TEST DRIVEN DEVELOPMENT

CONTENTS

CODE ROT

SMELLS

- Rigidity
- Fragility

EFFECTS ON A PROJECT

- Estimates grow and blown
- Unpredictability

FEAR OF CLEANING THE CODE

We might break it!

COMPREHENSIVE SUITE OF TESTS

Automated

Repeatable

Runs in minutes

Single command / button click


NO FEAR OF CLEANING THE CODE!!!44!!4!.

But… how?

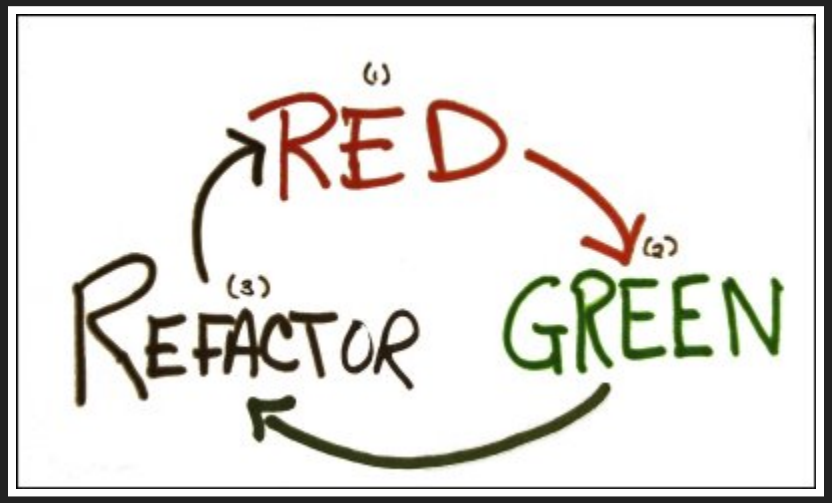
TEST DRIVEN DEVELOPMENT

... may seem strange

3 LAWS

1. Must not write any production code until you have a failing unit test first.
2. Must not write more of a test, than is sufficient to fail. Not compiling is failing.
3. Must not write more production code, than is sufficient to make the test pass.

RED-GREEN-REFACTOR



Focus on ONE thing in each step!

REFACTORING

Cleaning up the mess/code

> *Any fool can write code that a computer can understand. Good programmers write code that humans can understand.*

