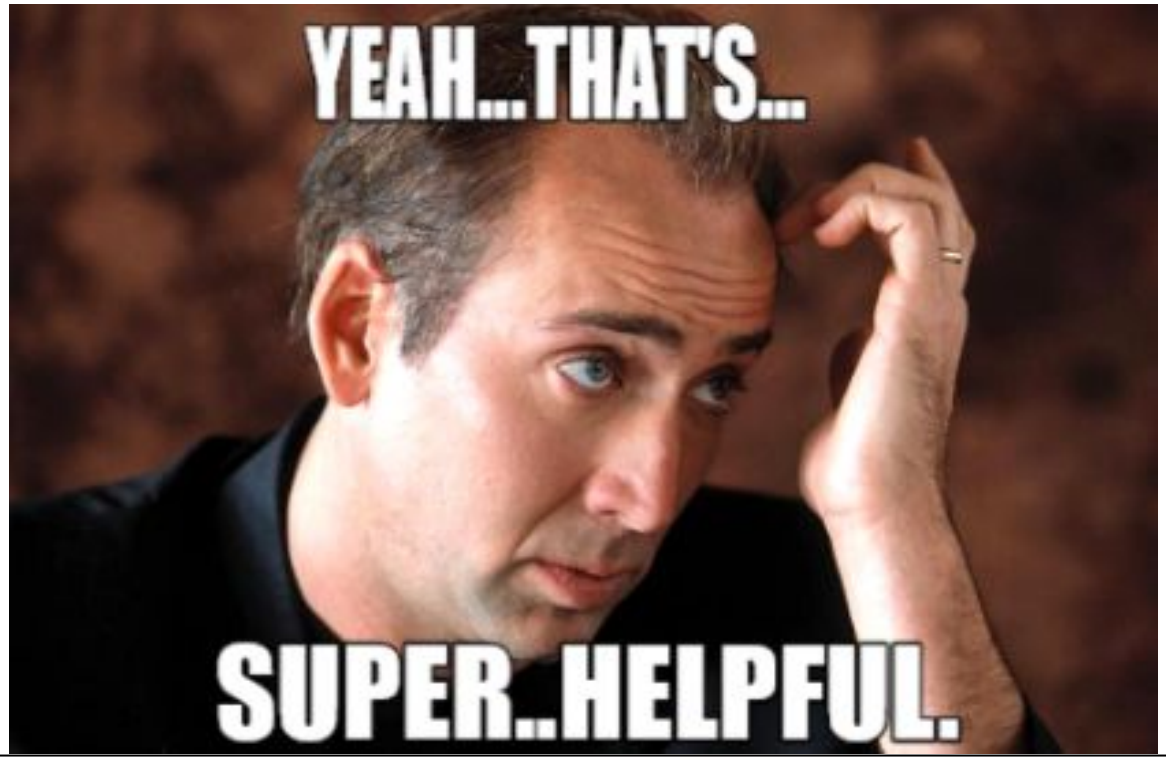
Kilka rad by spać spokojnie (j)

- Narzędzia ułatwiające prace
- Testy
- Continuous Integration

Narzędzia ułatwiające prace

- Phing - “ant” dla PHP :)
- Phpcs - php_codesniffer -
- PHPCPD - Copy/Paste Detector
- Phan - Statyczna analiza kodu

Dlaczego warto pisać testy?



Dlaczego warto pisać testy?

- Staram się zapewnić bezbłędność,
- Ułatwiam sobie modyfikacje aplikacji, unikając regresji,
- Próbuje spełnić oczekiwania klienta :)


WYMIEŃ TRZY TYPY TESTÓW


TVP 2 HD
PREMIERA
1
2
3
4
SUMA 0

Typy testów

- Testy strukturalne - “co”
- Testy funkcjonalne (wymagania specyfikacji - “jak”)
- Testy niefunkcjonalne (wydajność)

Poziomy testów

- Testy jednostkowe
- Testy integracyjne
- Testy funkcjonalne / end-to-end

Testy jednostkowe

- Najwincyj !
- Wykonują się szybko,
- Wyizolowane środowisko,
- Testujemy nimi pojedyncze metody,

