We make IT

JCommerce

Testowanie aplikacji AngularJS z użyciem Karma i Jasmine

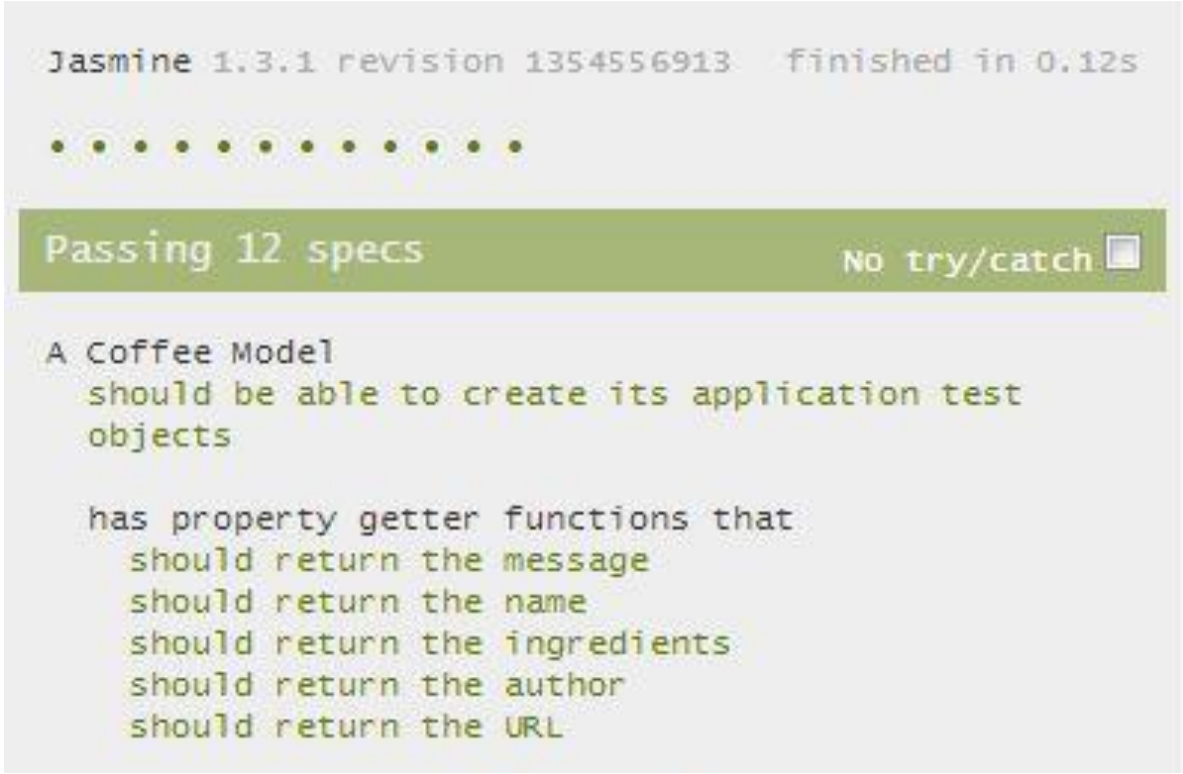
Katowice, 18 grudnia 2015r.

- Testy automatyczne
  - ✓ Korzyści z pisania testów
  - ✓ Rodzaje testów
  - ✓ Co testujemy, a czego nie
  - ✓ Inne metody dbania o wysoką jakość kodu
- Testowanie aplikacji AngularJS z Jasmine
  - ✓ Struktura testu
  - ✓ Mockowanie danych
  - ✓ Testowanie kontrolerów, dyrektyw i usług
- Część warsztatowa

Testy automatyczne

Czym są testy automatyczne

- Kod, który weryfikuje czy inny kod zachowuje się jak dobry kod.
- Integralna część developemntu, świadcząca o naszym profesjonaliźmie
- Jest to dowód, a nie wrażenie, że nasz kod działa poprawnie



