



TDD and following the Dimension

Test Driven Development

Michael Meissner 15.02.2023

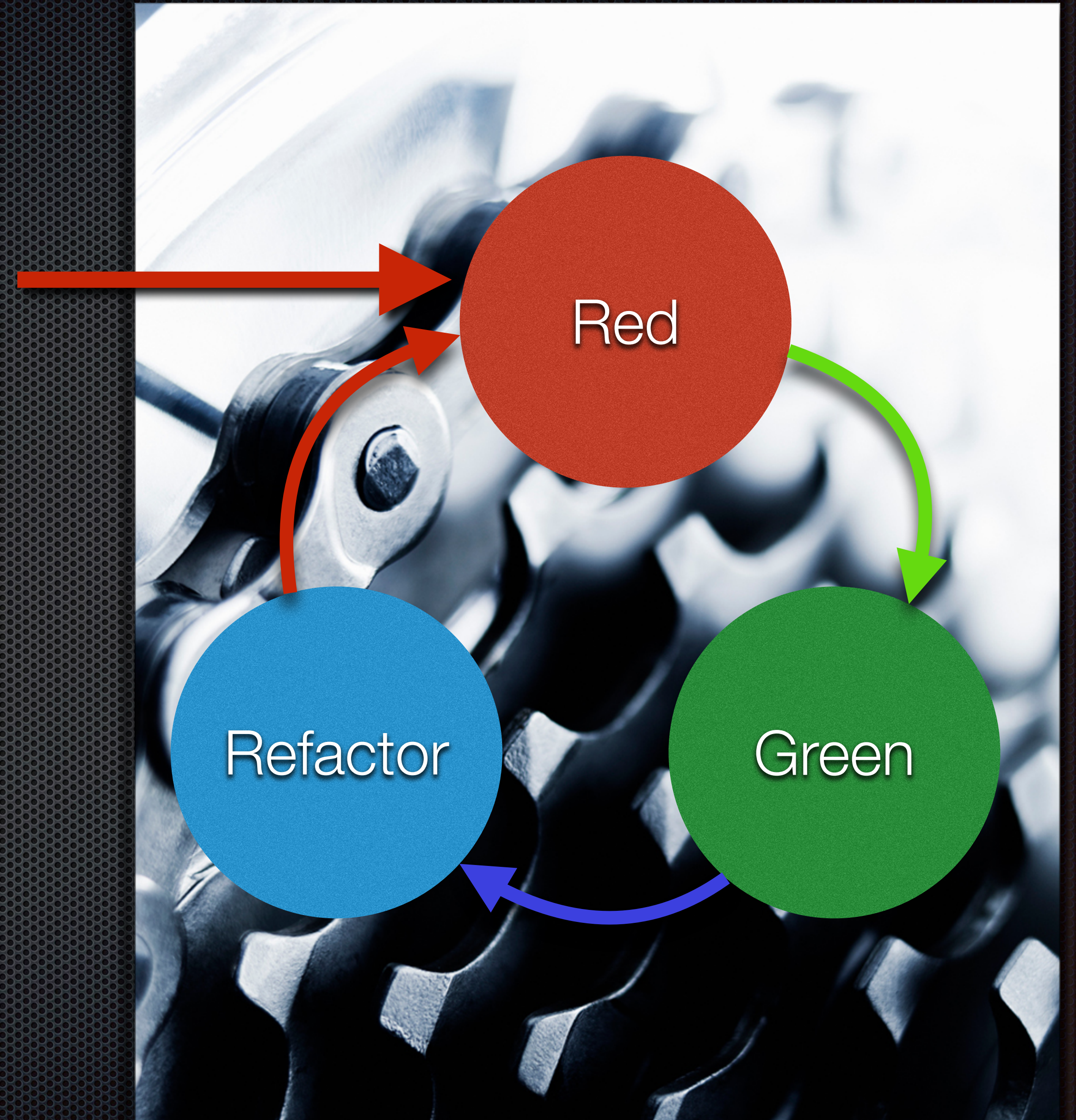
Agenda

- ✦ TDD - my short extract
- ✦ Testing Strategy
 - ✦ Following one Dimension
 - ✦ Example Roman Calendar



