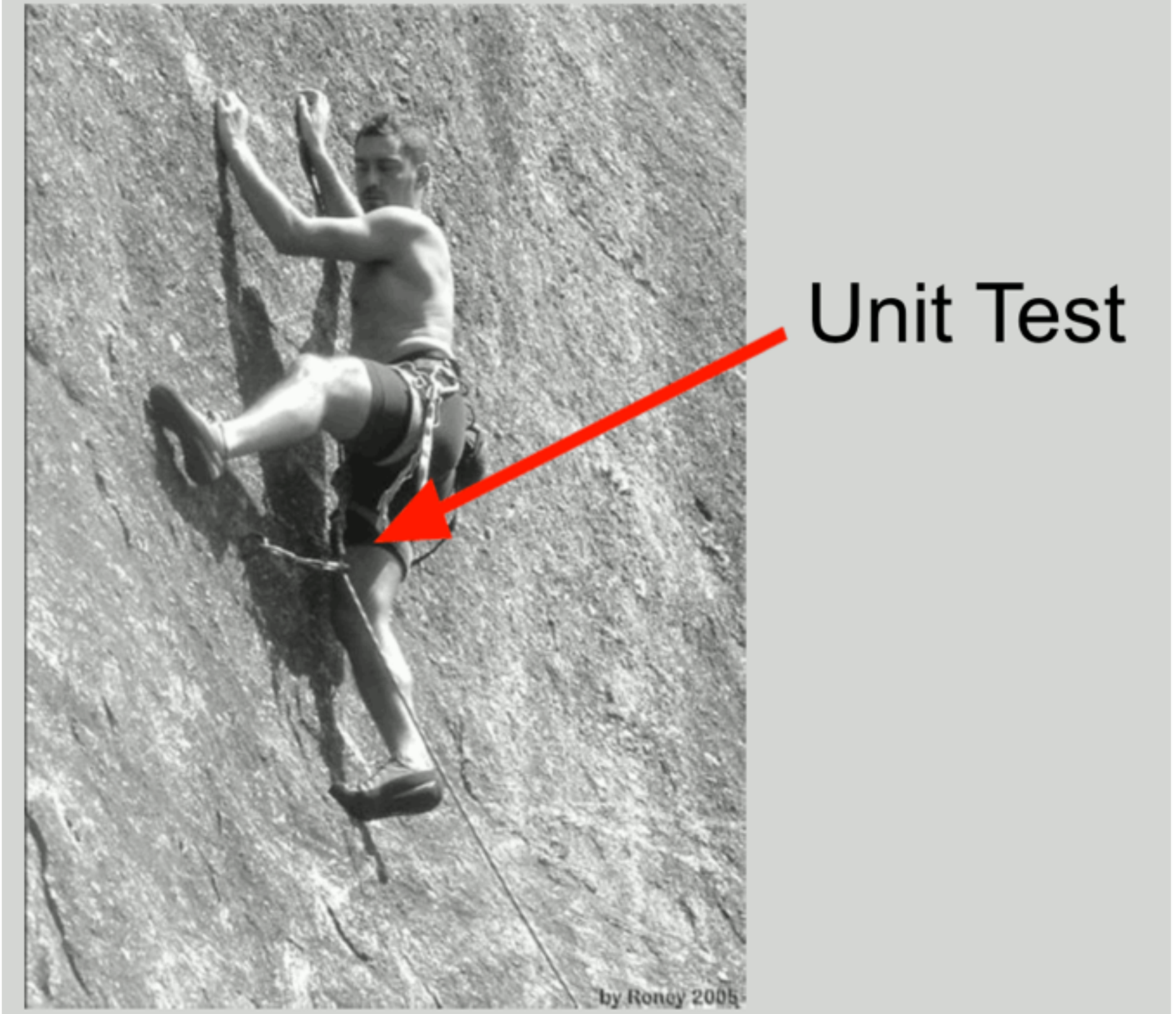
Unit

with JUnit



http://less.works/less/technical-excellence/unit-testing.html

"unit tests are so important that they should be a first class language construct"

- Jeff Atwood

← you know, the Coding Horror guy.

Why not write Unit test ?