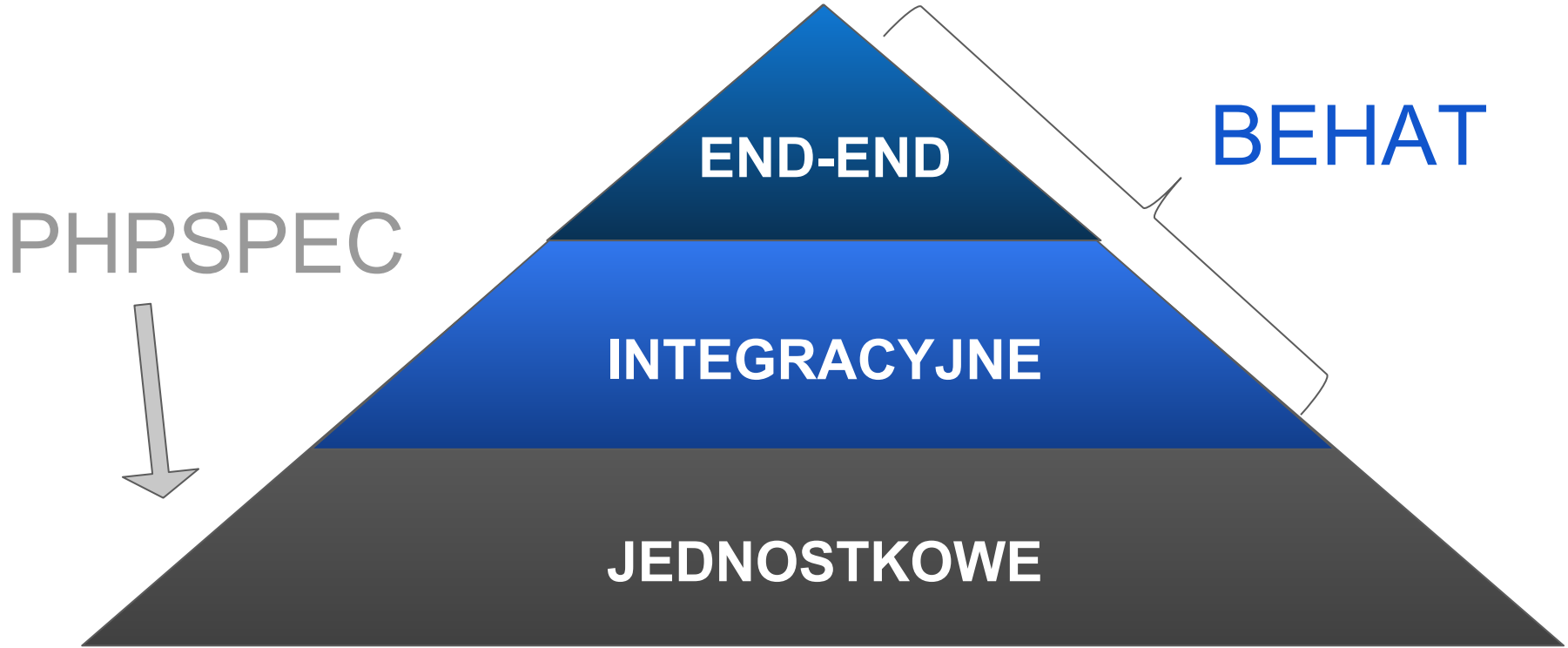
Testy integracyjne

- Integracja pomiędzy komponentami
- Środowisko nieprodukcyjne

Testy funkcjonalne/ end-end

- Ich głównym zadaniem jest sprawdzenie czy wszystko działa. Od początku do końca.
- Środowisko nieprodukcyjne lub produkcyjne - zależnie od projektu.

Poziomy testów



Jak pisać testowalny kod?

