UNIT TESTING

Ana Rita Silva, Manuel Zamith TVVS

Friday, October 7, 2016

What is this?

Unit Testing Techniques

Black Box Testing - Testing a system without having specific knowledge of the internal workings of the system, no access to the source code, and no knowledge of the architecture.

White/Clear Box Testing - Testing a system with full knowledge and access to all source code and architecture documents.

Grey Box Testing - Testing a system while having at least some knowledge of the internals of a system.

Why is this used?

- Tests one specific behavior
- Improves efficiency and reliability
- Can act as documentation of the system - the unit test itself is used to verify the design
- You can be more confident about your code
- Each test is autonomous

Benefits

Limitations

- Time consuming
- Does not show absence of errors
- Integration errors or system errors can be missed
- Test code can have more bugs than the code it is testing
- Value and accuracy of unit tests can be diminished if initial conditions are not set correctly. There are no tests for the tests, you don't know
- Does not help to define a product or a requirement - can lead to a unit test checking for the wrong thing, and writing code that passes the test

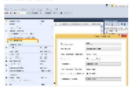
Unit Testing Tools

C/C++

PHP

Javascript

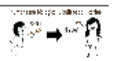
.NET & others

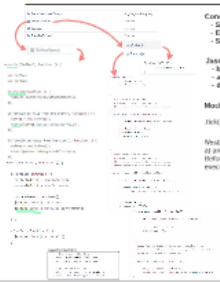


Code Coverage Tools for Javascript

- **JSCover** - free and coverage measured via a browser
- **BLANKET.JS** - deprecated for jasmine version
- **jasmine istanbul** - embedded in KARMA
- **KARMA** - spawns a web server that executes source code against test code for each of the browsers connected

Using Jasmine & KARMA





Concepts:
- **Specs** - function/test that contains 1 or + expectations
- **Expectation** - assertion that something is T/F
- **String** - title of the spec

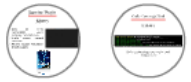
Jasmine Setup and Teardown:
- **beforeEach** - function called once before each spec
- **afterEach** - function called once after each spec
- **describe** - which function is called

Mocks & stubs

jasmine.clock()

tick(milliSecondsToTick); | install(); | uninstall() | mockDate();

Nesting - Calls to **describe** can be nested, with specs defined at any level - tree of functions corresponding.
Before a spec is executed, Jasmine walks down the tree executing each **beforeEach** function in order.



Exercise

Can you find the errors in source code?
Achieve 100% coverage and add new test case(s).

Follow the instructions provided in:
https://github.com/krampstudio/jasmine-test-users

References

> "Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it."
> Alan J. Perlis

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

UNIT TESTING

Ana Rita Silva, Manuel Zamith TVVS

Friday, October 7, 2016

What is this?

Unit Testing Techniques

Black Box Testing - Testing a system without having specific knowledge of the internal workings of the system, no access to the source code, and no knowledge of the architecture.

White/Clear Box Testing - Testing a system with full knowledge and access to all source code and architecture documents.

Grey Box Testing - Testing a system while having at least some knowledge of the internals of a system.

Why is this used?

- Tests one specific behavior
- Improves efficiency and reliability
- Can act as documentation of the system - the unit test itself is used to verify the design
- You can be more confident about your code
- Each test is autonomous

Benefits

Limitations

- Time consuming
- Does not show absence of errors
- Integration errors or system errors can be missed
- Test code can have more bugs than the code it is testing
- Value and accuracy of unit tests can be diminished if initial conditions are not set correctly. If there are no tests for the values, you don't know
- Does not help to define a product or a requirement - can lead to a unit test checking for the wrong thing, and writing code that passes the test

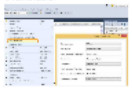
Unit Testing Tools

C/C++

PHP

Javascript

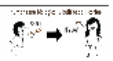
.NET & others

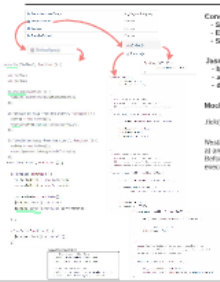


Code Coverage Tools for Javascript

- **JSCover** - free and coverage measured via a browser
- **BLANKET.JS** - deprecated for jasmine version
- **jasmine istanbul** - embedded in KARMA
- **KARMA** - spawns a web server that executes source code against test code for each of the browsers connected

Using Jasmine & KARMA





Concepts:
- **Specs** - function/test that contains 1 or + expectations
- **Expectation** - assertion that something is T/F
- **String** - title of the spec

Jasmine Setup and Teardown:
- **beforeEach** - function called once before each spec
- **afterEach** - function called once after each spec
- **describe** - which function is called

Mocks & stubs

jasmine.clock()

tick(milliSecondsToTick); install(); uninstall(); mockDate();

Nesting - Calls to **describe** can be nested, with specs defined at any level - tree of functions and specs.
Before a spec is executed, Jasmine walks down the tree executing each **beforeEach** function in order.