Integration testing find more bugs

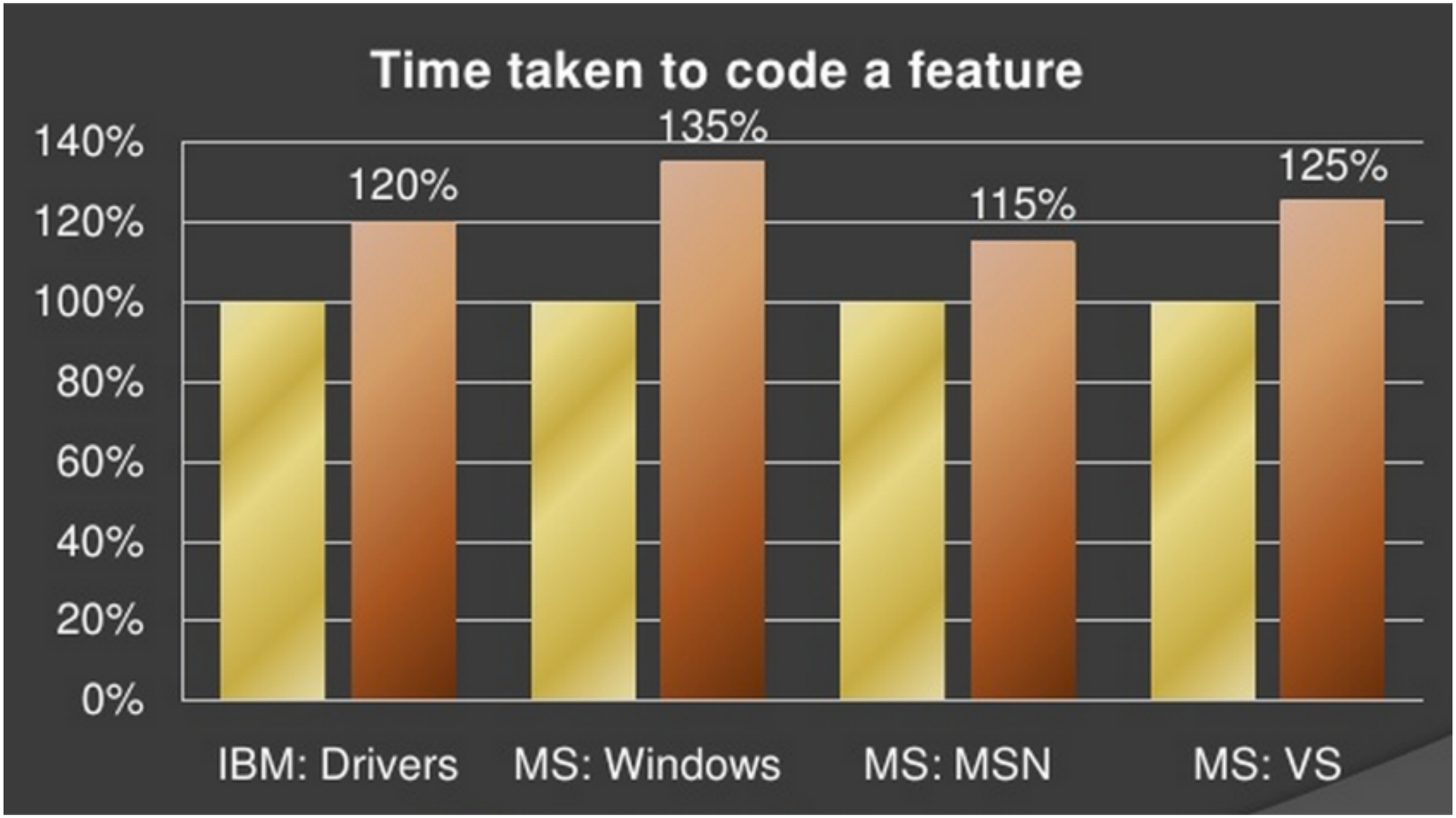
Why not write Unit test ?

