TEST-DRIVEN JAVASCRIPT

EIRIK REKSTEN (@EIREKSTEN)

TOOLING

KARMA

http://karma-runner.github.io/0.12/index.html

Karma is essentially a tool which spawns a web server that executes source code against test code for each of the browsers connected. The results for each test against each browser are examined and displayed via the command line to the developer such that they can see which browsers and tests passed or failed.

MOCHA

http://visionmedia.github.io/mocha/

Javascript Test Framework

```
// test suite
describe('factorizer', function () {

    var expect = chai.expect; // use any assertion library you want

    // nested test suite
    describe('factorize', function () {

        // test case
        it('should factorize the number 6 correctly', function () {
            var solution = factorizer.factorize(6);

            expect(solution).to.have.length(2);
            expect(solution).to.include.members([2,3]);
        });
    });
});
```

CHAI.JS

http://chaijs.com/

Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework. Supports different assertion styles:

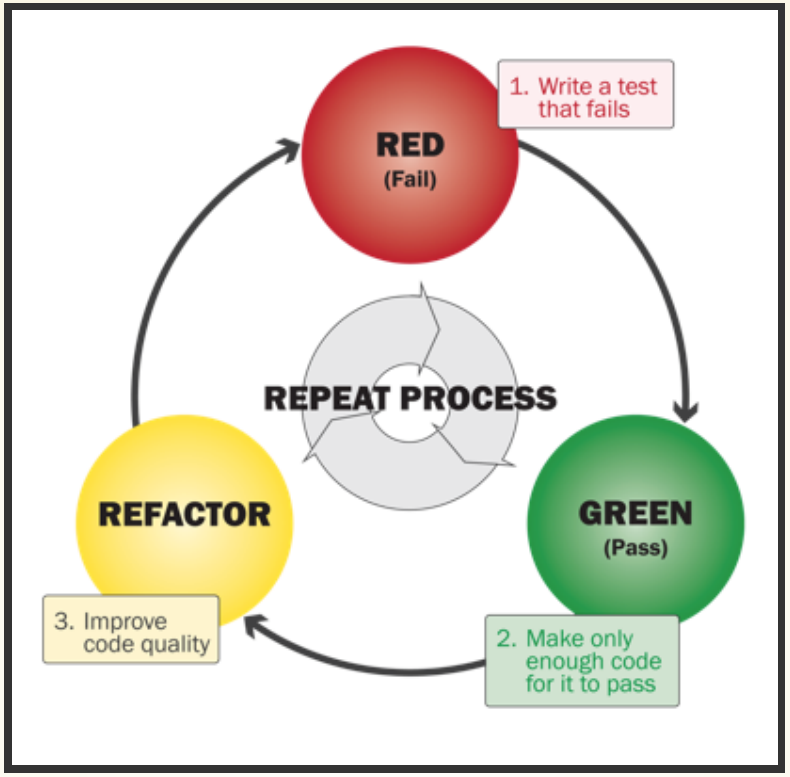
```
// BDD should style
foo.should.be.a('string');
foo.should.equal('bar');

// BDD expect style
expect(foo).to.be.a('string');
expect(foo).to.equal('bar');

// Traditional assertions
assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
```

EXERCISE 1 & 2

SETTING UP THE ENVIRONMENT AND RUNNING SOME TESTS

https://github.com/eireksten/jstdd

TEST-DRIVEN DEVELOPMENT

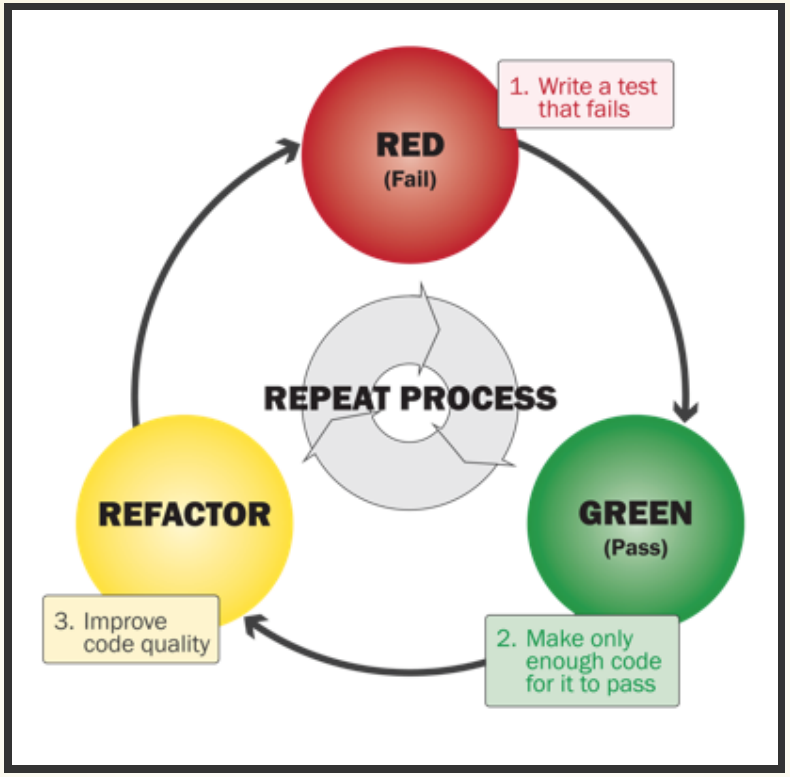

RED
(Fail)
1. Write a test that fails
REPEAT PROCESS
GREEN
(Pass)
2. Make only enough code for it to pass
REFACTOR
3. Improve code quality