Exercise

Can you find the errors in source code?
Achieve 100% coverage and add new test case(s).

Follow the instructions provided in:
https://github.com/tmcnab/jasmine-test-users

References

> Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.
> Alan J. Perlis

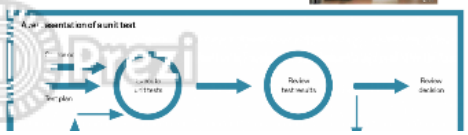
U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

UNIT TESTING

Ana Rita Silva, Manuel Zamith TVVS

Friday, October 7, 2016

What is this?



Why is this used?

- Tests one specific behavior
- Improves efficiency and reliability
- Can act as documentation of the system - the unit test itself is used to verify the design
- You can be more confident about your code
- Each test is autonomous

Benefits

- Improves efficiency and reliability
- Reduces defects in newly developed features or reduces bugs when changing the existing functionality
- Reduces cost of testing - defects are detected in an

Unit Testing Tools

C/C++

VECTOR software

BOOST TEST

PHP

PHPUnit

Javascript

Jasmine

JUnit 5

KARMA

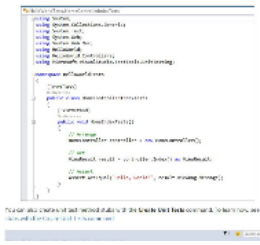
chai

mocha

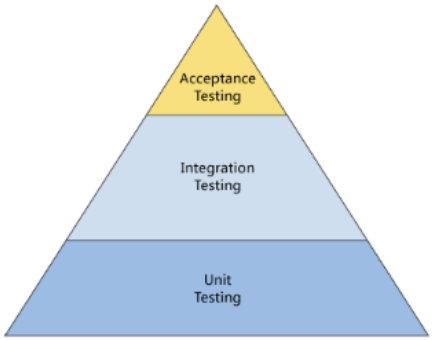
yui

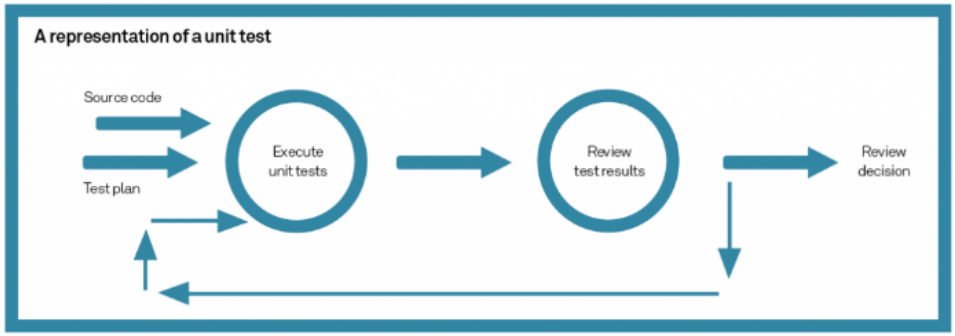
.NET & others

Microsoft Visual Studio Unit Testing Framework



What is this?





Unit Testing Techniques

Black Box Testing - Testing a system without having specific knowledge to the internal workings of the system, no access to the source code, and no knowledge of the architecture.

White/Clear Box Testing - Testing a system with full knowledge and access to all source code and architecture documents.

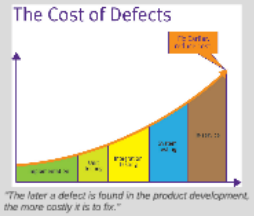
Grey Box Testing - Testing a system while having at least some knowledge of the internals of a system.

Why is this used?

- Tests one specific behavior
- Improves efficiency and reliability
- Can act as documentation of the system - the unit test itself is used to verify the design
- You can be more confident about your code
- Each test is autonomous