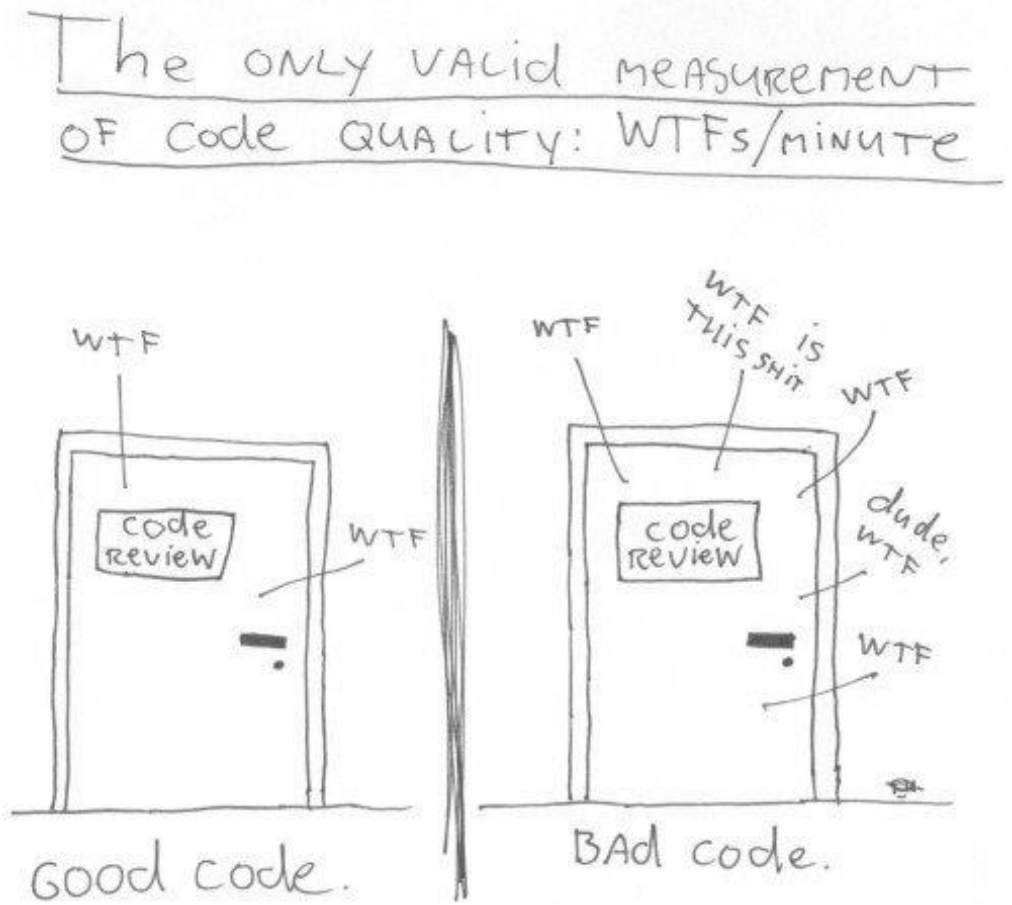
We make IT JCommerce

„Dobry kod nie wymaga testów”

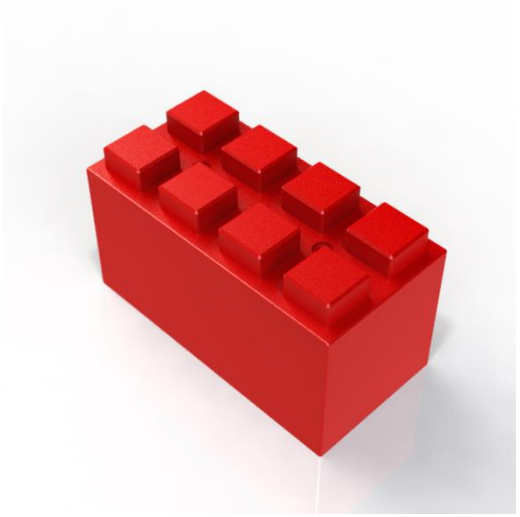
Senior Project Manager

Korzyści z pisania testów automatycznych

- Lepszej jakości kod:
  - minimalistyczne interfejsy
  - lepsza struktura usług / komponentów
  - analiza przypadków brzegowych
- Szybka weryfikacja poprawności kodu
- Ułatwia refaktoring
- Zdrowszy, spokojniejszy sen



We make IT JCommerce



Testy jednostkowe

skupiają się nad konkretnymi klasami / metodami
krótki czas wykonania – szybki feedback

Testy integracyjne

skupiają się nad iterakcją z interfejsem lub modułami
dłuższy czas wykonania

We make IT JCommerce

Testy jednostkowe: co testujemy

- każdą metodę publiczną
- przypadki brzegowe, które można zidentyfikować m.in. po występujących IF'ach

„Zróbcie to ASAP!!! To nie jest już czas na pisanie testów!”