Testing for QA Time = Cost

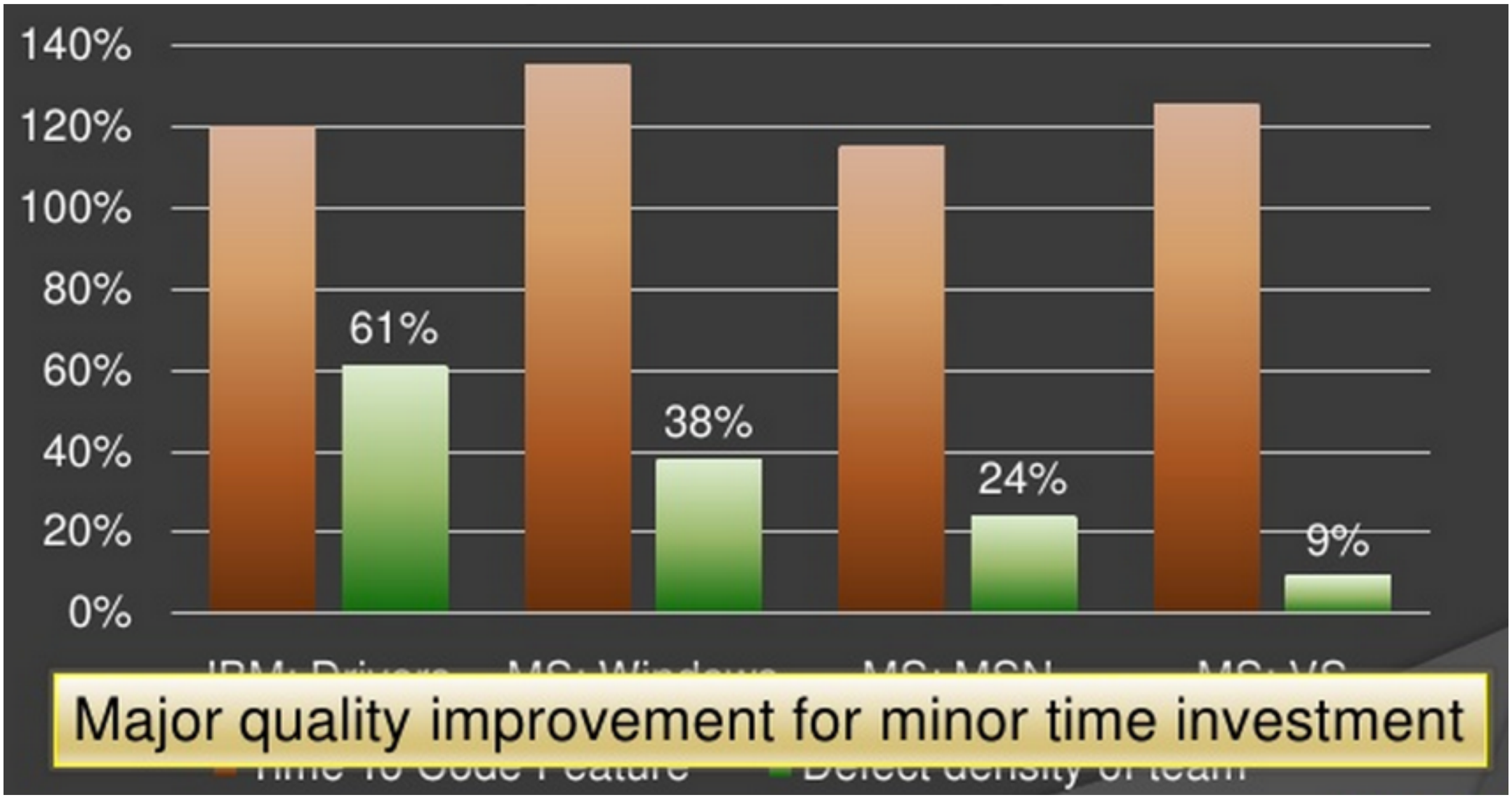
Manual testing find more bugs

Write more code

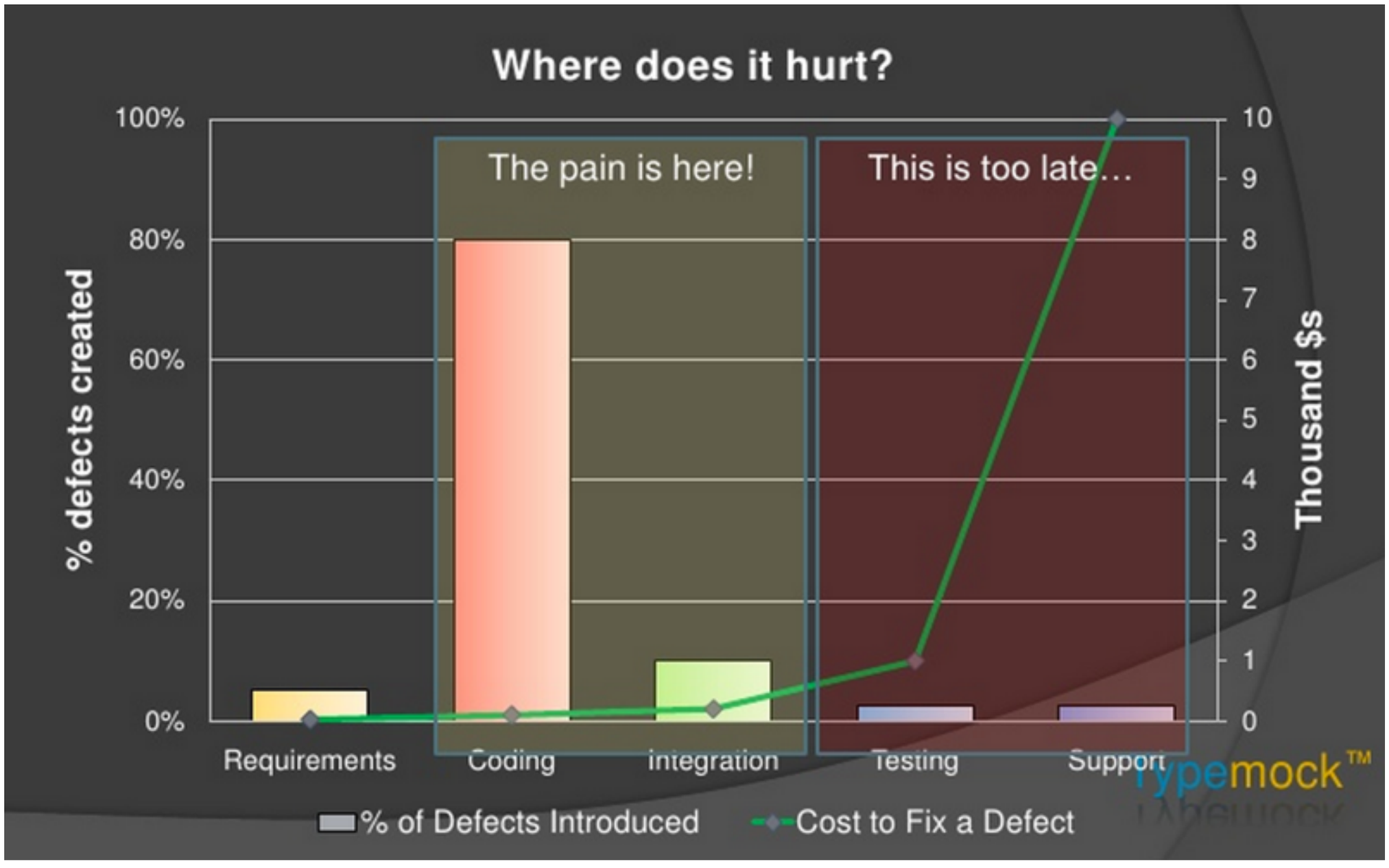
Waste time ?


Time taken to code a feature
140%
120%
100%
80%
60%
40%
20%
0%
120%
135%
115%
125%
IBM: Drivers
MS: Windows
MS: MSN
MS: VS

Waste time ?


140%
120%
100%
80%
60%
40%
20%
0%
61%
38%
24%
9%
Major quality improvement for minor time investment

Why Unit test ?



Anatomy JUnit 4

Test Name

```
public class FizzBuzzTest {

    @Test
    public void sayFizzWhenNumberIsDevidedByThree() {
        FizzBuzz fizzBuzz = new FizzBuzz();
        String actualResult = fizzBuzz.say(3);
        assertEquals("Fizz", actualResult);
    }

}
```

Annotation

Test Case Name

Test Naming

“What ’s in a name ?”