Benefits

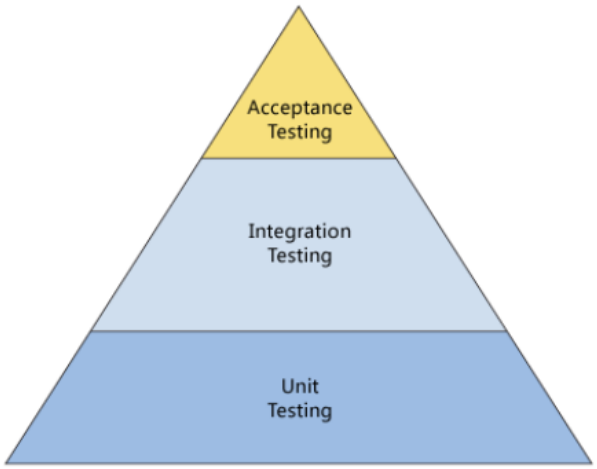
- Improves efficiency and reliability
- Reduces defects in newly developed features or reduces bugs when changing the existing functionality
- Reduces cost of testing - defects are detected in an early phase
- Improves design and allows better refactoring of code
- Integration testing is easier

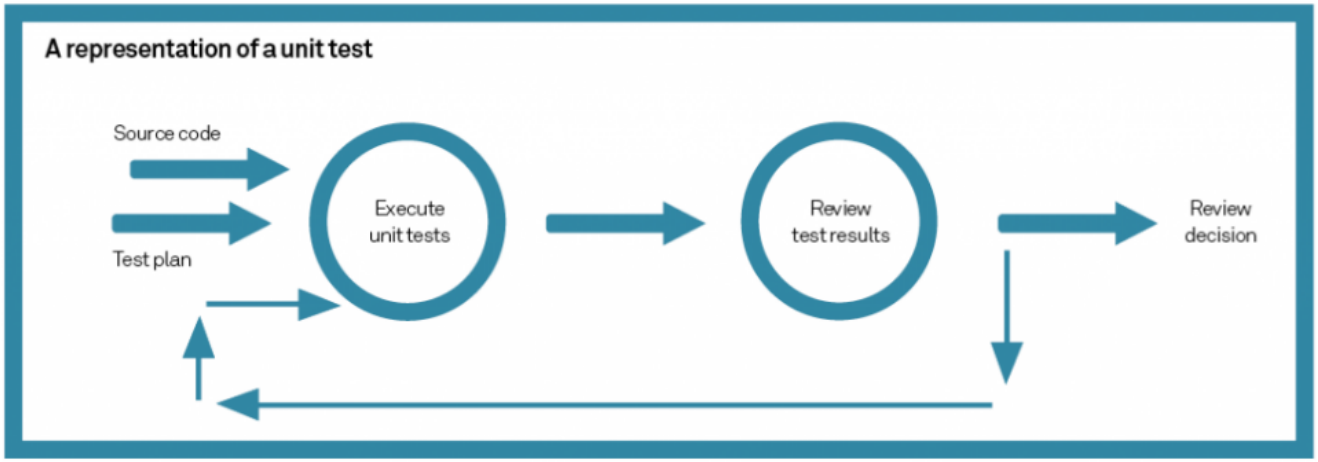


Limitations

- Time consuming
- Does not show absence of errors
- Integration errors or system errors can be missed
- Test code can have more bugs than the code it is testing
- Value and accuracy of unit tests can be diminished if initial conditions are not set correctly - there are no tests for the cases you don't have
- Does not help to better understand a requirement - can lead to a unit test checking for the wrong thing, and wrong code that passes the test

What is this?





Unit Testing Techniques

Black Box Testing - Testing a system without having specific knowledge to the internal workings of the system, no access to the

Why

- Tests
- Impr
- Can itself
- You
- Each

Ben

- Improve
- Reduces reduces function
- Reduces early ph
- Improve code
- Integrati

Lim

Prezi

Unit Testing Techniques

Black Box Testing - Testing a system without having specific knowledge to the internal workings of the system, no access to the source code, and no knowledge of the architecture.

White/Clear Box Testing - Testing a system with full knowledge and access to all source code and architecture documents.

Grey Box Testing - Testing a system while having at least some knowledge of the internals of a system.

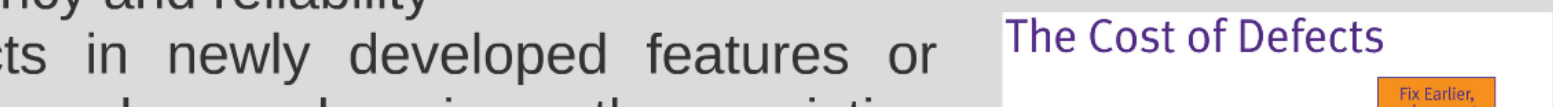
Why is this used?

