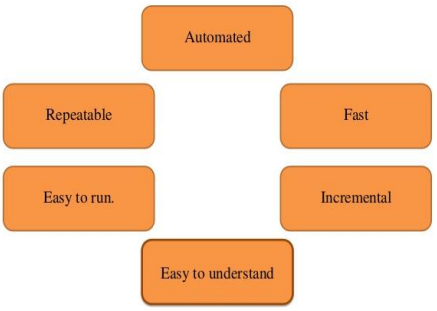
Jasmine & Karma

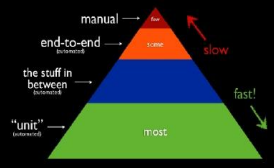
Good Unit Test Characteristics



Software Testing

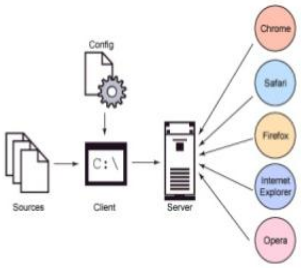
- Test software against a specification.
  - A specification for the system's behavior is the starting point.
  - Can write these specifications in code.
- Also, tests can evaluate a program's correctness after a modification.

The Healthy Test Pyramid

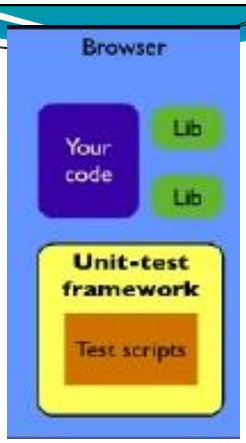


- the balance between many fast test and a few slow tests;

Test Setup



Architecture of the Test Harness



Jasmine

- A behaviour-driven testing framework for JavaScript language.
- Helps you express the intention (requirements) in code.
- SpecRunner.html: shows test results after execution of the tests.

Steps

- Add the src functions / .js file in **src** dir.
  - Update path in SpecRunner.html
- Add specs / .js file in **spec** dir.
  - Update path in SpecRunner.html
- Run SpecRunner.html in browser.

Jasmine Code / Keywords

```
describe("Cash withdrawal", function() {
    it("overdraft", function() {
      expect(getCash()).toEqual("Not  allowed!");
    });
});
```

- describe("Cash withdrawal" ... → test **suite**.
  - Typically defines a component of the application.
    - Could be a class, a function.
- Inside of that suite (an anonymous function), is the it() block.
  - This is called a specification, or a **spec** (in short).
  - It's a JavaScript function.
- toEqual() : a matcher

Test Result

Spec for "disemvoweler."

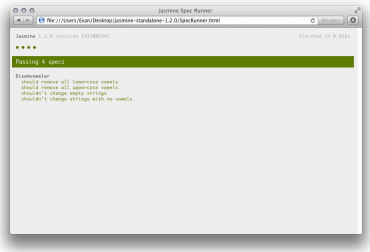
- A disemvoweler removes all vowels from a string.
- It should remove all lowercase vowels.
- It should remove all uppercase vowels.
- It shouldn't change empty strings.
- It shouldn't change strings with no vowels.

- E.g. Remove all lowercase vowels: "Hello world" should become "Hll wrld".

Example

```
describe("Disemvoweler", function() {
    it("should remove all lowercase vowels", function() {
    expect(disemvowel("Hello world")).toEqual("Hll wrld");
    });
    it("should remove all uppercase vowels", function() {
    expect(disemvowel("Artistic Eagle")).toEqual("rtstc gl");
    });
    it("shouldn't change empty strings", function() {
    expect(disemvowel("")).toEqual("");
    });
    it("shouldn't change strings with no vowels", function() {
    expect(disemvowel("Mhmm")).toEqual("Mhmm");
  });
});
```

Test Result for “disemvoweler.”



Testing Component-wise

- Testing all in one shot:

```
describe("calculator addition", function() {
    it("can add, subtract, multiply, and divide positive integers",
        function() {
            var calc = new Calculator;

            expect(calc.add(2, 3)).toEqual(5);
            expect(calc.sub(8, 5)).toEqual(3);
            expect(calc.mult(4, 3)).toEqual(12);
            expect(calc.div(12, 4)).toEqual(3);
    });
});
```


Testing Components (Alt. approach)

```
describe("calculator addition", function() {
        var calc;
        beforeEach(function() {
            calc = new Calculator();
        });

        it("can add positive integers", function() {
            expect(calc.add(2, 3)).toEqual(5);
        });
        it("can subtract positive integers", function() {
            expect(calc.sub(8, 5)).toEqual(3);
        });
        it("can multiply positive integers", function() {
            expect(calc.mult(4, 3)).toEqual(12);
        });
        it("can divide positive integers", function() {
            expect(calc.div(12, 4)).toEqual(3);
        });
});
```


Focussed on Black-Box Testing

- Testing the person object has a function that includes a *private* method:

```
var  person = {
     // Private method
     _generateHello: function() {
        return "hello";
    },
     // Public method
     helloWorld: function() {
        return this._generateHello() + " world";
     }
};
```

Matchers

- Equality: toEqual:
  - E.g. expect([1, 2, 3]).toEqual([1, 2, 3]);
- Identity: toBe: checks if two things are the same object, not just if they are equivalent. (=== operator).

```
var arr = [1, 2, 3];
expect(arr).toEqual([1, 2, 3]); // success; equivalent
expect(arr).toBe([1, 2, 3]); // failure; not the same array
```


Karma

Karma

- A simple tool that allows execution of JavaScript code in multiple real browsers.
  - All the major browsers are supported.
- Karma is not a testing framework, neither an assertion library.
  - Designed for low level (unit) testing.
  - It automates the test execution process.
- It works well with the following test frameworks :
  - Jasmine
  - QUnit
  - Mocha

Karma: Steps

- Installation:

```
$ npm install -g karma
$ npm install -g karma-cli
$ karma --version
```

Go to the jasmine proj dir.

- To create a config file:

```
$ karma init
```

- Once the configuration is done:

```
$ karma start
```