That which we call a rose

by any other name would smell

as sweet

Romeo and Juliet

Guide to Test Writing

Don’t say “**test**”, say “**should**”

SPRINT3R

Siam Chamnankit Co., Ltd., Odd-e (Thailand) Co., Ltd. and Alliance

Don’t use the word “test”

Use the word “should”

```
tranferShouldDeductSumFromSourceAccountBalance()

transferShouldAddSumLessFeesToDestinationAccountBalance()

depositShouldAddAmountToAccountBalance()
```

Guide to Test Writing

Don’t test your class, test **behaviour**

SPRINT3R

Siam Chamnankit Co., Ltd., Odd-e (Thailand) Co., Ltd. and Alliance

Guide to Test Writing

Test class names are important too

Structure your test well

Tests are **deliverable** too

SPRINT3R

Siam Chamnankit Co., Ltd., Odd-e (Thailand) Co., Ltd. and Alliance

Test Structure

The ratio of time spent (code)

versus writing is

over **10** to **1**

Robert C. Martin, Clean Code

Good Unit Test

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(3);

    assertEquals("Fizz", actualResult);
}
```

Arrange

Act

Assert

Setup Pattern

```java
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(3);

    assertEquals("Fizz", actualResult);
}

@Test
public void sayBuzzWhenNumberIsDevidedByFive() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(5);

    assertEquals("Buzz", actualResult);
}
```

Setup Pattern

Inline Setup

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(3);

    assertEquals("Fizz", actualResult);
}

@Test
public void sayBuzzWhenNumberIsDevidedByFive() {
    FizzBuzz fizzBuzz = new FizzBuzz();

    String actualResult = fizzBuzz.say(5);

    assertEquals("Buzz", actualResult);
}
```

Setup Pattern

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = setup();
    String actualResult = fizzBuzz.say(3);
    assertEquals("Fizz", actualResult);
}
```

Delegate Setup

```
@Test
public void sayBuzzWhenNumberIsDevidedByFive() {
    FizzBuzz fizzBuzz = setup();
    String actualResult = fizzBuzz.say(5);
    assertEquals("Buzz", actualResult);
}
```

```
private FizzBuzz setup() {
    FizzBuzz fizzBuzz = new FizzBuzz();
    return fizzBuzz;
}
```

Setup Pattern

```
@Before
public void initial() {
    fizzBuzz = new FizzBuzz();
}
```

Implicit Setup

```
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    String actualResult = fizzBuzz.say(3);
    assertEquals("Fizz", actualResult);
}

@Test
public void sayBuzzWhenNumberIsDevidedByFive() {
    String actualResult = fizzBuzz.say(5);
    assertEquals("Buzz", actualResult);
}
```

Implicit Teardown

```
@After
public void clearResources(){
    fizzBuzz = null;
}
```

Ignore Testcase

```
@Ignore("Pending implemetation")
@Test
public void sayFizzWhenNumberIsDevidedByThree() {
    FizzBuzz fizzBuzz = new FizzBuzz();
    String actualResult = fizzBuzz.say(3);
    assertEquals("Fizz", actualResult);
}
```

Handle Exception

SPRINT3R

Siam Chamnankit Co., Ltd., Odd-e (Thailand) Co., Ltd. and Alliance

Traditional approach

```
@Test
public void invalidWhenNumberLessThanOne() {
    try {
        fizzBuzz.say(0);
        fail();
    } catch (IllegalStateException expected) {
    }
}
```

Expected Annotation

```
@Test(expected=IllegalStateException.class)
public void invalidWhenNumberLessThanOne() {
        fizzBuzz.say(0);
}
```

ExpectedException Rule

```
@Rule
public ExpectedException thrown = ExpectedException.none();