- Tests one specific behavior
- Improves efficiency and reliability
- Can act as documentation of the system - the unit test itself is used to verify the design
- You can be more confident about your code
- Each test is autonomous

Benefits

Improves efficiency and reliability

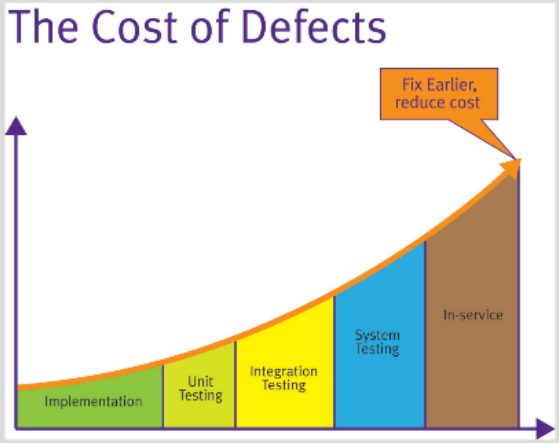
Reduces defects in newly developed features or



- Each test is autonomous

Benefits

- Improves efficiency and reliability
- Reduces defects in newly developed features or reduces bugs when changing the existing functionality
- Reduces cost of testing - defects are detected in an early phase
- Improves design and allows better refactoring of code
- Integration testing is easier



"The later a defect is found in the product development, the more costly it is to fix."

Limitations

- Time consuming
- Does not show absence of errors

code
- Integration testing is easier

"The later a defect is found in the product development, the more costly it is to fix."

Limitations

- Time consuming
- Does not show absence of errors
- Integration errors or system errors can be missed
- Test code can have more bugs than the code it is testing
- Value and accuracy of unit tests can be diminished if initial conditions are not set correctly - there are no tests for the cases you don't have
- Does not help to better understand a requirement - can lead to a unit test checking for the wrong thing, and wrong code that passes the test

Humans are biologically built to spill coffee

Unit Testing Tools

C/C++

VECTOR software

BOOST TEST

PHP

PHPUnit

Javascript

Jasmine

JUnit 5

KARMA

chai

mocha

yui

.NET & others

Microsoft Visual Studio Unit Testing Framework

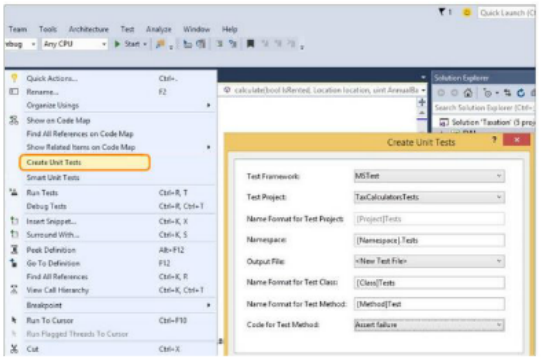
```
HelloWorldTests.HomeControllerIndexTests
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.Web.Mvc;
using HelloWorld;
using HelloWorld.Controllers;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace HelloWorldTests
{
    [TestClass]
    0 references
    public class HomeControllerIndexTests
    {
        [TestMethod]
        0 references
        public void HomeIndexTests()
        {
            // Arrange
            HomeController controller = new HomeController();

            // Act
            ViewResult result = controller.Index() as ViewResult;

            // Assert
            Assert.AreEqual("Hello, World!", result.ViewBag.Message);
        }
    }
}
```

You can also create unit test method stubs with the **Create Unit Tests** command. To learn how, see Create unit test method stubs with the Create Unit Tests command.

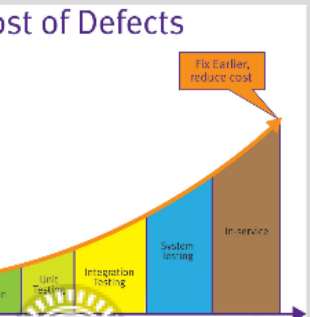


Code Coverage Tools for Javascript

- **JSCover -** free and coverage measured via a browser
- **BLANKET.JS** - deprecated for jasmine version
- **jasmine istanbul** - embedded in KARMA
- **KARMA** - spawns a web server that executes source code against test code for each of the browsers connected