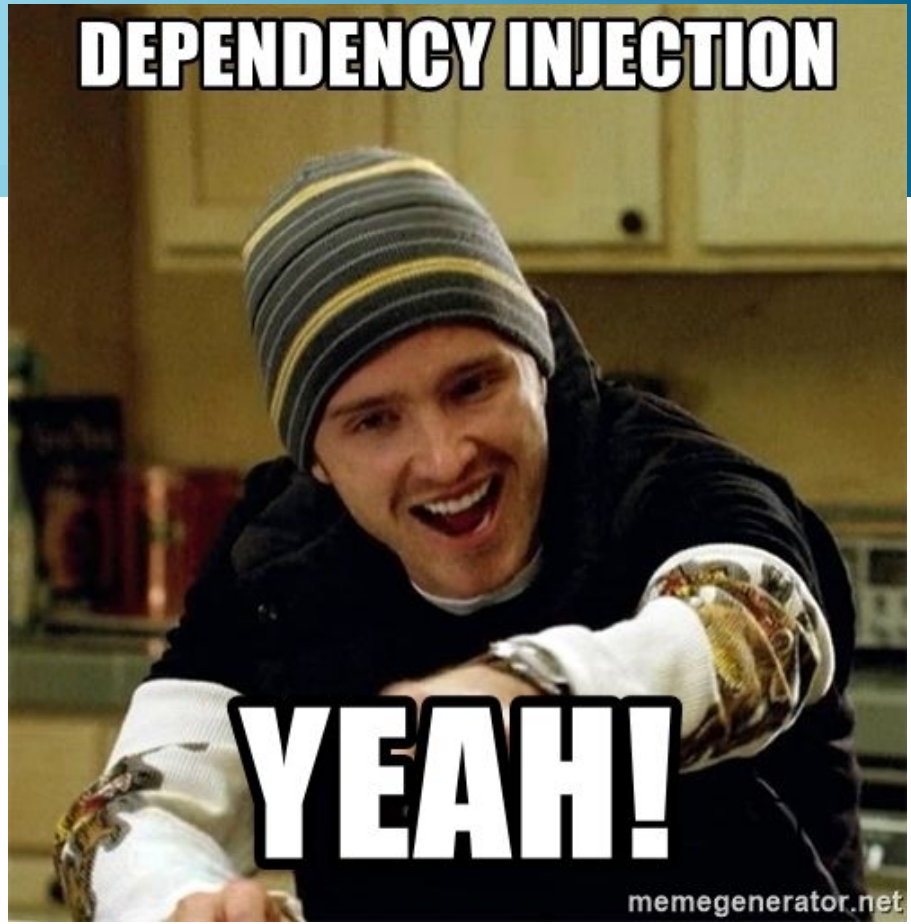
+

SOLID,

Zacznij od testów?

UNIKAJ

- Minimalizuj liczbę miejsc w których tworzysz nowe obiekty.



S - Single responsibility principle

JEDNA ODPOWIEDZIALNOŚĆ, TYLKO JEDEN POWÓD MODYFIKACJI KLASY

Łatwiej testować jeden niewielki kawałek kodu na raz.

O - Open close principle

Otwarcie klasy na rozszerzenia, ale ZAMKNIĘCIE JEJ NA MODYFIKACJE

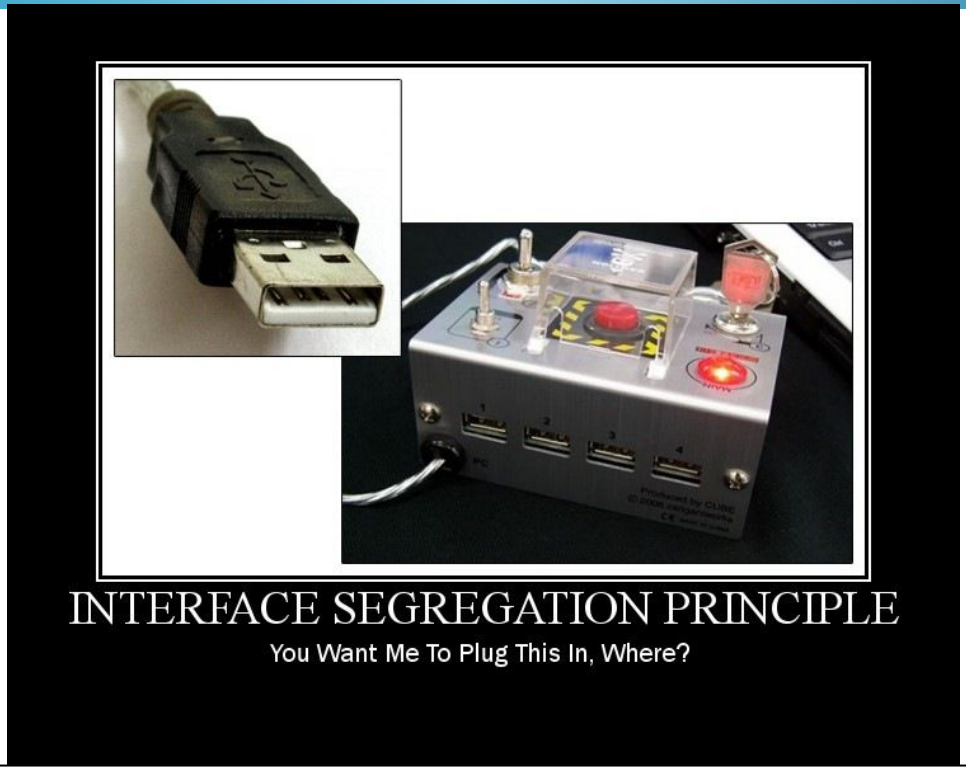
Jeśli Twój kod jest zamknięty na modyfikacje, to testy także będą się rzadko zmieniały.