@Test
public void invalidWhenNumberLessThanOne() {
    thrown.expect(IllegalStateException.class);
    fizzBuzz.say(0);
}
```

Data-Driven with JUnit

@Parameterized

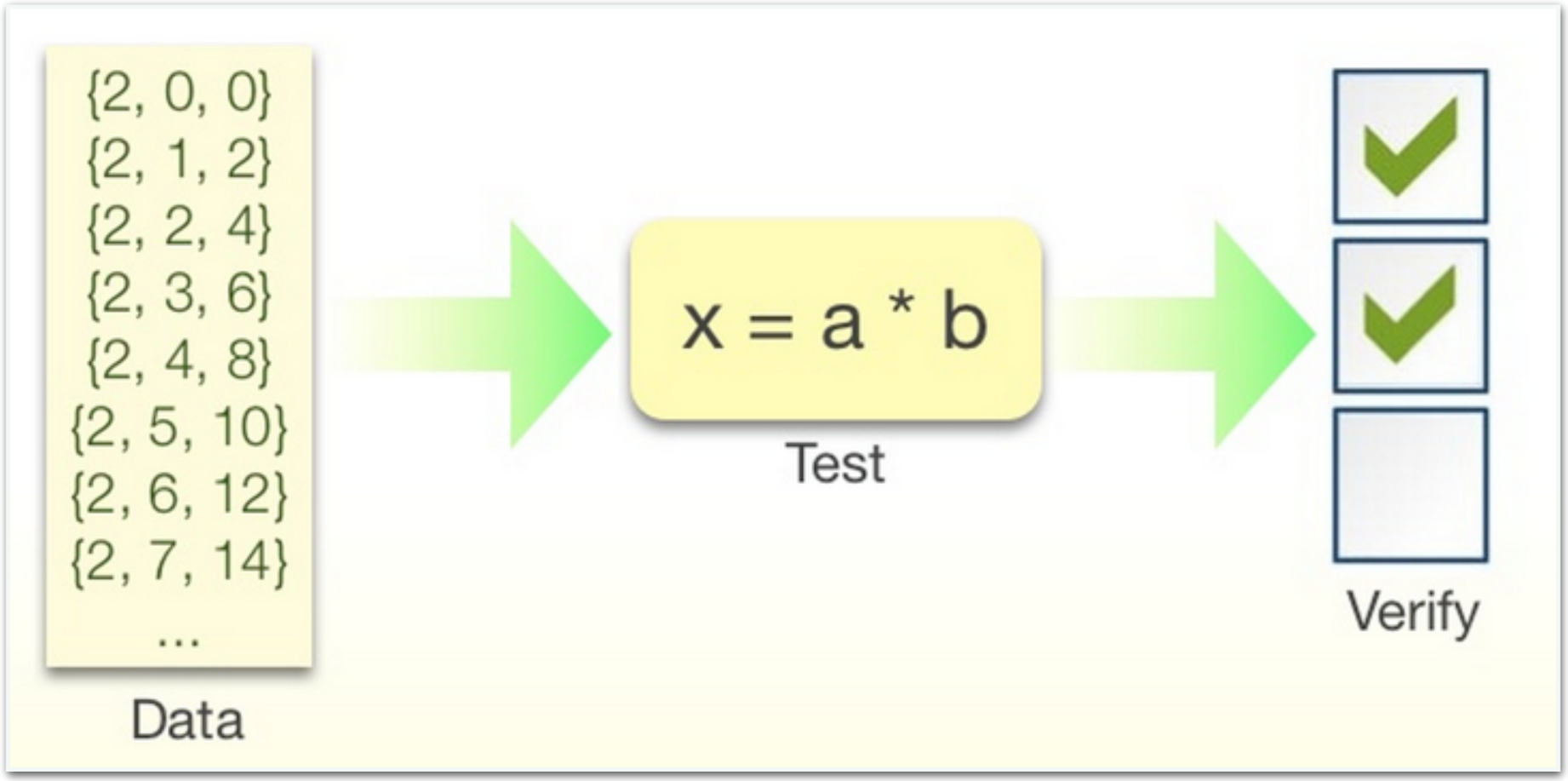
SPRINT3R

Siam Chamnankit Co., Ltd., Odd-e (Thailand) Co., Ltd. and Alliance

Using parameterized

Set of test data
Expected result
Define test that uses the test data
Verify result against expected result

Data-Driven Development



Demo

@Parameterized
with Calculate Grade

Input	Expected Result
80	A
70	B
60	C
50	D
40	F

Step 1 :: Test Data

Input	Expected Result
80	A
70	B
60	C
50	D
40	F

Step 2 :: Add Runner

```
@RunWith(Parameterized.class)
public class CalculateGradeTest {

    private int score;
    private String expectedGrade;

    public CalculateGradeTest(int score, String expectedGrade) {
        this.score = score;
        this.expectedGrade = expectedGrade;
    }

    @Test
    public void convertScoreToGrade() {
    }

}
```

Step 3 :: Matching fields

```
@RunWith(Parameterized.class)
public class CalculateGradeTest {

    private int score;
    private String expectedGrade;

    public CalculateGradeTest(int score, String expectedGrade) {
        this.score = score;
        this.expectedGrade = expectedGrade;
    }