Unit Testing Tool

he unit test



C/C++

VECTOR software

BOOST TEST

PHP

PHPUnit

J

JUnit 5

chai

Code Coverage Tools for Java

pt

ine

RMA

yui

rowser

source
s

.NET & others

Microsoft
Visual Studio
Unit Testing
Framework

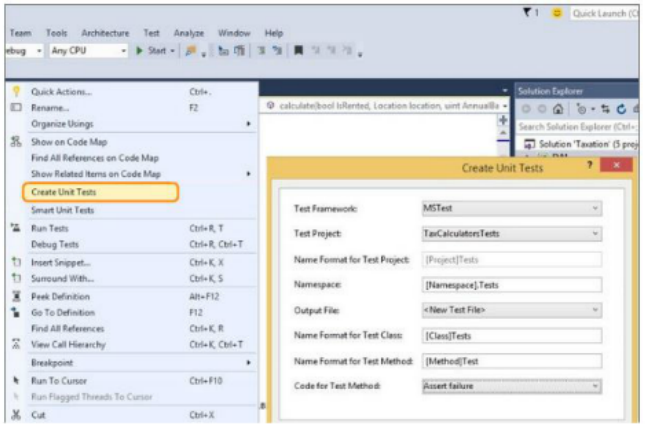
```
HelloWorldTests.HomeControllerIndexTests
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.Web.Mvc;
using HelloWorld;
using HelloWorld.Controllers;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace HelloWorldTests
{
    [TestClass]
    0 references
    public class HomeControllerIndexTests
    {
        [TestMethod]
        0 references
        public void HomeIndexTests()
        {
            // Arrange
            HomeController controller = new HomeController();

            // Act
            ViewResult result = controller.Index() as ViewResult;

            // Assert
            Assert.AreEqual("Hello, World!", result.ViewBag.Message);
        }
    }
}
```

You can also create unit test method stubs with the **Create Unit Tests** command. To learn how, see Create unit test method stubs with the Create Unit Tests command.



ting Tools

Javascript

.NET

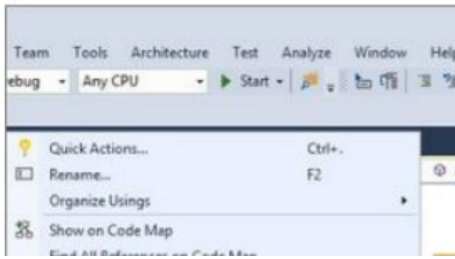
```
HelloWorldTests.HomeControllerIndexTests
using System;
using System.Collections.Generic
using System.Text;
using System.Web;
using System.Web.Mvc;
using HelloWorld;
using HelloWorld.Controllers;
using Microsoft.VisualStudio.Te

namespace HelloWorldTests
{
    [TestClass]
    0 references
    public class HomeController
    {
        [TestMethod]
        0 references
        public void HomeIndexTe
        {
            // Arrange
            HomeController cont

            // Act
            ViewResult result =

            // Assert
            Assert.AreEqual("He
        }
    }
}
```

You can also create unit test method stubs wi
stubs with the Create Unit Tests command.