L - Liskov substitution principle

Jeśli możesz użyć dowolnej implementacji, to możesz je wymienić na mocka.

I - Interface segregation principle

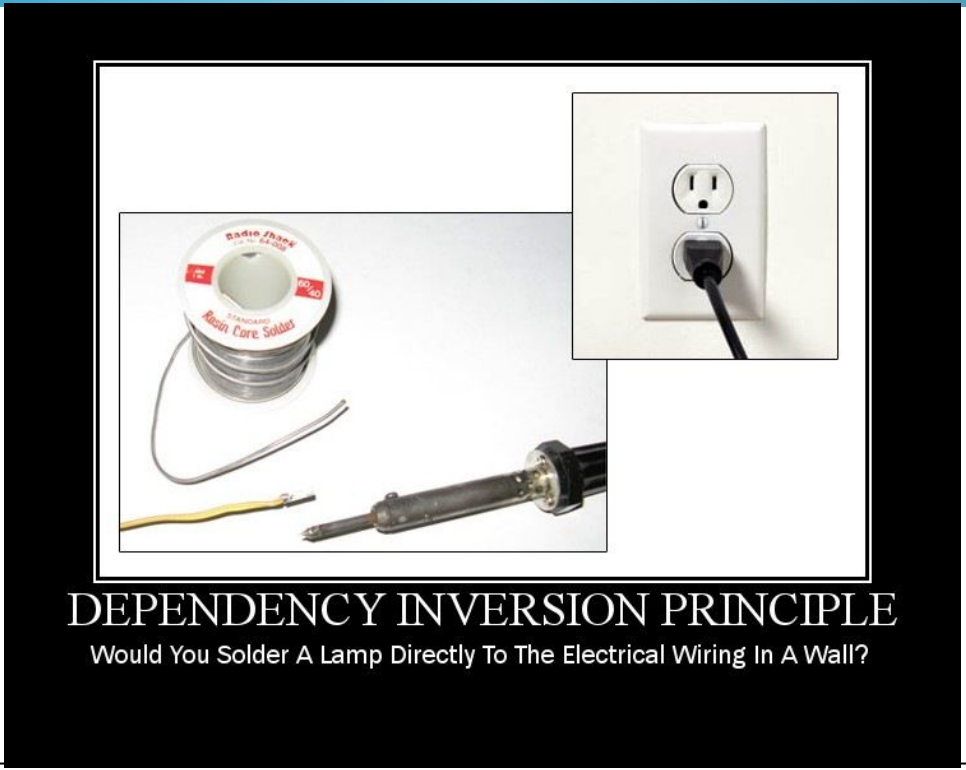
Wiele dedykowanych interfejsów jest lepsze niż jeden ogólny.



Małe i specyficzne interfejsy sprawią, że łatwiej będzie Ci wyspecyfikować co ma się stać.

D - Dependency Inversion principle

Wysokopoziomowe moduły nie powinny zależeć od modułów niskopoziomowych.



Jeśli moduł zależy od abstrakcji, a nie konkretnej implementacji, łatwo wstrzykniesz do niego mocki.

Continuous Integration

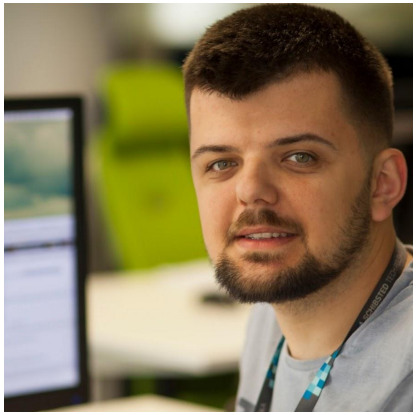
- sprawdza, czy “u mnie działa”.. To też na serwerze :)
- automatycznie uruchamiane testy i budowanie sie aplikacji.
- automatyczny deploy. Brak ryzyka, czy coś się nie wysypie podczas kopiowania plików.
- aktualizacja środowiska po każdym mergu
- itd..