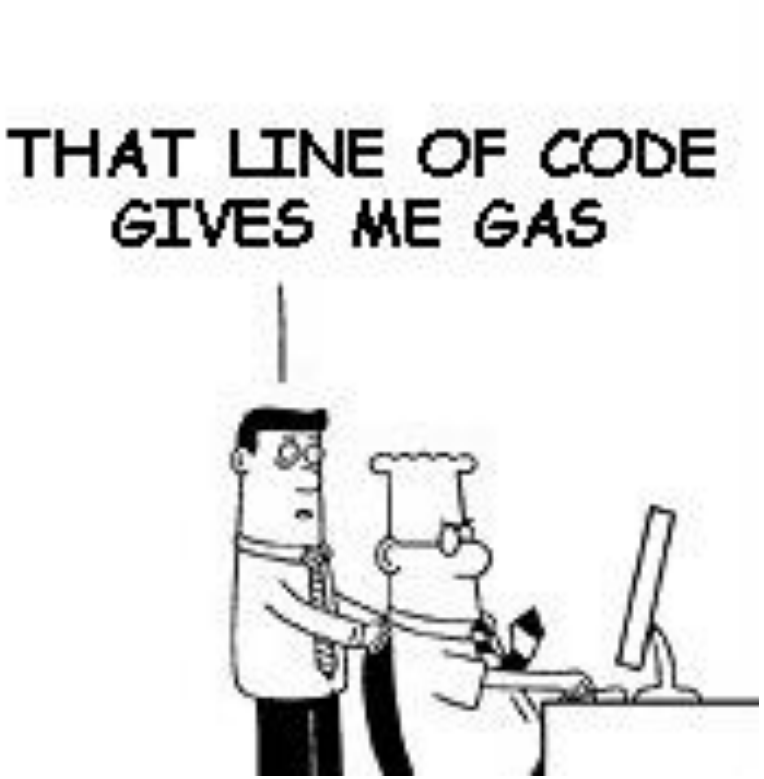
Senior Project Manager

Testy jednostkowe: czego nie testujemy

- metod prywatnych
- metod z zewnętrznych bibliotek
- metod, które zostały już przetestowane w innym miejscu

Inne metody utrzymywania wysokiej jakości kodu

- korzystanie z lintów (statyczna analiza kodu),
- śledzenie wskaźników - np. test code coverage,
- wspólne planowanie większych features
- code reviews



Testowanie aplikacji AngularJS z Jasmine

Stuktura testu

- `describe`
  wskazywanie nazwy testowanej metody lub warunków testu
- `beforeEach`
  przygotowanie warunków testowych dla grupy testów
- `it`
  definiowanie spodziewanego zachowania lub wartości

```
describe('commentsLabel', function () {
  beforeEach(function(){
    model = {};
  });

  describe('when comments count is 0', function () {
    it('should return label for no comments"', function () {
      model.comments_count = 0;
      expect(model.commentsLabel()).toEqual('no comments');
    });
  });

  describe('when comments count is 1', function () {
    it('should return label for one comment', function () {
      model.comments_count = 1;
      expect(model.commentsLabel()).toEqual('1 comment');
     });
   });
});
```

We make IT JCommerce

Weryfikacja

Korzystając z Jasmine mamy do dyspozycji wiele matcherów, które sprawdzają testowe oczekiwania.

```
cat = {name: Bonifancy, age: 10, isCute: true};

expect(cat.name).toContain(”fancy”);
expect(cat.age).toBe(10);
expect(cat.isCute).toBeTruthy();
```

Wstrzykiwanie usług

Usługę wstrzykujemy, gdy chcemy np. ją przetestować lub zamockować wykonanie jej metod

- Dozwolona jest notacja z podkreślnikami, gdy chcemy przypisać do czytelnej zmiennej
- inject może być użyty tylko w blockach beforeEach oraz it.

```
var Kitten;

beforeEach(inject(function (_Kitten_) {
  Kitten = _Kitten_;
}));
```

Mockowanie metod z użyciem spyOn

Mockowanie metod pozwala na wytworzenie specyficznych warunków testowych poprzez:

- Narzucenie rezultatu wykonania danej metody
- Śledzenia wykonania danej metody

```
var kitten;

beforeEach(inject(function (Kitten) {
  kitten = Kitten.new({cute: false});
  spyOn(kitten, isCute).and.returnValue(true)
}));

it(should be cute, function(){
  expect(kitten.isCute()).toBeTruthy();
});

it(should call cute method, function(){
  kitten.isCute()
  expect(kitten.isCute).toHaveBeenCalled();
});
```