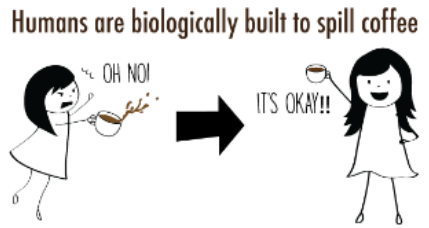


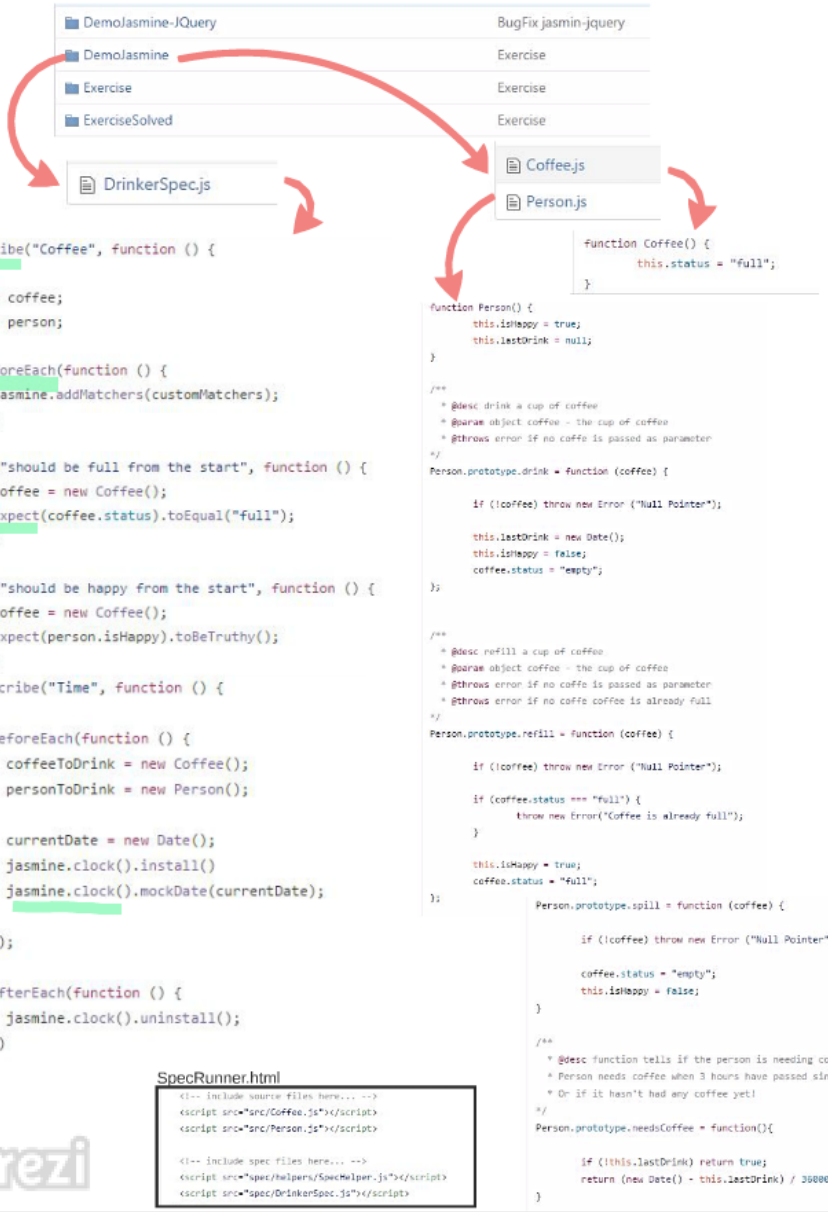
Tools for Javascript

Code Coverage Tools for Javascript

- **JSCover -** free and coverage measured via a browser
- **BLANKET.JS** - deprecated for jasmine version
- **jasmine istanbul** - embedded in KARMA
- **KARMA** - spawns a web server that executes source code against test code for each of the browsers connected

Using Jasmine & KARMA





Concepts:

- **Specs** - function/test that contains 1 or + expectations
- **Expectation** - assertion that something is T/F
- **String** - title of the spec

Jasmine Setup and Teardown:

- **beforeEach** - function called once before each spec
- **afterEach** - function called once after each spec
- **describe** - which function is called

Mocks & stubs

jasmine.clock()
.tick(*nºmilliseconds*); .install(); .uninstall() .mockDate();

Nesting - Calls to **describe** can be nested, with specs defined at any level - tree of functions composition.
Before a spec is executed, Jasmine walks down the tree executing each **beforeEach** function in order.

ARMA

Concepts:

- **Specs** - function/test that contains 1 or + expectations
- **Expectation** - assertion that something is T/F
- **String** - title of the spec

Jasmine Setup and Teardown:

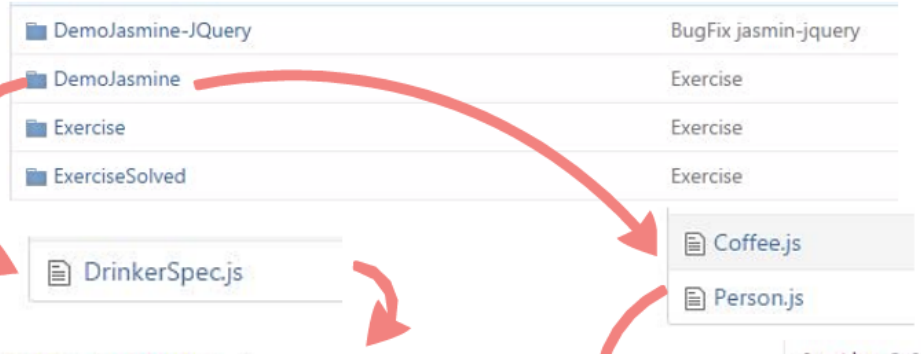
- **beforeEach** - function called once before each spec
- **afterEach** - function called once after each spec
- **describe** - which function is called

Mocks & stubs

jasmine.clock()

.tick(*nºmilliseconds*); .install(); .uninstall() .mockDate();

Using Jasmine & KARMA




```
describe("Coffee", function () {

  var coffee;
  var person;

  beforeEach(function () {
    jasmine.addMatchers(customMatchers);
  });

  it("should be full from the start", function () {
    coffee = new Coffee();
    expect(coffee.status).toEqual("full");
  });

  it("should be happy from the start", function () {
    coffee = new Coffee();
    expect(person.isHappy).toBeTruthy();
  });
describe("Time", function () {

    beforeEach(function () {
      coffeeToDrink = new Coffee();
      personToDrink = new Person();
```

```
function Coffee() {
        this.status = "full";
}
```

```
function Person() {
        this.isHappy = true;
        this.lastDrink = null;
}

/**
  * @desc drink a cup of coffee
  * @param object coffee - the cup of coffee
  * @throws error if no coffe is passed as parameter
*/
Person.prototype.drink = function (coffee) {

        if (!coffee) throw new Error ("Null Pointer");

        this.lastDrink = new Date();
        this.isHappy = false;
        coffee.status = "empty";
};


/**
  * @desc refill a cup of coffee
  * @param object coffee - the cup of coffee
  * @throws error if no coffe is passed as parameter
  * @throws error if no coffe coffee is already full
*/
Person.prototype.refill = function (coffee) {

        if (!coffee) throw new Error ("Null Pointer");

        if (coffee.status === "full") {
                throw new Error("Coffee is already full");
```