Co dalej?

Maciej Kosiedowski

Robert Burczak

Zmiana!



phpspec

SCHIBSTED
TECH POLSKA

Maciej Kosiedowski

PHPSPEC - Agenda

- Wprowadzenie
- Struktura testów
- Mockowanie
- Różne rodzaje asercji
- Sprawdzanie pokrycia kodu
- Ćwiczenie

Co to jest phpspec?

- Zestaw narzędzi do pisania testów poprzez opis zachowania obiektów (SpecBDD)
- Główna różnica między phpspec a xUnit: **język**
- Testy piszemy nie „językiem testów”, lecz „językiem zachowań”
- Dzięki temu nasze myśli zostają bliżej domeny biznesowej

Instalacja i uruchomienie

- `composer require --dev phpspec/phpspec`
- `composer install`
- `./vendor/bin/phpspec`

Struktura testów

- Klasy testu piszemy z perspektywy testowanego obiektu (`$this` to nasz obiekt):
  - namespace z prefiksem `spec\`
  - nazwa zakończona sufiksem `Spec`
  - `extends ObjectBehaviour`
- Inicjalizacja/finalizacja funkcją:
  - `public function let(…) {}`
  - `public function letGo() {}`
- Testy w metodach:
  - `public function it_...(…) {}`

Struktura testów

Mockowanie

- Mock (atrapa) – obiekt który w kontrolowany sposób naśladuje zachowanie rzeczywistego obiektu
- Wystarczy zdefiniować parametry funkcji testowej a framework podstawi pod nie mocki w trakcie wywołania
- phpspec nie wspiera partial-mocków

Konfiguracja mocka

- `$mock->method("foo")->willReturn("bar")`
- `$mock->getFoo()->willThrow(new \RuntimeException("bar"))`
- `$mock->calculate(2, 4)->willReturnArgument(1)`

Konfiguracja mocka

- `$mock->getFoo(`**`$value`**`)->...`
- `$mock->getFoo(`**`new AnyValueToken()`**`)->...`
- `$mock->getFoo(`**`new CallbackToken(function () {...})`**`)->...`

Rodzaje asercji

- `$mock->should…(…)`
  - `$this->getValue()->shouldReturn(50);`
  - `$mock->isEqual($value)->shouldBe(true);`
  - `$db->save($object)->shouldBeCalled();`
- `$mock->shouldNot…(…)`
  - `$db->save($object)->shouldNotBeCalled()`
- `$mock->shouldThrow(…)->during(…)`

Rodzaje asercji

- `$mock->shouldTrigger(...)->during(...)`
- `$mock->getFoo()->shouldHaveType(...)`
- `$mock->shouldBeArray()`
- `$traversableMock->shouldContain(...)`
- `$mock->shouldDoBarrelRoll(5)`

```
public function getMatchers(): array {
  return [
    'doBarellRoll' => function(...) {...}
  ];
}
```

Pokrycie kodu

- Pokrycie kodu to miara która mówi jak duża część naszego kodu została uruchomiona w trakcie testów

```
composer require --dev leanphp/phpspec-code-coverage
```

- Dodatek w `phpspec.yml`

```
extensions:

LeanPHP\PhpSpec\CodeCoverage\CodeCoverageExtension: ~
```

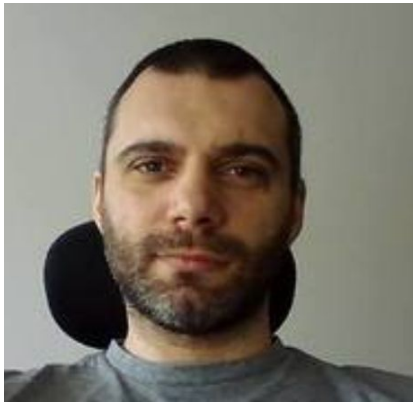
- Uwaga na Xdebug przy dużych projektach!

```
phpdbg -qrr phpspec run
```

Ćwiczenie

- Pobierz repozytorium
  https://github.com/mkosiedowski/php-testing-workshops
- Zainstaluj zależności
- Napisz testy dla metod klasy `\Domain\BasketManager`, w szczególności
  - Sprawdź czy metoda `getValue` korzysta z konwertera walut
  - Sprawdź czy metody rzucają wyjątkami gdy koszyk nie jest ustawiony
- Testy uruchom poleceniem `./bin/phing phpspec`
- Sprawdź raport pokrycia kodu

Zmiana!



behat

SCHIBSTED
TECH POLSKA

Robert Burczak

Czym jest Behat

- framework PHP (open source)
- BDD (Behaviour-Driven Development)
- Testy oparte na scenariuszach/historyjkach (StoryBDD)
- Gherkin language - lepsza komunikacja między deweloperami a biznesem
- rozszerzenia

Gherkin

- zrozumiały przez biznes i deweloperów
- dodatkowo otrzymujemy dokumentację funkcjonalną
- może być pisany w ponad 40 językach (w tym j. polski)
- parser jako osobna biblioteka (behat/gherkin)
- Cucumber (Java, JavaScript, Ruby, Kotlin)

Feature

- wyjaśnia wartość biznesową wynikającą z tej funkcjonalności
- opisuje kontekst
- nie jest parsowana, służy tylko do opisu

```
Feature:
  In order to…
  As an…
  I want to …
```

Scenario

- scenariusz tworzony przy użyciu kroków
- przekazywanie parametrów

- `Given` - doprowadzenie systemu do odpowiedniego stanu, zanim użytkownik wejdzie z nim w interakcję
- `When` - wykonanie akcji na systemie (zmiana stanu)
- `Then` - sprawdzanie danych wyjściowych systemu (tylko dane dostępne dla użytkownika/podsystemu)
- `And, But` - łączenie kroków, Behat interpretuje je dokładnie tak samo jak poprzednie kroki

Parametry

- Domyślnie - `:argument`
- Słowa alternatywne - `this/that`
- Wyrażenia regularne - `/^(\d+) items$/i`
- Tablice - `|name|price|currency|`
- Multiline Strings (Pystrings) -

```
"""
Quite a long text
In two lines
"""
```

Pluginy

- **MinkExtension** - możliwość testowania UI (Selenium)
- **Symfony2Extension** - integracja z frameworkiem Symfony2
- **Behat-Laravel-Extension** - integracja z frameworkiem Laravel
- **WebApiExtension** - ułatwia testowanie API (parsowanie JSON, nagłówki)
- **Inne** (Wordpress, Drupal, Zend Framework, SEO… i wiele więcej (291))

Instalacja

Najprościej korzystając z Composera:

```
php composer.phar require --dev behat/behat
```

Alternatywa:

Pobierając plik phar ze strony Behat.org i uruchomić go korzystając z PHP

Konfiguracja

- Cała konfiguracja w pliku behat.yml (YAML)
- Definicja profili (domyślny default), które służą do uruchomienia
- Konfiguracja zestawów testowych (suites):
  - filtry
  - ścieżki do plików
  - Kontekst
- Konfiguracja rozszerzeń

Uruchomienie

- Tworzymy szablon pliku kontekstu

```
bin/behat --init
```

- Tworzenie metod dla każdego z kroków (jeśli nie istnieją):

```
bin/behat --dry-run --append-snippets
```

- Uruchomienie testów:

```
bin/behat
```

Uruchomienie - opcje

- Domyślnie uruchamiamy wszystkie zestawy testów
- Wskazujemy zestaw testów:

```
bin/behat --suite=web_features
```

- Wskazujemy plik z funkcjonalnością:

```
bin/behat features/price.feature
```

- Wskazujemy scenariusz:

`bin/behat features/price.feature:10` - wskazujemy linię kodu w której znajduje się scenariusz

Ćwiczenie 1

Napisz test do następującego scenariusza:

```
Scenario: Add item to the basket

 Given I have empty basket

 When I add item to this basket

 Then Basket should contain one item
```

Ćwiczenie 2

Napisz test do scenariusza z możliwością podania parametrów:

```
Scenario: Add several items to the basket

 Given I have empty basket

 When I add item with name "first" and price "20 PLN"

 And I add item with name "second" and price "50 PLN"

 Then Basket should contain "2" items

 And Sum of all items in basket should be "70 PLN"
```

Ćwiczenie 3

Napisz test, który sprawdza poprawność usuwania przedmiotu z koszyka.

Ćwiczenie 4

Napisz test, który sprawdza wyliczenie sumy koszyka, który zawiera przedmioty w różnych walutach (konwersja walut).

```
Scenario: Convert item prices with different currencies

 Given I have empty basket

 When I add item with name "first" and price "20 EUR"

 And I add item with name "second" and price "50 CHF"

 Then Sum of all items in "PLN" currency should be correct
```

Dzięki!