- Martin Fowler

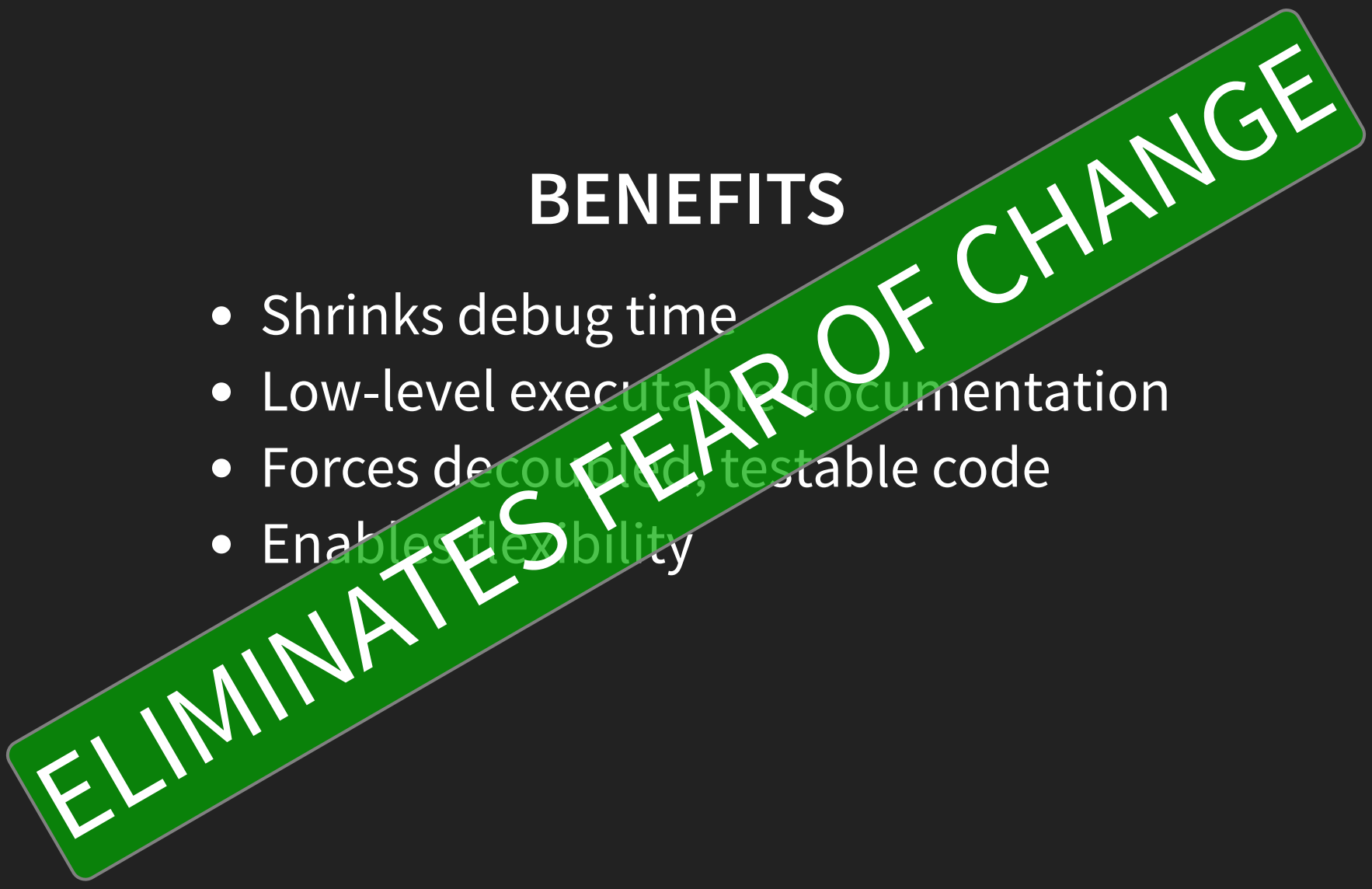
REFACTORING

- Remove duplications
- Introduce abstractions
- Introduce design/coding patterns

Applies to the tests, too!

BENEFITS

- Shrinks debug time
- Low-level executable documentation
- Forces decoupled, testable code
- Enables flexibility



XUNIT

```
var Calculator = function() ;
Calculator.prototype.add = function(a, b) {
    return a + b;
};
```

In a test-framework like JUnit, QUnit, PHPUnit -> xUnit

```
TestCase('CalculatorTests', {
    testAdd_GivenTwoNumbers_ReturnsSumOfThem: function() {
        var calculator = new Calculator();
        var result = calculator.add(1,2);
        assertEquals(3, result);
    }
});
```

```
TestCase('CalculatorTests', {
    setUp: function() {
        this.calculator = new Calculator();
    },
    testAdd_GivenTwoNumbers_ReturnsSumOfThem: function() {
        assertEquals(3, this.calculator.add(1,2));
    },
    testAdd_GivenAZero_ReturnsTheOtherNumber: function() {
        assertEquals(2, this.calculator.add(0,2));
        assertEquals(3, this.calculator.add(3,0));
    }
});
```

```
TestCase('CalculatorTests', {
    setUp: function() {
        this.calculator = new Calculator();
    },
    tearDown: function() {
        this.calculator.remove();
    },
    testAdd_GivenTwoNumbers_ReturnsSumOfThem: function() {
        assertEquals(3, this.calculator.add(1,2));
    }
});
```

NAMING TEST FUNCTIONS:

Class and method name

Scenario / context

Expected behaviour

```
testMethod_scenario_expectedBehaviour: function() {...},

testAdd_GivenTwoNumbers_ReturnsSumOfThem: function() {
    assertEquals(3, this.calculator.add(1,2));
}
```

```
function execute_Perfect_Perfect() {...}
function getSettings_Perfect_Perfect() {...}
function getSections_SectionExists_SectionsReturnedProperly() {...}
```

BDD STYLE

Frameworks like RSpec, Jasmine, Mocha

```
describe('Calculator', function() {
    describe('#add', function() {
        it('should return the sum of the given numbers', function() {
            var calculator = new Calculator();
            expect(calculator.add(1,2)).to.equal(3);
        });
    });
});
```

```
describe('Calculator', function() {
    var calculator;
    beforeEach(function() {
        calculator = new Calculator();
    });

    describe('#add', function() {
        it('should return the sum of the given numbers', function() {
            expect(calculator.add(1,2)).to.equal(3);
        });
        it('should return the first number if the second is zero', function() {
            expect(calculator.add(3,0)).to.equal(3);
        });
    });
});
```

```
describe('Calculator', function() {
    var calculator;
    beforeEach(function() { calculator = new Calculator(); });

    describe('#add', function() {
        describe('with positive number parameters', function() {
            it('should return the sum of them', function() {
                expect(calculator.add(1,2)).to.equal(3);
            });
        });
        describe('with zero parameter', function() {
            it('should return the other number', function() {
                expect(calculator.add(3,0)).to.equal(3);
            });
        });
    });
});
```