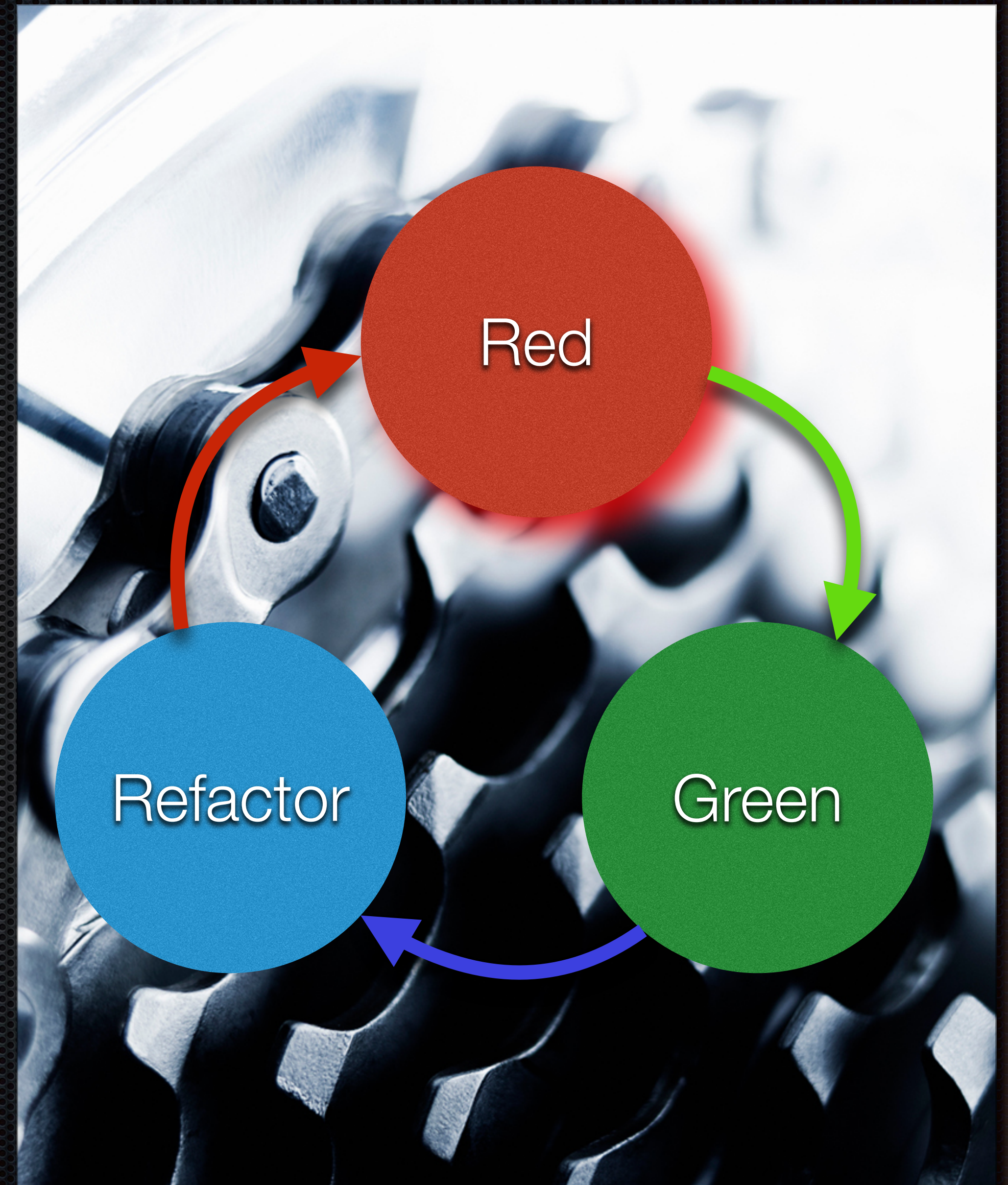
TDD : Circle

- Red - One Failing Test
- Make It Green
- Refactor



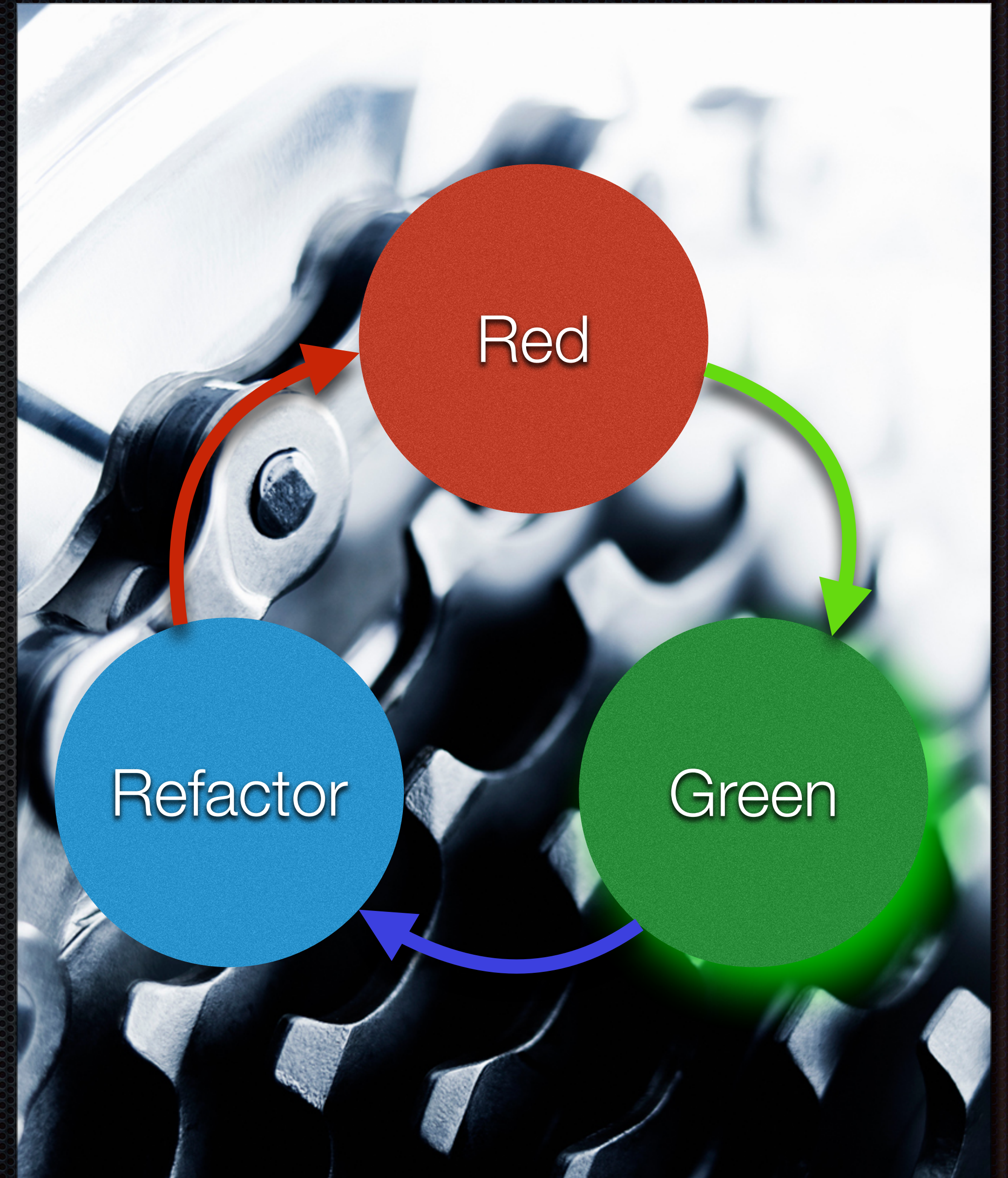
Red - One Failing Test

- write one simple Test failing for the right Reason
- test the behavior and not the implementation
- prioritize the happy path
- start with the assertion
- structure the test: arrange, Act and Assert
Naming Convention
<Class>Should : Test: <verb><Expectation>