Testowanie kontrolerów

```
yo angular:controller name
```

- Testowanie zapytań AJAX przy pomocy $httpBackend
- Mockowanie promises z użyciem $q

```
describe('fetching collection', function () {
    it('should assign fetched collection', inject(function ($httpBackend) {
      $httpBackend.expect('GET', '/kittens').respond([{}]);
      $httpBackend.flush();
      expect(KittensIndexCtrl.collection.length).toBe(1);
    }));
  });
```

Testowanie kontrolerów

```
yo angular:controller name
```

- Testowanie zapytań AJAX przy pomocy $httpBackend
- Mockowanie promises z użyciem $q

```
describe('fetching collection', function () {
    it('should assign fetched collection', inject(function ($q, Kitten) {
      var promise = $q.resolve([{id: 1, name: 'Kitty!'}]);
      spyOn(Kitten, 'getList').and.returnValue(promise);
      expect(KittensIndexCtrl.collection.length).toBe(1); }));
  });
```

Testowanie dyrektyw

```
yo angular:directive name
```

- karma-ng-html2js-preprocessor wykorzystywany jest przy zewnętrznych szablonach HTML
- Kod dyrektyw powinien być w miarę możliwości wyciągany do osobnego kontrolera i tam testowany. Natomiast w dyrektywie testujemy np. Obsługę eventów na DOM lub stan wyrenderowanych elementów

```
beforeEach(module('kittensApp.templates'));
  var element, scope;

  beforeEach(inject(function ($rootScope, $compile) {
    scope = $rootScope.$new();
    scope.resource = {title: 'Bonifancy'};
    var markup = '<kitten-row ' +
                 'resource="resource"></kitten-row>';
    element = angular.element(markup);
    element = $compile(element)(scope);
    scope.$digest();
  }));

  describe('rendered directive', function () {
    it('should contain title', function () {
      expect(element.text()).toContain('Bonifancy');
    });
  });
```

Testowanie dyrektyw

- Jeśli koniecznie chcemy dostać się do wyizolowanego scope'u elementu, możemy skorzystać z metody `isolateScope()`

```
describe('with isolated scope', function () {
  it('should have title', function () {
    expect(element.isolateScope().resource.title).toEqual('Bonifancy');
  });
});
```

Testowanie usług

- Jeśli Testowanie usług wygląda bardzo podobnie do testowania kontrolerów. Często powtarzające się testy możemy wyciągnąć do osobnej metody.

```
describe('Service: Kitten', function () {
  # ...
  var Kitten;
  beforeEach(inject(function (_Kitten_) {
    Kitten = _Kitten_;
  }));

  window.expectHavingNewMethod('Kitten');

  describe('as Restangular service', function () {
    it('should have getList defined', function () {
      expect(Kitten.getList).toBeDefined();
    });
  });
});
```

Testowanie usług

- Jeśli Testowanie usług wygląda bardzo podobnie do testowania kontrolerów. Często powtarzające się testy możemy wyciągnąć do osobnej metody.

```
window.expectHavingNewMethod = function(serviceName) {
  var service,
    model;

  describe('new', function () {
    beforeEach(inject([serviceName, function (_service_) {
      service = _service_;
      model = service.new();
    }]));
    it('should return restangularized element', function () {
      expect(model.restangularized).toBeTruthy();
    });

    it('should have correct route', function () {
      expect(model.route).toEqual(service.route);
    });
  });
}
```

Work in progress

W trakcie pisania testów możemy skorzystać z kilku dodatkowych metod

- `pending()` powoduje pominięcie wykonania testu
- `fit, fdescribe` powoduje wykonanie tylko wskazanego testu
- `jasmine.any(...)` pozwala matchować w teście różne typy danych np. przy sprawdzeniu wykonania metody

```
it('should do sth', function(){
  pending();
});

fit('should return true', function(){
  expect(kitten.isCute()).toBeTruthy();
});

it('should return true', function(){
  spyOn(kitten, 'rub').and.callThrough();
  kitten.rub('belly');
  expect(kitten.rub).
    toHaveBeenCalledWith(jasmine.any(String));
});
```

Część warsztatowa

Część warsztatowa

- Pobieramy KittenApp z: github.com/tbprojects/kittensApp
- Setup aplikacji zgodnie z informacjami zawartymi w readme.md
- Wykonanie zadań zawartymi w sekcji Workshop exercises w readme.md

Pytania

We make IT JCommerce

We make IT JCommerce

I ty możesz zostać niezależnym developerem!

Tomasz Borowski

FrontEnd Developer

tomasz.borowski@jcommerce.pl