    @Test
    public void convertScoreToGrade() {
    }

}
```

Step 4 :: Define test case

```
@RunWith(Parameterized.class)
public class CalculateGradeTest {

    private int score;
    private String expectedGrade;

    public CalculateGradeTest(int score, String expectedGrade) {
        this.score = score;
        this.expectedGrade = expectedGrade;
    }

    @Test
    public void convertScoreToGrade() {
    }

}
```

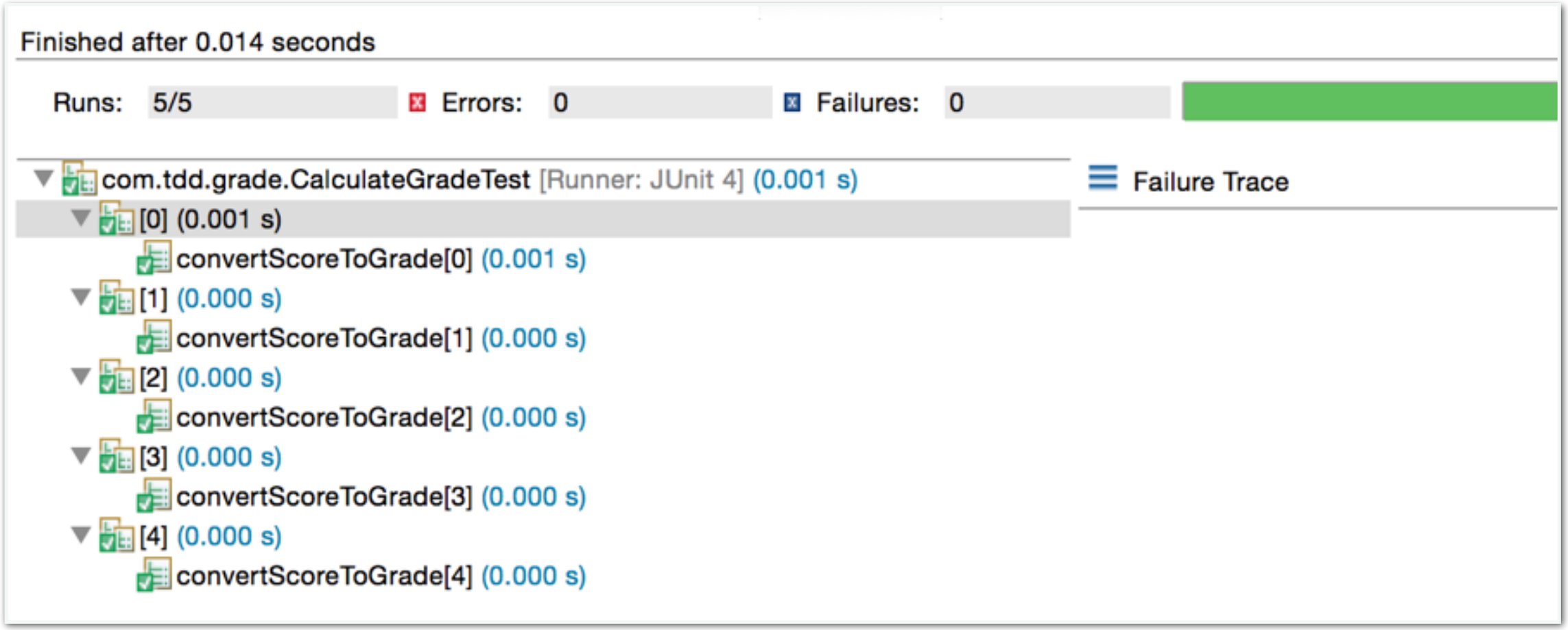
Step 5 :: Add @parameters

```
@RunWith(Parameterized.class)
public class CalculateGradeTest {

    private int score;
    private String expectedGrade;

    @Parameters
    public static Collection<Object[]> data(){
        return Arrays.asList(new Object[][]{
                {80, "A"},
                {70, "B"},
                {60, "C"},
                {50, "D"},
                {40, "F"}
        });
    }
```

Step 6 :: Test Result



Why not write Unit test ?