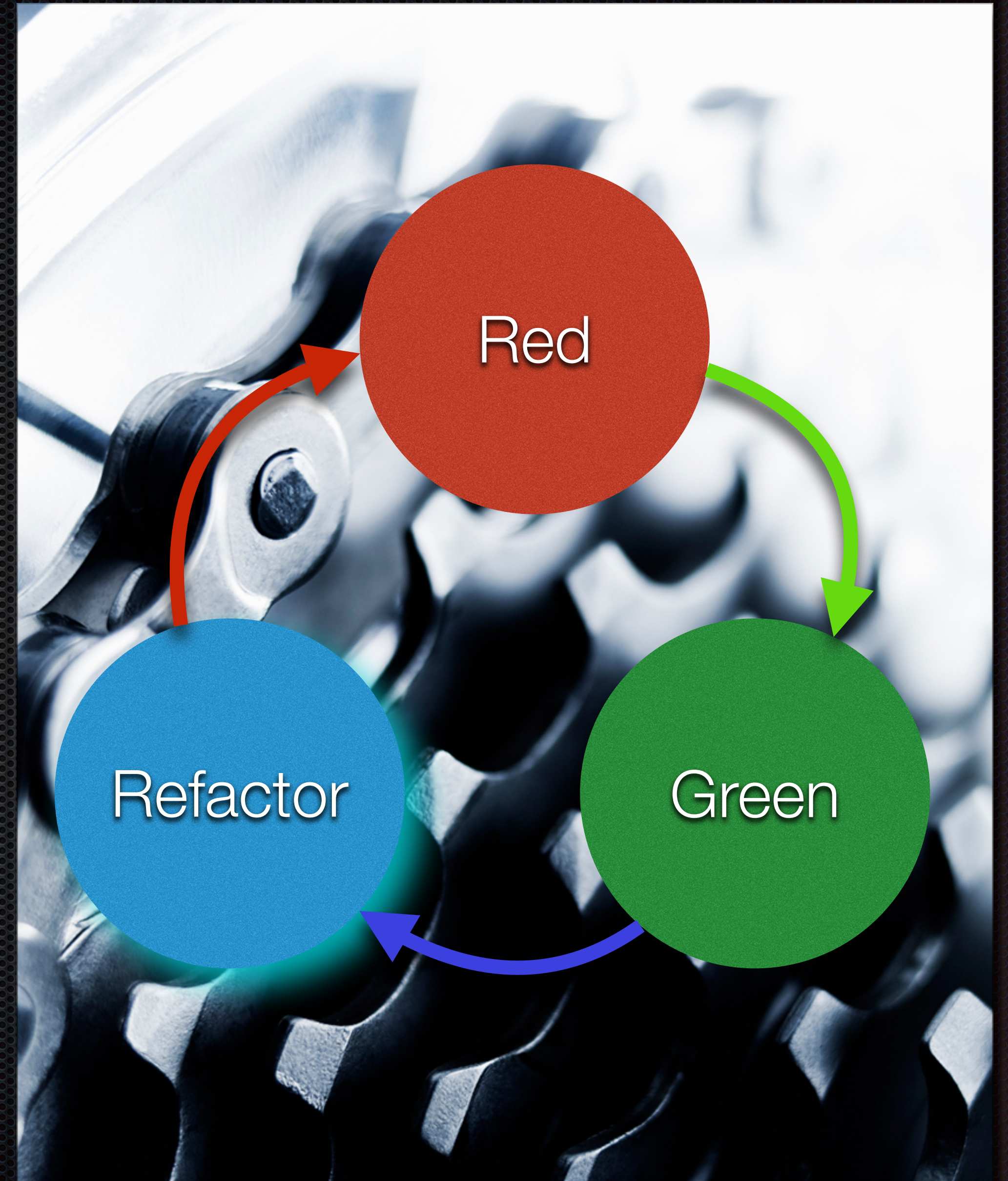
Make It Green

- Write just simplest code to pass all tests
 - Fake implementation
 - Triangulation
 - Obvious Implementation
 - Transformation Priority Premise
- Commit with name of test



Refactor

- Refactor Test and Production Code
 - better naming
 - remove Code duplications
 - „Rule of Three“
 - extract methods
- run Green and commit



TDD Habits - FIRST principles

→ Fast

- ◆ They should run very often, hence they must be fast: one second matter here

→ Isolated

- ◆ They should be no dependency between tests, hence they must run in any order

→ Repeatable

- ◆ They should always have the same result when run multiple times (no flaky tests)

→ Self validating

- ◆ Only two states: red or green. Absolutely no manual or human interpretation.

→ Timely

- ◆ Must be written at the right time (->BEFORE the code they suppose to test)

Testing Strategie

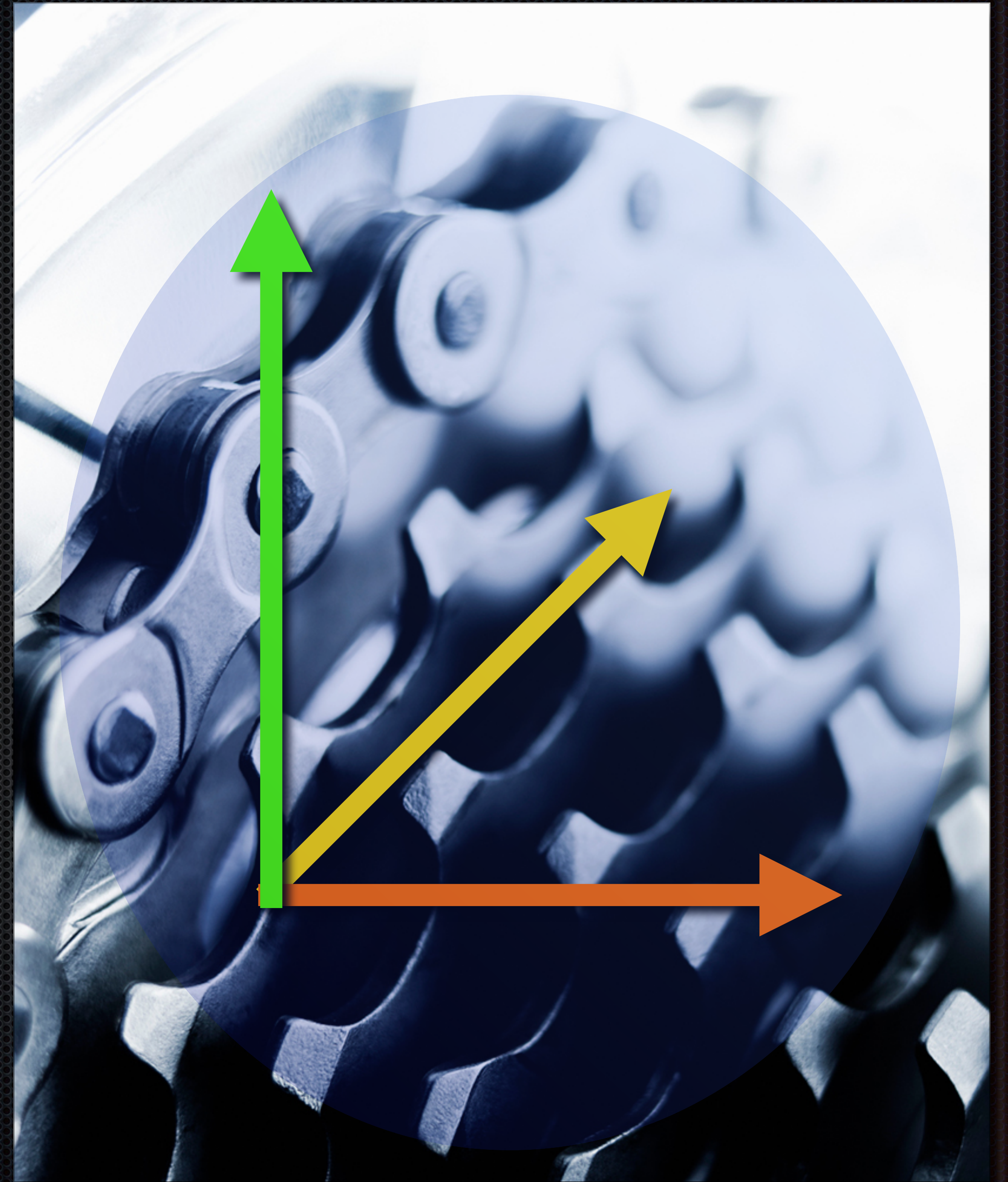
It's impossible to test the complete system.

So we test until we have sufficient confidence.



Follow the Dimension

Test one dimension of freedom, so that you have enough confidence for the production code , before switching to the next dimension



Example

Roman Calendar

Dimensions:

1. Additive Symbols I,X,C,M
2. Special Cases



1. Repetition of I, X, C

```
void setUp() {  
}
```

no usages alex +1 *

```
@ParameterizedTest
```

```
@CsvSource({
```

```
    "1, 'I'",  
    "2, 'II'",  
    "3, 'III'",  
    "10, 'X'",  
    "20, 'XX'",  
    "30, 'XXX'",  
    "100, 'C'",  
    "200, 'CC'",  
    "1000, 'M'",
```

```
1 public class RomanCalculator {
```

1 usage new *

```
2 public String calculate(int arabic) {
```

```
3     if(arabic % 1000==0){
```

```
4         return "M".repeat( count: arabic/1000);
```

```
5     }
```

```
6     if(arabic % 100==0){
```

```
7         return "C".repeat( count: arabic/100);
```

```
8     }
```

```
9     if(arabic % 10==0){
```

```
10        return "X".repeat( count: arabic/10);
```

```
11    }
```

```
12    return "I".repeat(arabic);
```

```
13 }
```

```
14 }
```

```
15
```

1. Repetition of I, X, C

```
void setUp() {  
}  
  
no usages alex +1 *  
@ParameterizedTest  
@CsvSource({  
    "1, 'I'",  
    "2, 'II'",  
    "3, 'III'",  
    "10, 'X'",  
    "20, 'XX'",  
    "30, 'XXX'",  
    "100, 'C'",  
    "200, 'CC'",  
    "1000, 'M'",  
})  
  
1 public class RomanCalculator {  
2     1 usage new *  
3     public String calculate(int arabic) {  
4         if(arabic % 1000==0){  
5             return "M".repeat( count: arabic/1000);  
6         }  
7         if(arabic % 100==0){  
8             return "C".repeat( count: arabic/100);  
9         }  
10        if(arabic % 10==0){  
11            return "X".repeat( count: arabic/10);  
12        }  
13        return "I".repeat(arabic);  
14    }  
15 }
```



```
void setUp() {  
}  
  
no usages alex +1 *  
@ParameterizedTest  
@CsvSource({  
    "1, 'I'",  
    "2, 'II'",  
    "3, 'III'",  
    "10, 'X'",  
    "20, 'XX'",  
    "30, 'XXX'",  
    "100, 'C'",  
    "200, 'CC'",  
    "1000, 'M'",  
})  
  
6 public RomanCalculator() {  
7     1 usage new *  
8     arabicToRoman.put(1000,"M");  
9     arabicToRoman.put(100,"C");  
10    arabicToRoman.put(10,"X");  
11    arabicToRoman.put(1,"I");  
12 }  
  
13 1 usage new *  
14 public String calculate(int arabic) {  
15     for ( Integer arabicValue: arabicToRoman.keySet() ) {  
16         if(arabic % arabicValue==0){  
17             return arabicToRoman.get(arabicValue).repeat( count: arabic/arabicValue);  
18         }  
19     }  
20     return "";  
21 }
```

2. additive numbers of different characters I, X, C

```
void setUp() {  
}  
  
no usages alex +1 *  
@ParameterizedTest  
@CsvSource({  
    "1, 'I'",  
    "2, 'II'",  
    "3, 'III'",  
    "10, 'X'",  
    "20, 'XX'",  
    "30, 'XXX'",  
    "100, 'C'",  
    "200, 'CC'",  
    "1000, 'M'",  
})  
public void returnSomething(int arabic) {  
    1 usage new *  
    public RomanCalculator() {  
        arabicToRoman.put(1000, "M");  
        arabicToRoman.put(100, "C");  
        arabicToRoman.put(10, "X");  
        arabicToRoman.put(1, "I");  
    }  
  
    1 usage new *  
    public String calculate(int arabic) {  
        for ( Integer arabicValue: arabicToRoman.keySet() ) {  
            if(arabic % arabicValue==0){  
                return arabicToRoman.get(arabicValue).repeat( count: arabic/arabicValue);  
            }  
        }  
        return "";  
    }  
}
```

```
no usages alex +1 *  
@ParameterizedTest  
@CsvSource({  
    "1, 'I'",  
    "2, 'II'",  
    "3, 'III'",  
    "10, 'X'",  
    "20, 'XX'",  
    "30, 'XXX'",  
    "100, 'C'",  
    "200, 'CC'",  
    "1000, 'M'",  
    "11, 'XI'",  
})  
public void returnSomething(int arabic) {  
    6  
    public RomanCalculator() {  
        7  
        arabicToRoman.put(1000, "M");  
        8  
        arabicToRoman.put(100, "C");  
        9  
        arabicToRoman.put(10, "X");  
        arabicToRoman.put(1, "I");  
    }  
  
    1 usage new *  
    public String calculate(int arabic) {  
        14  
        String result = "";  
        15  
        for ( Integer arabicValue: arabicToRoman.keySet() ) {  
            16  
            while(arabic >= arabicValue){  
                17  
                result += arabicToRoman.get(arabicValue);  
                18  
                arabic -= arabicValue;  
            }  
        }  
        19  
        return result;  
    }  
}
```

3. special cases

Symbol Repetition
and Addition

for Confidence

new single Symbols

new Symbols
special case

```
@CsvSource({
    "1, 'I'",
    "2, 'II'",
    "3, 'III'",
    "10, 'X'",
    "20, 'XX'",
    "30, 'XXX'",
    "100, 'C'",
    "200, 'CC'",
    "1000, 'M'",
    "11, 'XI'",
    "1231, 'MCCXXXI'",
    "5, 'V'",
    "6, 'VI'",
    "4, 'IV'",
    "9, 'IX'",
    "19, 'XIX'",
    "50, 'L'",
    "40, 'XL'",
    "49, 'XLIX'",
    "90, 'XC'",
    "99, 'XCIX'",
    "500, 'D'",
    "400, 'CD'",
    "900, 'CM'",
    "846, 'DCCCXLVI'",
    "1999, 'MCMXCIX'",
    "2008, 'MMVIII'",
})

static LinkedHashMap<Integer,String> arabicToRoman = new LinkedHashMap<>();

1 usage new *
public RomanCalculator() {
    arabicToRoman.put(1000,"M");
    arabicToRoman.put(900,"CM");
    arabicToRoman.put(500,"D");
    arabicToRoman.put(400,"CD");
    arabicToRoman.put(100,"C");
    arabicToRoman.put(90,"XC");
    arabicToRoman.put(50,"L");
    arabicToRoman.put(40,"XL");
    arabicToRoman.put(10,"X");
    arabicToRoman.put(9,"IX");
    arabicToRoman.put(5,"V");
    arabicToRoman.put(4,"IV");
    arabicToRoman.put(1,"I");
}

1 usage new *
public String calculate(int arabic) {
    String result = "";
    for ( Integer arabicValue: arabicToRoman.keySet() ) {
        while(arabic >= arabicValue){
            result += arabicToRoman.get(arabicValue);
            arabic -= arabicValue;
        }
    }
    return result;
}
```

3. special cases

The image shows a code editor with two panels. The left panel displays a CSV source file with the following content:

```
@CsvSource({
    "1, 'I'",
    "2, 'II'",
    "3, 'III'",
    "10, 'X'",
    "20, 'XX'",
    "30, 'XXX'",
    "100, 'C'",
    "200, 'CC'",
    "1000, 'M'",
    "11, 'XI'",
    "1231, 'MCCXXXI'",
    "5, 'V'",
    "6, 'VI'",
    "4, 'IV'",
    "9, 'IX'",
    "19, 'XIX'",
    "50, 'L'",
    "40, 'XL'",
    "49, 'XLIX'",
    "90, 'XC'",
    "99, 'XCIX'",
    "500, 'D'",
    "400, 'CD'",
    "900, 'CM'",
    "846, 'DCCCXLVI'",
    "1999, 'MCMXCIX'",
    "2008, 'MMVIII'"
})
```

The right panel shows a Java implementation for Roman numeral conversion:

```
static LinkedHashMap<Integer, String> arabicToRoman = new LinkedHashMap<>();

1 usage new *
public RomanCalculator() {
    arabicToRoman.put(1000, "M");
    arabicToRoman.put(900, "CM");
    arabicToRoman.put(500, "D");
    arabicToRoman.put(400, "CD");
    arabicToRoman.put(100, "C");
    arabicToRoman.put(90, "XC");
    arabicToRoman.put(50, "L");
    arabicToRoman.put(40, "XL");
    arabicToRoman.put(10, "X");
    arabicToRoman.put(9, "IX");
    arabicToRoman.put(5, "V");
    arabicToRoman.put(4, "IV");
    arabicToRoman.put(1, "I");
}

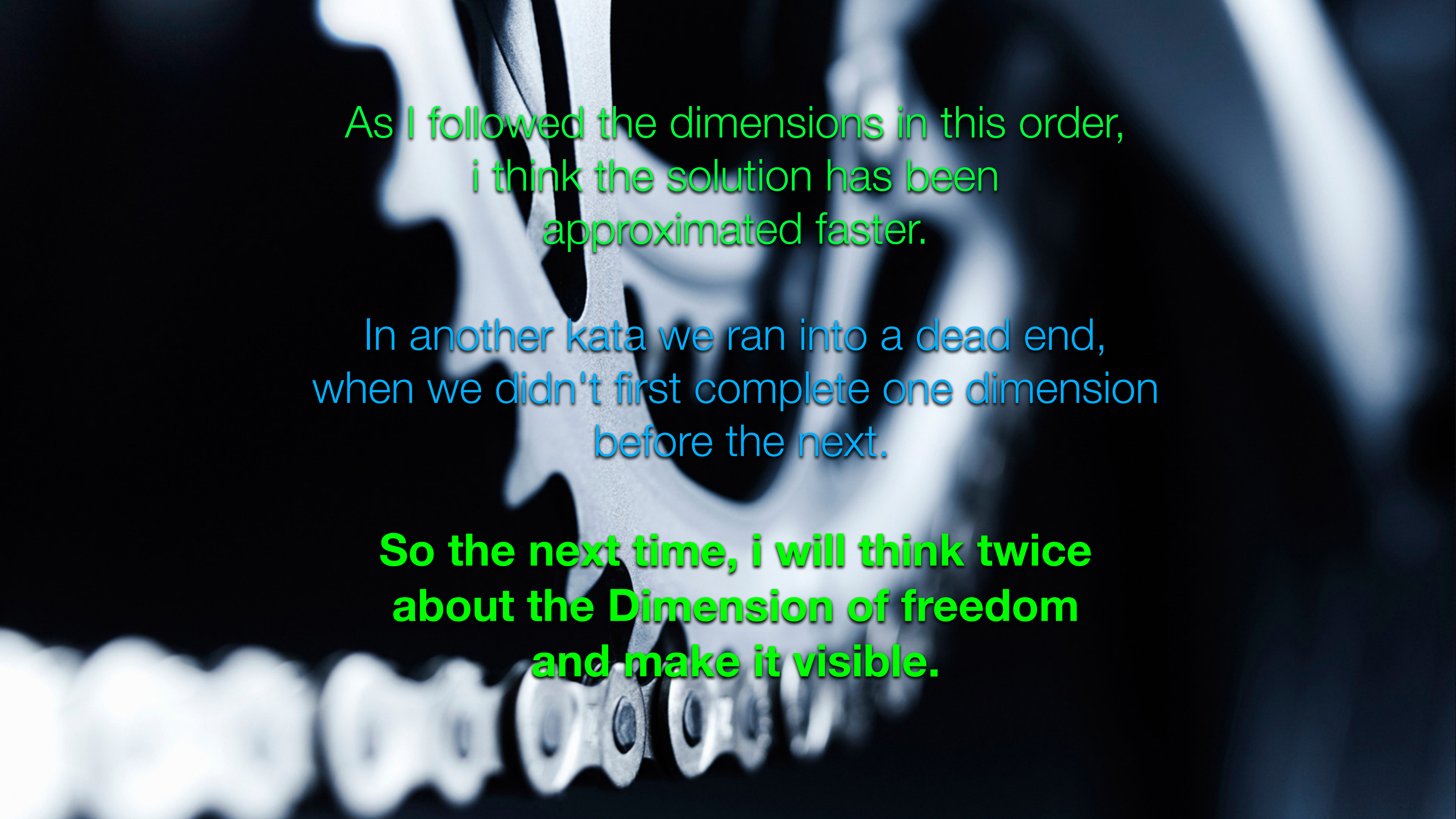
1 usage new *
public String calculate(int arabic) {
    String result = "";
    for (Integer arabicValue: arabicToRoman.keySet()) {
        while(arabic >= arabicValue){
            result += arabicToRoman.get(arabicValue);
            arabic -= arabicValue;
        }
    }
    return result;
}
```

Annotations on the left side of the CSV source file explain special cases:

- Symbol Repetition and Addition: Points to "1, 'I'", "2, 'II'", "3, 'III'", "10, 'X'", "20, 'XX'", "30, 'XXX'", "100, 'C'", "200, 'CC'", "1000, 'M'".
- for Confidence: Points to "11, 'XI'", "1231, 'MCCXXXI'", "19, 'XIX'", "49, 'XLIX'", "99, 'XCIX'".
- new single Symbols: Points to "5, 'V'", "6, 'VI'", "4, 'IV'", "9, 'IX'", "50, 'L'", "40, 'XL'", "49, 'XLIX'", "90, 'XC'", "99, 'XCIX'".
- new Symbols special case: Points to "500, 'D'", "400, 'CD'", "900, 'CM'", "846, 'DCCCXLVI'", "1999, 'MCMXCIX'", "2008, 'MMVIII'".

I've also mixed the Dimension of special cases:

- Single Symbols, which are not used multiplicative (V,L,...)
- Values which are subtraktive used (IV,IX,XC,...)



As I followed the dimensions in this order,
i think the solution has been
approximated faster.

In another kata we ran into a dead end,
when we didn't first complete one dimension
before the next.

**So the next time, i will think twice
about the Dimension of freedom
and make it visible.**

Questions ?

Appendix

Transformation Priority Premise - What is "Obvious implementation" ?

#	TRANSFORMATION	STARTING CODE	FINAL CODE
1	<code>{}</code> => <code>nil</code>		<code>return nil</code>
2	<code>nil</code> => <code>constant</code>	<code>return nil</code>	<code>return "1"</code>
3	<code>constant</code> => <code>constant+</code>	<code>return "1"</code>	<code>return "1" + "2"</code>
4	<code>constant</code> => <code>scalar</code>	<code>return "1" + "2"</code>	<code>return argument</code>
5	<code>statement</code> => <code>statements</code>	<code>return argument</code>	<code>return arguments</code>
6	<code>unconditional</code> => <code>conditional</code>	<code>return arguments</code>	<code>if(condition) return arguments</code>
7	<code>scalar</code> => <code>array</code>	<code>dog</code>	<code>[dog, cat]</code>
8	<code>array</code> => <code>container</code>	<code>[dog, cat]</code>	<code>{dog = "DOG", cat = "CAT"}</code>
9	<code>statement</code> => <code>recursion</code>	<code>a + b</code>	<code>a + recursion</code>
10	<code>conditional</code> => <code>loop</code>	<code>if(condition)</code>	<code>while(condition)</code>
11	<code>recursion</code> => <code>tail recursion</code>	<code>a + recursion</code>	<code>recursion</code>
12	<code>expression</code> => <code>function</code>	<code>today - birthday</code>	<code>CalculateAge()</code>
13	<code>variable</code> => <code>mutation</code>	<code>day</code>	<code>var day = 10; day = 11;</code>
14	<code>switch case</code>		