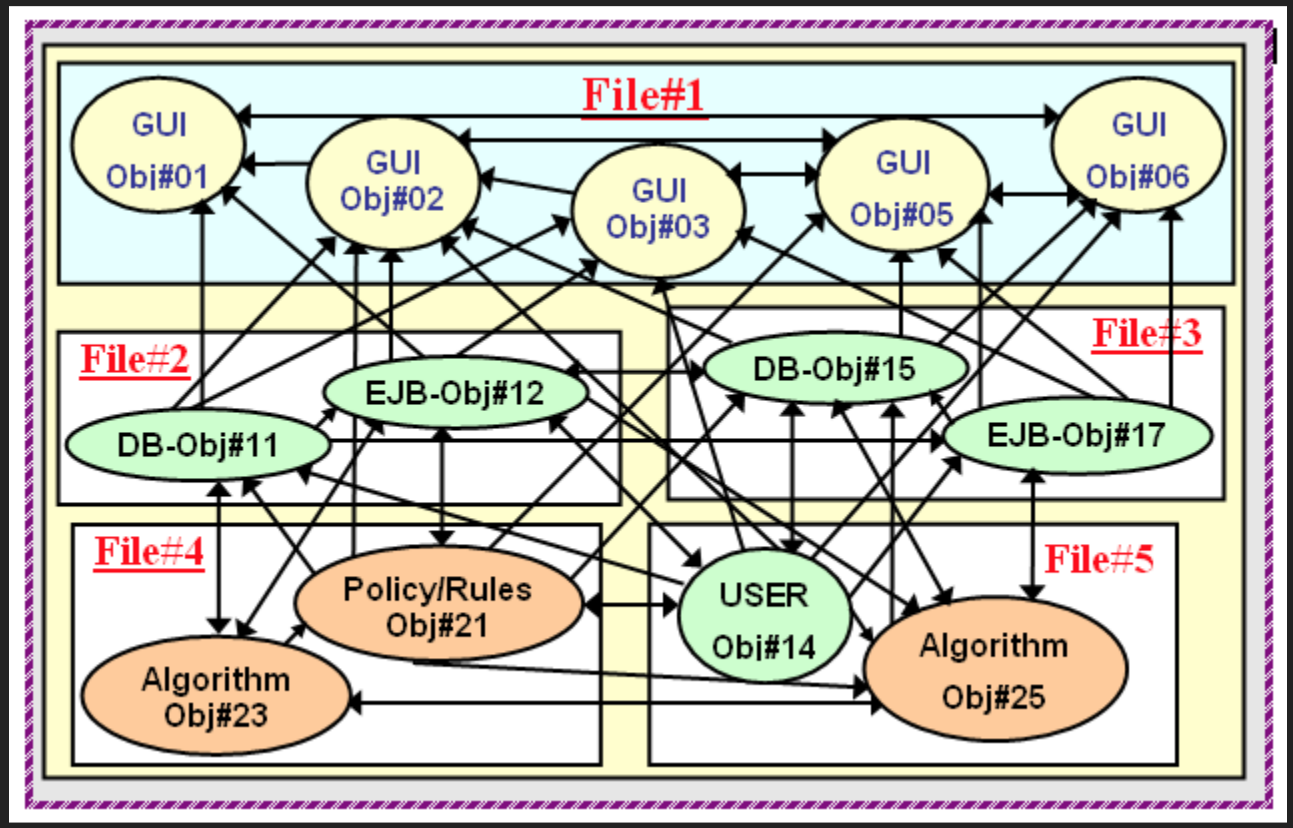
TESTING AFTER THE FACT

... is a bad idea

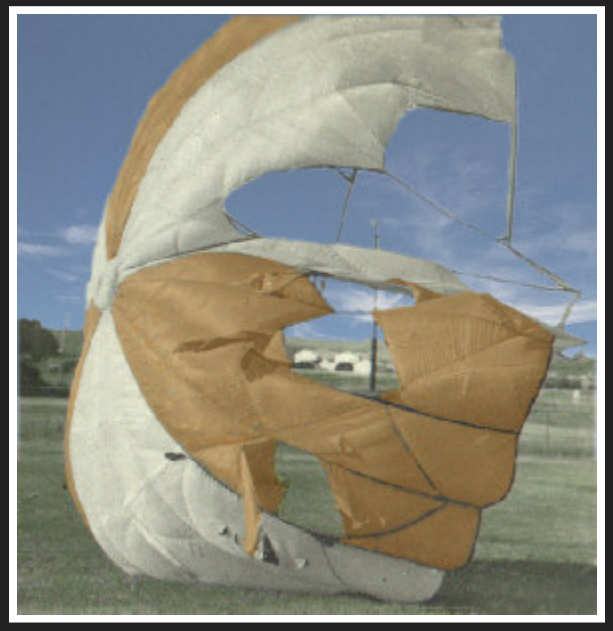
LESS FUN

HUMAN FACTOR

TIGHTLY COUPLED, UNTESTABLE CODE



CUTTING CORNERS, HOLES IN THE TESTS




ACTION