THREE RULES OF TDD

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

BENEFITS OF TDD

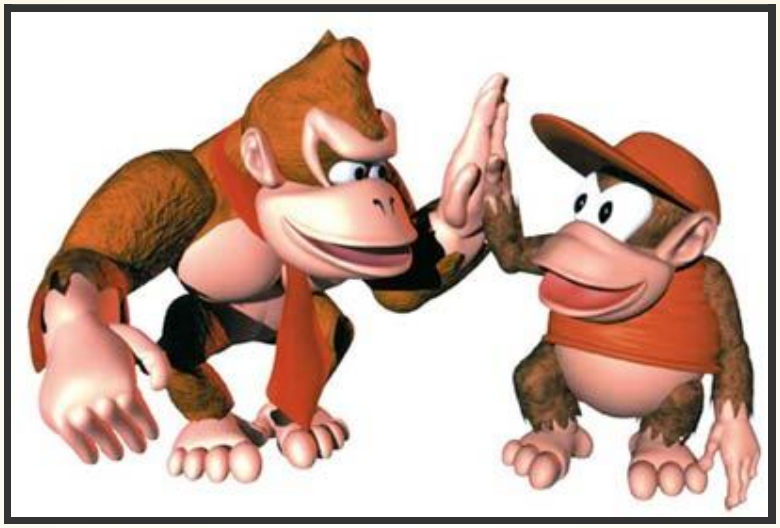
- Makes you think about the important stuff, what the code should do, instead of just how it should do it.
- Helps decoupling each unit from the rest of the system.
- More fun than writing tests for a system that already works.
- Lets you spend less time debugging!
- Makes refactoring a little less scary.
- Documents how the unit can be used.
- And so on...

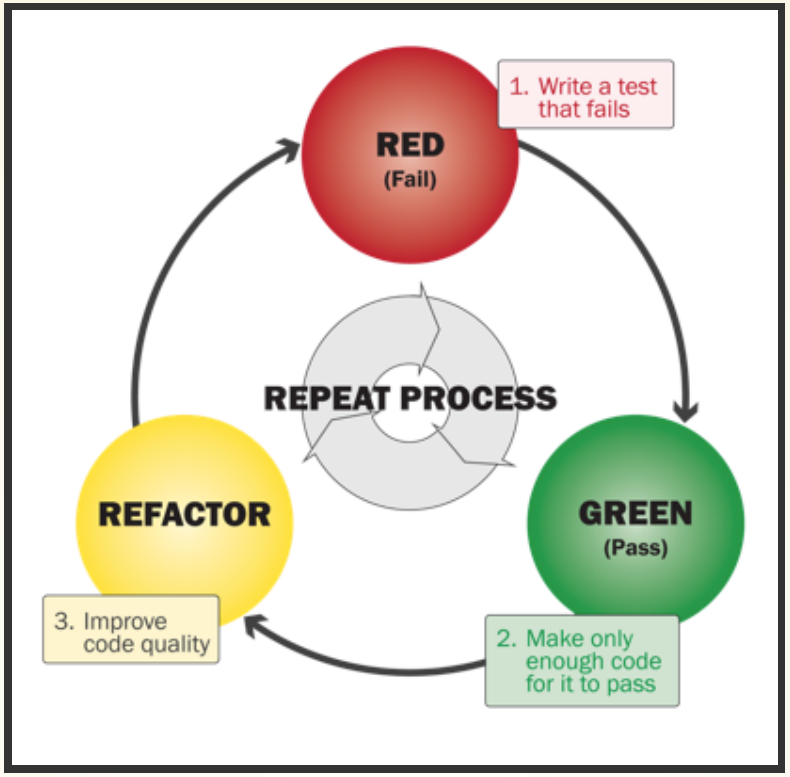

RED
(Fail)
1. Write a test that fails
REPEAT PROCESS
GREEN
(Pass)
2. Make only enough code for it to pass
REFACTOR
3. Improve code quality

WHAT KIND OF TESTS TO WRITE?

- The smallest logical step to improving the code!
- Tests should form a design/requirements specification for the code!
- Delete unnecessary or duplicated tests when done!

DONKEY AND DIDDY KONG




RED
(Fail)
1. Write a test that fails
REPEAT PROCESS
GREEN
(Pass)
2. Make only enough code for it to pass
REFACTOR
3. Improve code quality

EXERCISE 3 - BOWLING GAME KATA

https://github.com/eireksten/jstdd

TESTING MORE COMPLEX COMPONENTS

SINON.JS

http://sinonjs.org/

sinon.js is a library that provides a lot of different testing utilities for javascript, including:

- Spies, stubs and mocks
- Fake server
- Fake timers

FOURTH RULE OF TDD

ALL UNITS CAN BE TESTED!

EXERCISE 4 - TESTING A UI COMPONENT

https://github.com/eireksten/jstdd

REFERENCES

- *Test-Driven Javascript - Christian Johansen*
- **Karma Test Runner**
- **Mocha**
- **chai.js**
- **sinon.js**