Concepts:

- **Specs** - function/test that con
- **Expectation** - assertion that
- **String** - title of the spec

Jasmine Setup and Teardown

- **beforeEach** - function called
- **afterEach** - function called o
- **describe** - which function is

Mocks & stubs

jasmine.c

.tick(*nºmilliseconds*); .install();

Nesting - Calls to **describe** can
at any level - tree of functions co
Before a spec is executed, Ja
executing each **beforeEach** func

- **Expectation** - assertion that something is T/F
- **String** - title of the spec

Jasmine Setup and Teardown:

- **beforeEach** - function called once before each spec
- **afterEach** - function called once after each spec
- **describe** - which function is called

Mocks & stubs

jasmine.clock()

.tick(*nºmilliseconds*); .install(); .uninstall() .mockDate();

Nesting - Calls to **describe** can be nested, with specs defined at any level - tree of functions composition.
Before a spec is executed, Jasmine walks down the tree executing each **beforeEach** function in order.

```
it("should be full from the start", function () {
  coffee = new Coffee();
  expect(coffee.status).toEqual("full");
});

it("should be happy from the start", function () {
  coffee = new Coffee();
  expect(person.isHappy).toBeTruthy();
});
describe("Time", function () {

  beforeEach(function () {
    coffeeToDrink = new Coffee();
    personToDrink = new Person();

    currentDate = new Date();
    jasmine.clock().install()
    jasmine.clock().mockDate(currentDate);

  });

  afterEach(function () {
    jasmine.clock().uninstall();
  })
```

SpecRunner.html

```
<!-- include source files here... -->
<script src="src/Coffee.js"></script>
<script src="src/Person.js"></script>

<!-- include spec files here... -->
<script src="spec/helpers/SpecHelper.js"></script>
<script src="spec/DrinkerSpec.js"></script>
```

```
Person.prototype.drink = function (coffee) {

        if (!coffee) throw new Error ("Null Pointer");

        this.lastDrink = new Date();
        this.isHappy = false;
        coffee.status = "empty";
};


/**
  * @desc refill a cup of coffee
  * @param object coffee - the cup of coffee
  * @throws error if no coffe is passed as parameter
  * @throws error if no coffe coffee is already full
*/
Person.prototype.refill = function (coffee) {

        if (!coffee) throw new Error ("Null Pointer");

        if (coffee.status === "full") {
                throw new Error("Coffee is already full");
        }

        this.isHappy = true;
        coffee.status = "full";
};
```

```
Person.prototype.spill = function (coffee) {

        if (!coffee) throw new Error ("Null Pointer");

        coffee.status = "empty";
        this.isHappy = false;
}

/**
  * @desc function tells if the person is needing coffee
  * Person needs coffee when 3 hours have passed since last drink,
  * Or if it hasn't had any coffee yet!
*/
Person.prototype.needsCoffee = function(){

        if (!this.lastDrink) return true;
        return (new Date() - this.lastDrink) / 3600000.0 >= 3.0;
}
```

.tick(*nºmilliseconds*

Nesting - Calls to
at any level - tree
Before a spec is
executing each **be**

Jasm

- Used for
  manipulation,
  handling and animatio
- Works across s
  browsers.
- **Allows layout valid**
  of web pages.

Jasmine Plugin

jQuery

- Used for HTML manipulation, event handling and animation.
- Works across several browsers.
- **Allows layout validation of web pages.**

```
describe("JQuery", function () {

        var fixture;
        beforeEach(function () {
                loadFixtures('test.html');
        })


        it('should have a cool title', function () {

                title = $('h1')
                expect(title).toHaveText("MESW Login");
        });
```

MESW Login

Username

Password

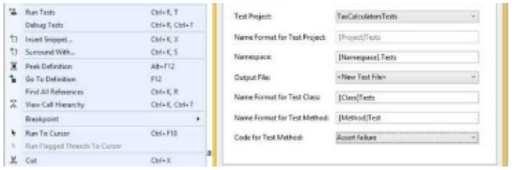
Submit

Code Coverage Tool

KARMA

Only requires type one line in cmd:
"karma start"

- **JSCover -** free and coverage measured via a browser
- **BLANKET.JS** - deprecated for jasmine version
- **jasmine istanbul** - embedded in KARMA
- **KARMA** - spawns a web server that executes source code against test code for each of the browsers connected



Exercise

Can you find the errors in source code?
Achieve 100% coverage and add new test case(s).

all files src/

70% Statements 21/30 **42.86%** Branches 6/14 **77.78%** Functions 7/9 **74.07%** Lines 20/27

File	Statements		Branches		Functions		Lines	
Movie.js	100%	6/6	100%	0/0	100%	3/3	100%	6/6
Television.js	62.5%	15/24	42.86%	6/14	66.67%	4/6	66.67%	14/21

Follow the instructions provided in:
https://github.com/mzamith/jasmine-test-cases

References

- iTGROW Software and Systems Training
- ISTQB Foundation Level (materials provided during the course)
- Software testing & QA blog by TestFort, 2016
- "Microsoft Developer Network", available at: https://msdn.microsoft.com/en-us
- Stage1 Magazine
- "Tutorialspoint", available at: https://www.tutorialspoint.com
- "Embedded Insights", available at: http://www.embeddedinsights.com/

> Fools ignore complexity. Pragmatists suffer it.
> Some can avoid it. Geniuses remove it.
> Alan J. Perlis

References

- iTGROW Software and Systems Training
- ISTQB Foundation Level (materials provided during the course)
- Software testing & QA blog by TestFort, 2016
- "Microsoft Developer Network", available at: https://msdn.microsoft.com/en-us
- Gadget Magazine
- "Tutorialspoint", available at: https://www.tutorialspoint.com
- "Embedded Insights", available at: http://www.embeddedinsights.com/channels/topics/unit-test/
- "Jasmine" GitHub, available at: http://jasmine.github.io/2.0/introduction.html
- "KARMA", available at: https://karma-runner.github.io/0.8/intro/configuration.html

> Fools ignore complexity. Pragmatists suffer it.
> Some can avoid it. Geniuses remove it.
>
> Alan J. Perlis

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

UNIT TESTING

Ana Rita Silva, Manuel Zamith TVVS

Friday, October 7, 2016

What is this?

Unit Testing Techniques

Black Box Testing - Testing a system without having specific knowledge of the internal workings of the system, no access to the source code, and no knowledge of the architecture.

White/Clear Box Testing - Testing a system with full knowledge and access to all source code and architecture documents.

Grey Box Testing - Testing a system while having at least some knowledge of the internals of a system.

Why is this used?

- Tests one specific behavior
- Improves efficiency and reliability
- Can act as documentation of the system - the unit test itself is used to verify the design
- You can be more confident about your code
- Each test is autonomous

Benefits

Limitations

- Time consuming
- Does not show absence of errors
- Integration errors or system errors can be missed
- Test code can have more bugs than the code it is testing
- Value and accuracy of unit tests can be diminished if initial conditions are not set correctly. There are no tests for the tests, you don't know
- Does not help to define a product or a requirement - can lead to a unit test checking for the wrong thing, and writing code that passes the test

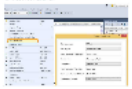
Unit Testing Tools

C/C++

PHP

Javascript

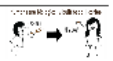
.NET & others

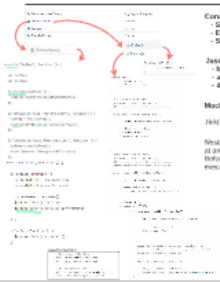


Code Coverage Tools for Javascript

- **JSCover** - free and coverage measured via a browser
- **BLANKET.JS** - deprecated for jasmine version
- **jasmine istanbul** - embedded in KARMA
- **KARMA** - spawns a web server that executes source code against test code for each of the browsers connected

Using Jasmine & KARMA





Concepts:
- **Specs** - function/test that contains 1 or + expectations
- **Expectation** - assertion that something is T/F
- **String** - title of the spec

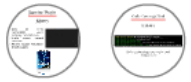
Jasmine Setup and Teardown:
- **beforeEach** - function called once before each spec
- **afterEach** - function called once after each spec
- **describe** - which function is called

Mocks & stubs

jasmine.clock()

tick(milliSecondsToTick); | install(); | uninstall() | mockDate();

Nesting - Calls to **describe** can be nested, with specs defined at any level - tree of functions corresponding.
Before a spec is executed, Jasmine walks down the tree executing each **beforeEach** function in order.



Exercise

Can you find the errors in source code?
Achieve 100% coverage and add new test case(s).

Follow the instructions provided in:
https://github.com/krampstudio/jasmine-test-users

References

> "Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it."
> Alan J. Perlis

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO