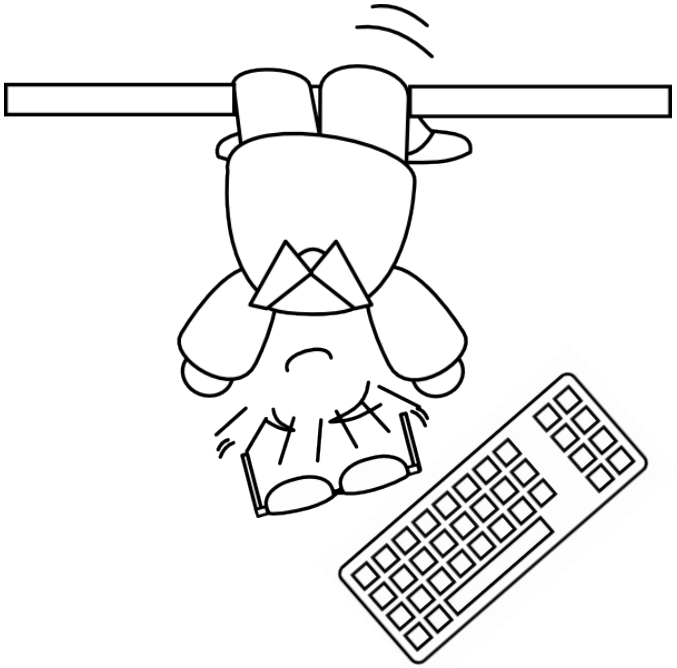
Coding backwards
in order to
to think straight



Initiation to

TDD

(Test Driven Development)

So, TDD ...

... what is that?

It is a practice ...

... a development one

It is a practice …

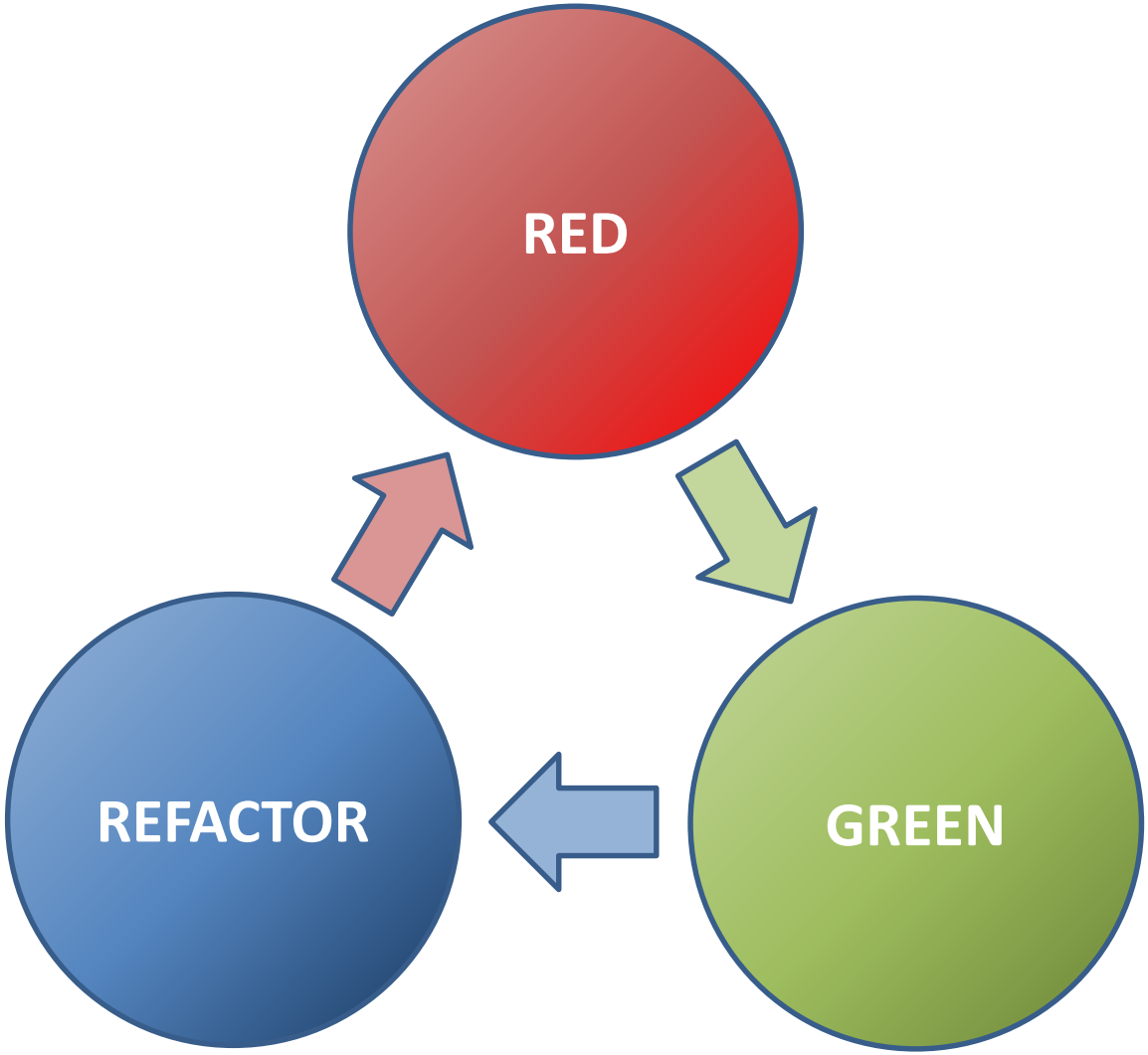
… a development one

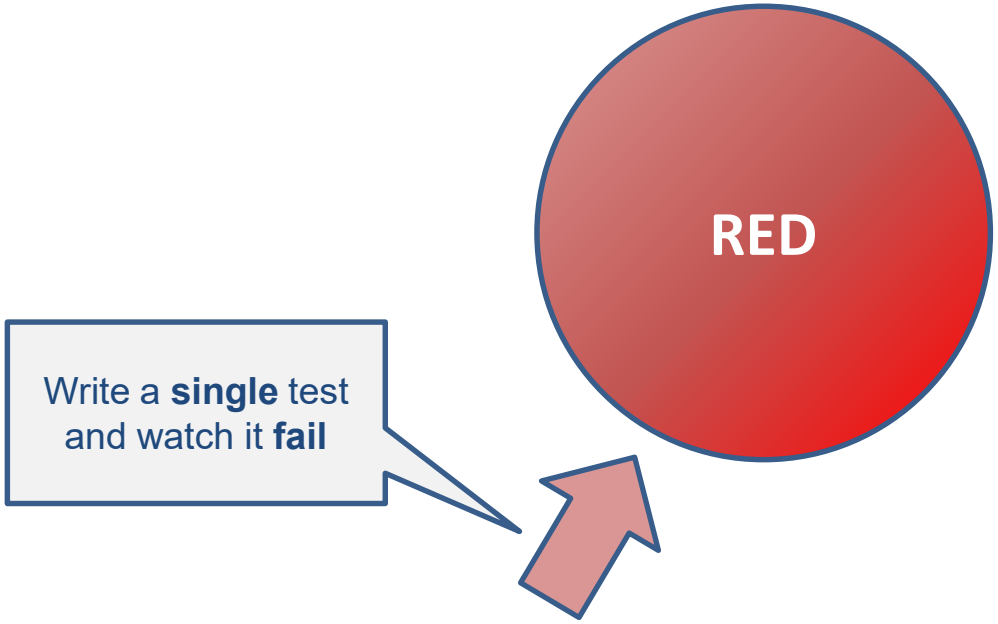
It is not a testing practice

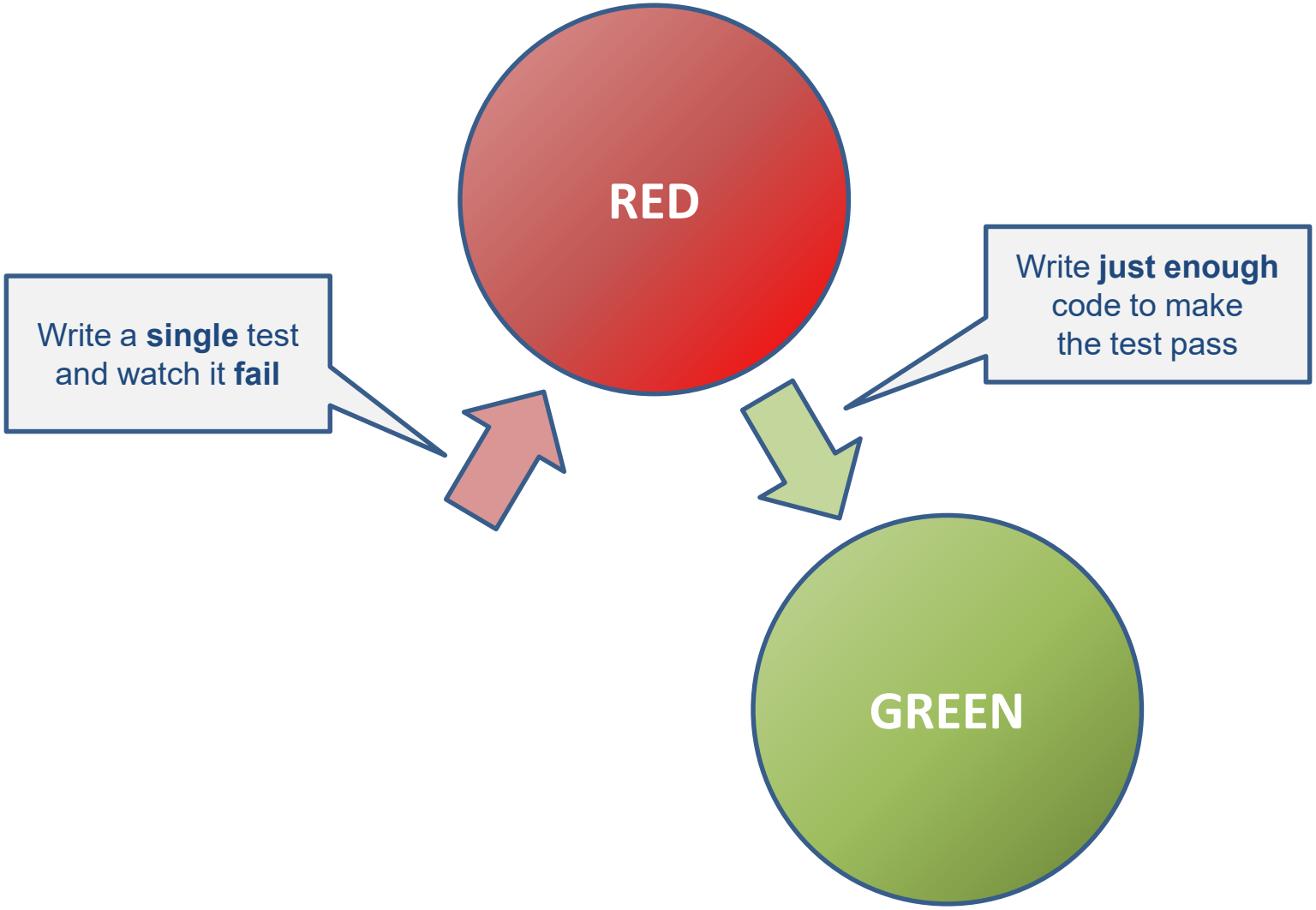
Its purpose is to create production code

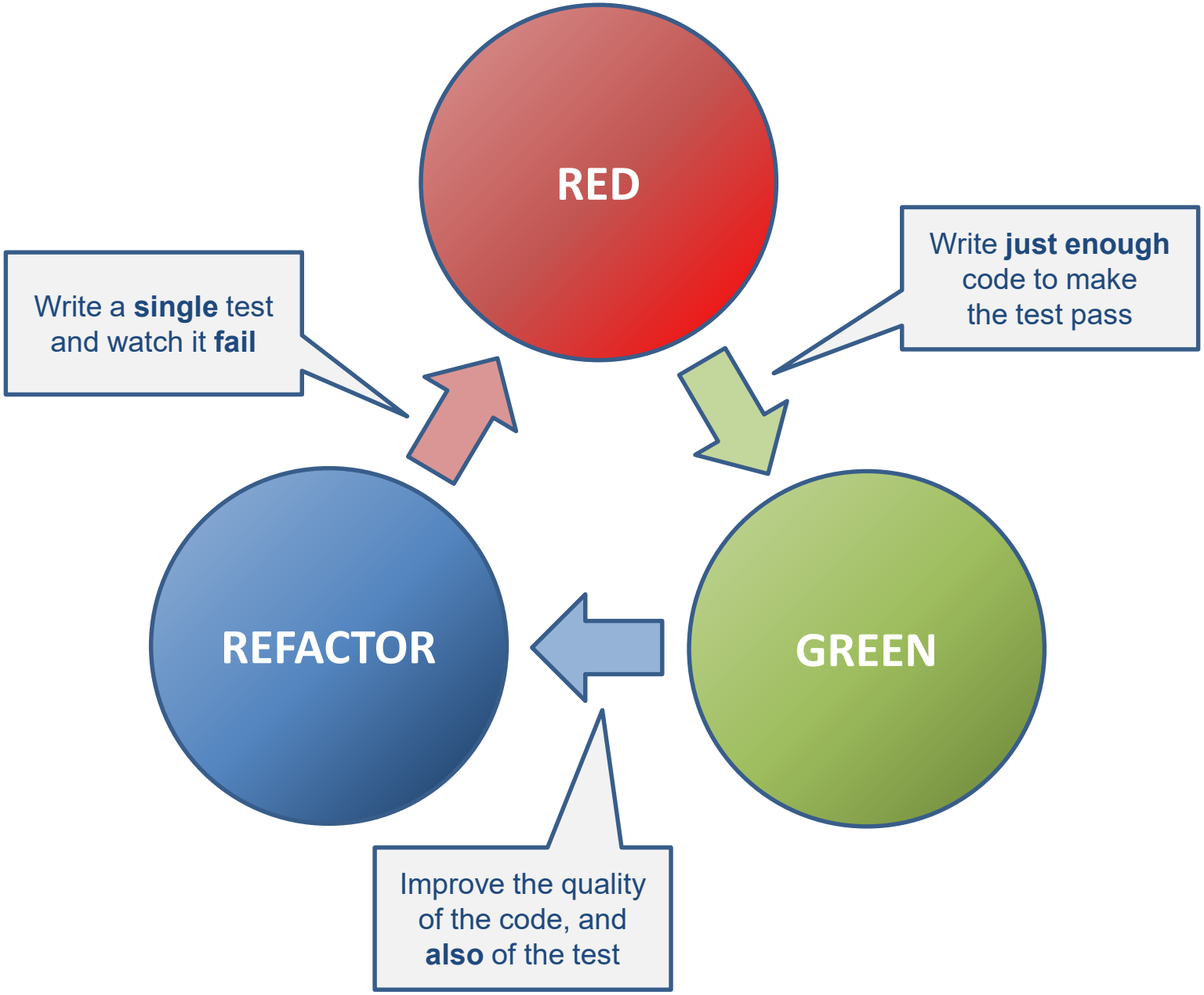
Test coverage is a welcome side-effect

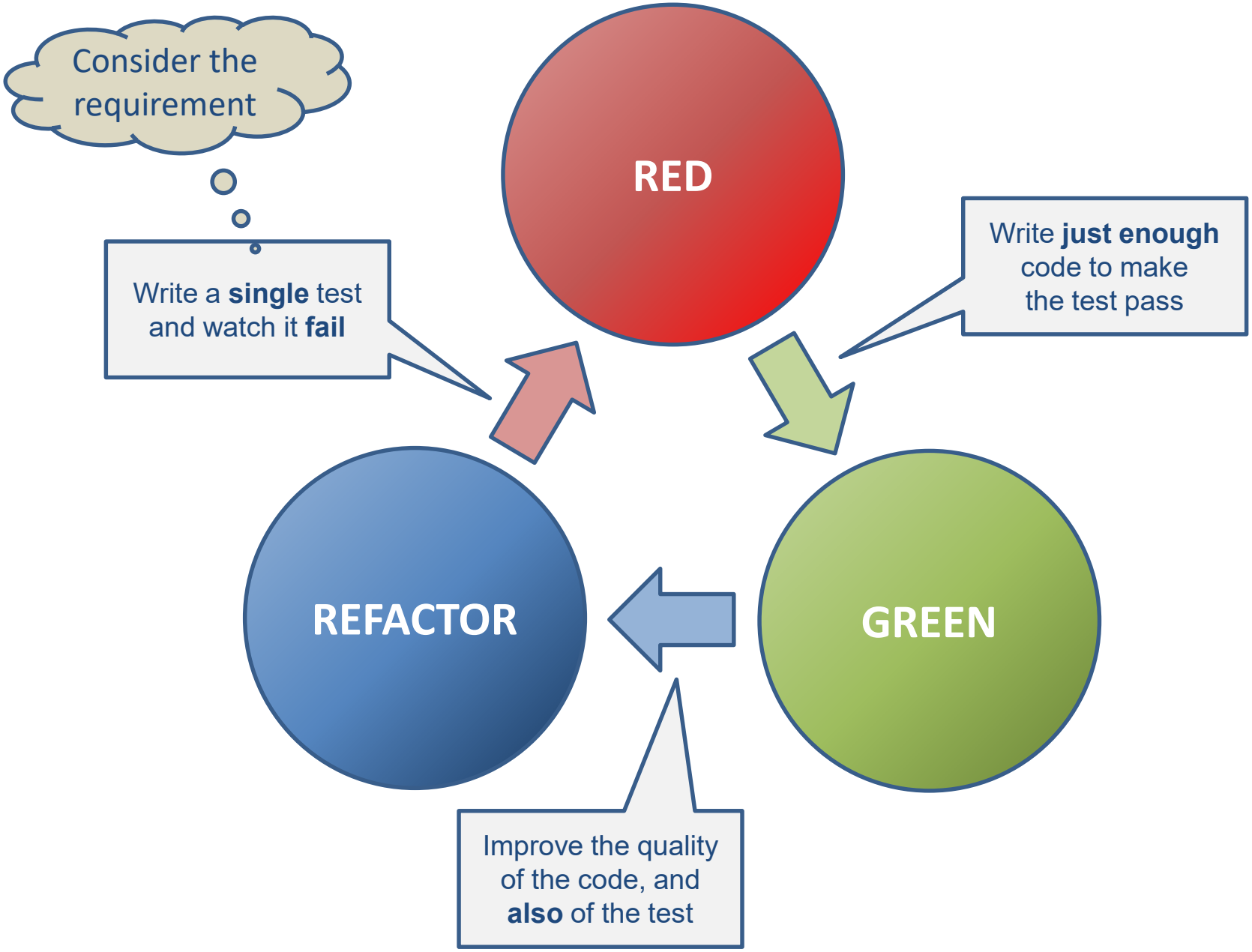
It is only one of XP practices (which are complementary)

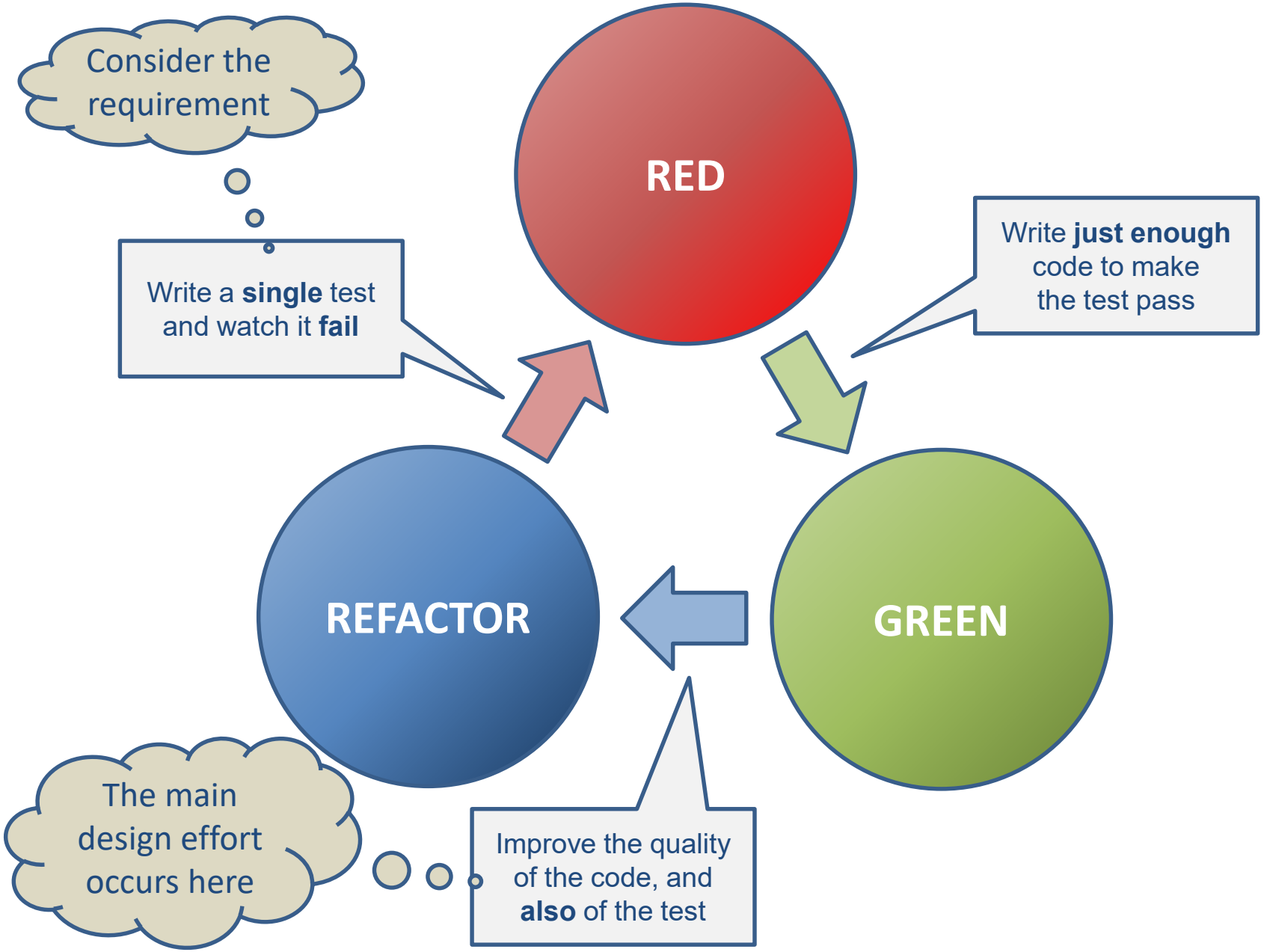

RED
GREEN
REFACTOR


RED
Write a **single** test
and watch it **fail**


RED
Write a **single** test and watch it **fail**
Write **just enough** code to make the test pass
GREEN

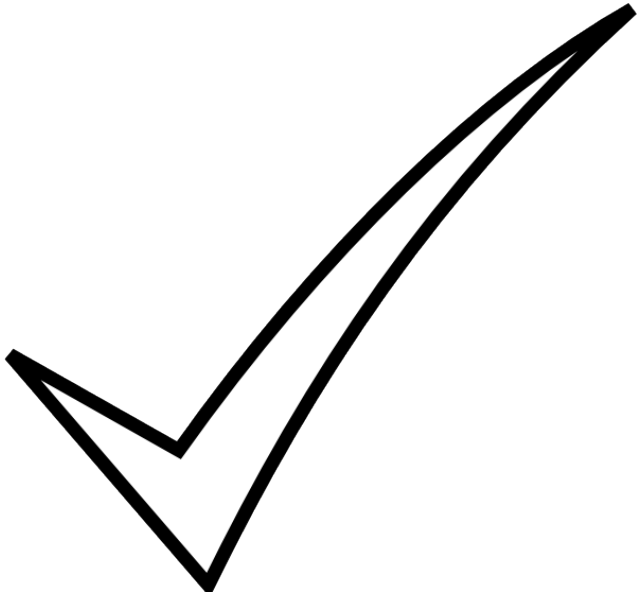

RED
Write **just enough** code to make the test pass
GREEN
Improve the quality of the code, and **also** of the test
REFACTOR
Write a **single** test and watch it **fail**


Consider the requirement
Write a **single** test and watch it **fail**
RED
Write **just enough** code to make the test pass
GREEN
Improve the quality of the code, and **also** of the test
REFACTOR

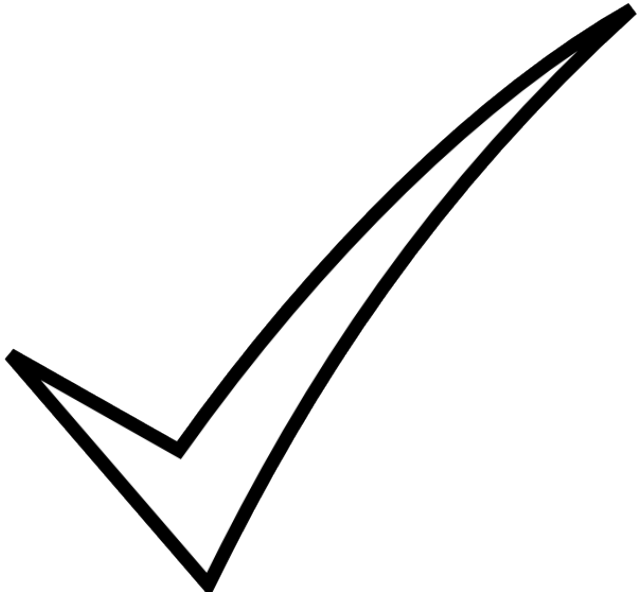

Consider the requirement
Write a **single** test and watch it **fail**
RED
Write **just enough** code to make the test pass
GREEN
REFACTOR
The main design effort occurs here
Improve the quality of the code, and **also** of the test

Okay, but ...

... for what purpose?



Higher Quality



Higher Quality

Users and Business Satisfaction

Product Relialibility and Robusteness

Tolerance to Change

Building Trust and Confidence

Continuous Improvement Cycle

Okay, but ...

... how does TDD help in all this?

Cognitive Biases

Cognitive Biases

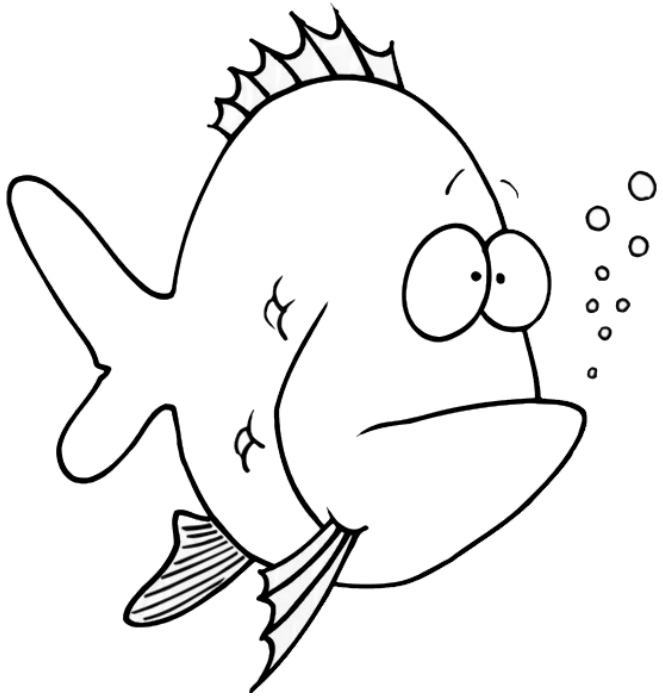
Four main categories of bias:

- **Too much information**
- **Limits of memory**
- **Lack of meaning**
- **Acting in urgency**

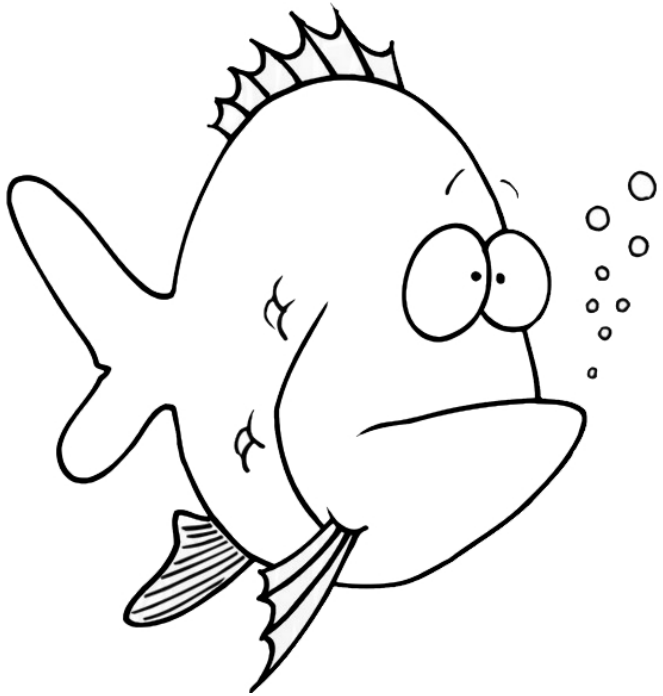
Cognitive Biases

Four main categories of bias:

- Too much information	**Confirmation Bias**
- Limits of memory	**Testing Effect**
- Lack of meaning	**Bandwagon Effect**
- Acting in urgency	**Less-is-better Effect**



Attention Span

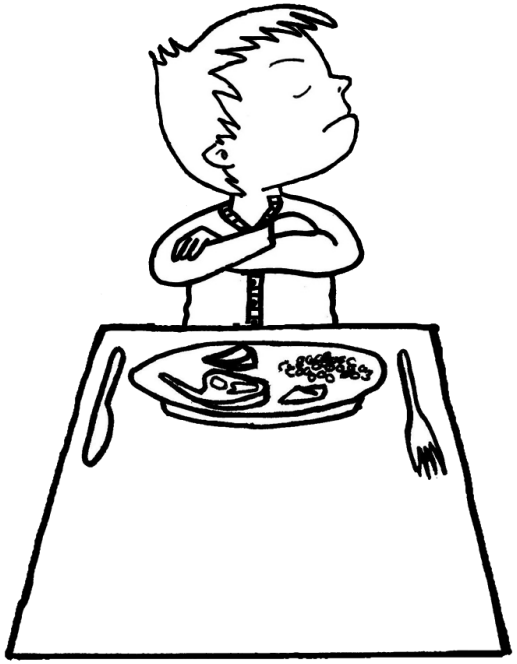


Attention Span

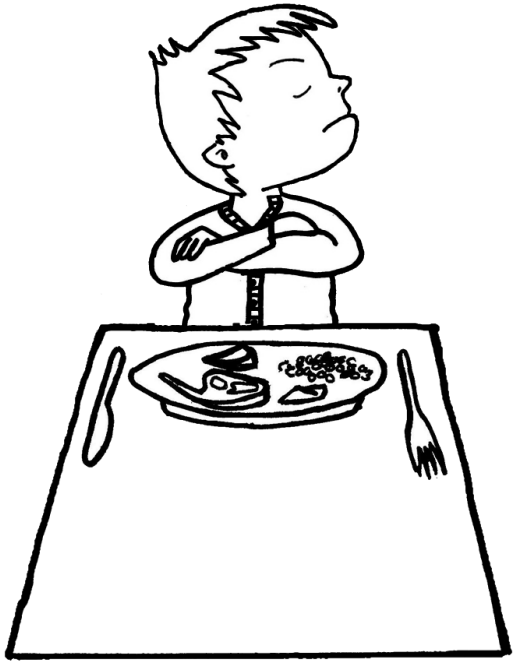
TDD

Framework to focus on

Milestones over time and progress



Operant Conditioning



Operant Conditioning

TDD

« Green » highlights each achievement

« Red » stresses incompleteness

Confidence increases with each cycle

Learning Process

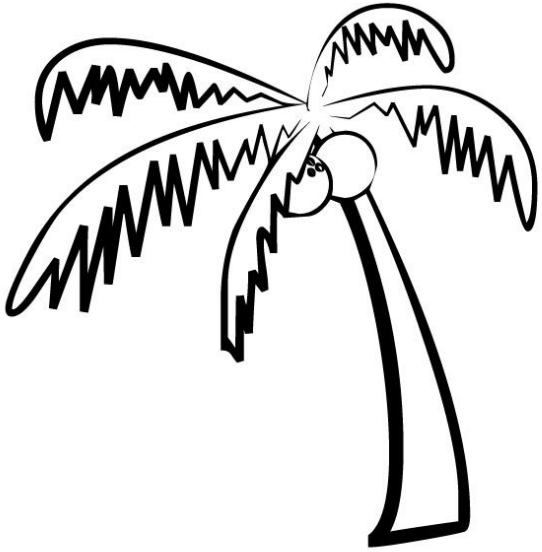
Learning Process

Several combined effects:

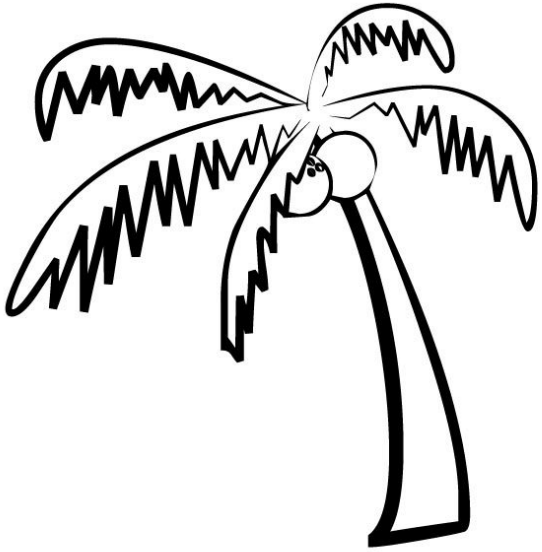
TDD

- **Trial and error**
- **Repetition**
- **Solution emergence**

Continuous Improvment

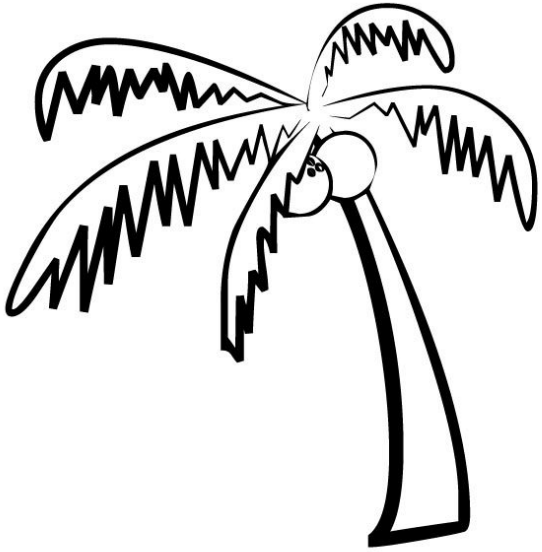


Procrastination



Procrastination

TODO



Procrastination

TDD

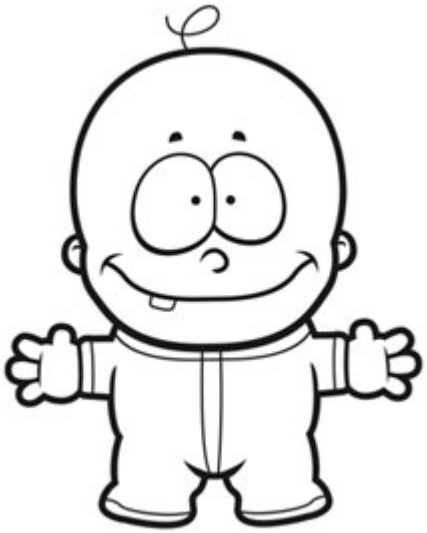
Progress indicator

Iterative process

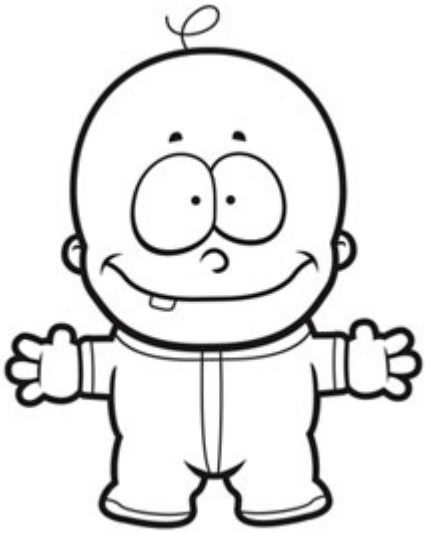
Tests cannot be forgotten or postponed

Okay, but ...

... why is it so uncommon?



An Emerging Profession



An Emerging Profession

Very little hindsight on the different practices

School curricula are evolving… at their own pace

A high overall complexity of the variables involved

Easy from a distance...

... far from being easy

Easy from a distance...

... far from being easy

Changing work habits

Conforming to a constrained cycle

Understanding before coding

It is a practice, not a silver bullet, not a dogma

A « Magnifying » Effect

A « Magnifying » Effect

Colliding with existing code

Discovering problems:

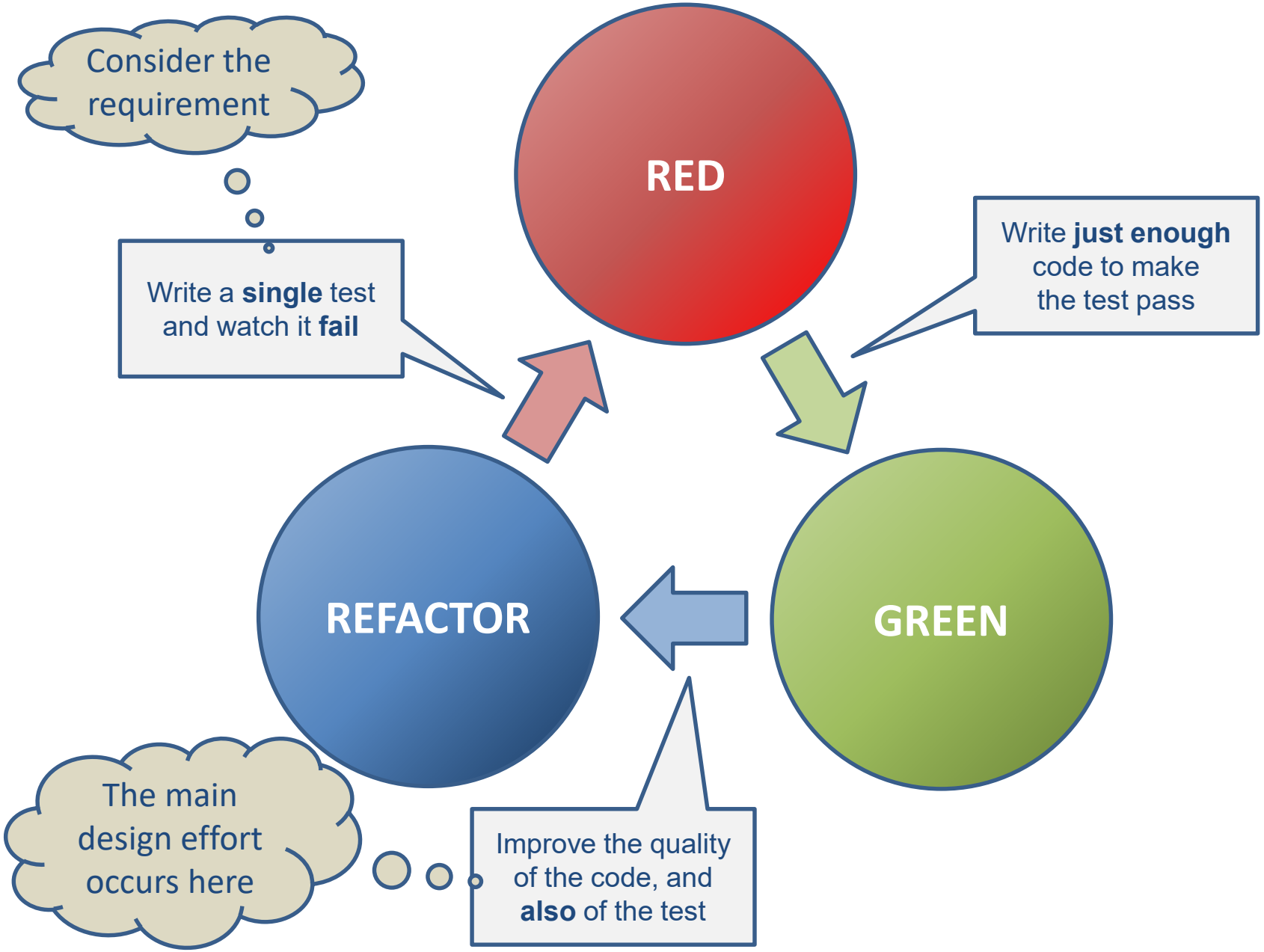
- **design**
- **expression of requirements**
- **organization**
- **...**

➔ **need to accept them and improve ourselves**

Do not blame the practice for what it reveals

Let's get back to the point ...

... TDD is therefore:


Consider the requirement
Write a **single** test and watch it **fail**
RED
Write **just enough** code to make the test pass
GREEN
REFACTOR
The main design effort occurs here
Improve the quality of the code, and **also** of the test

Okay, but speaking of design...

... what is good design?

Three main

Categories of Design

Three main

Categories of Design

Architecture

Macro design

Micro design

Three main

Categories of Design

Architecture = infrastructure, persistence ...

Macro design = packages, dependencies ...

Micro design = business code, rules ...

Three main

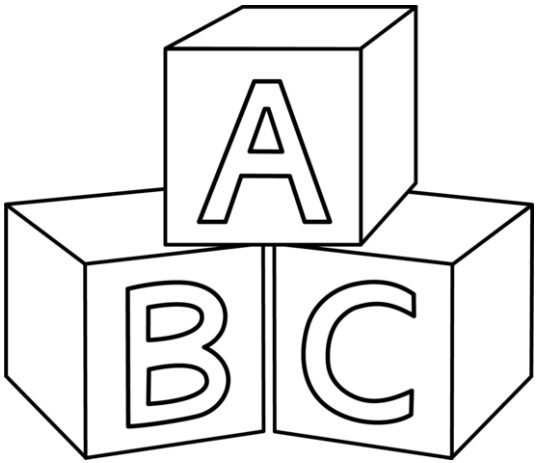
Categories of Design

Architecture = infrastructure, persistence ...

Macro design = packages, dependencies ...

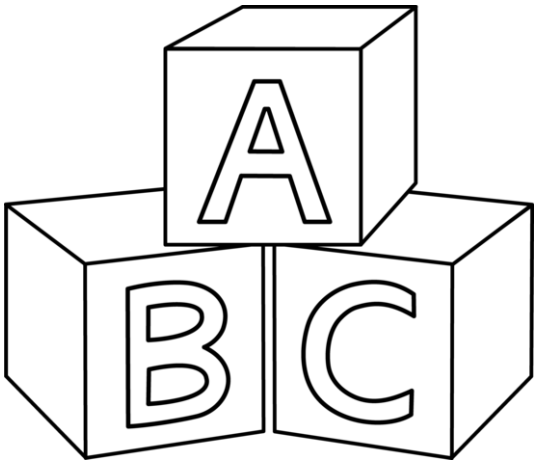
TDD

Micro design = business code, rules ...



Four Rules

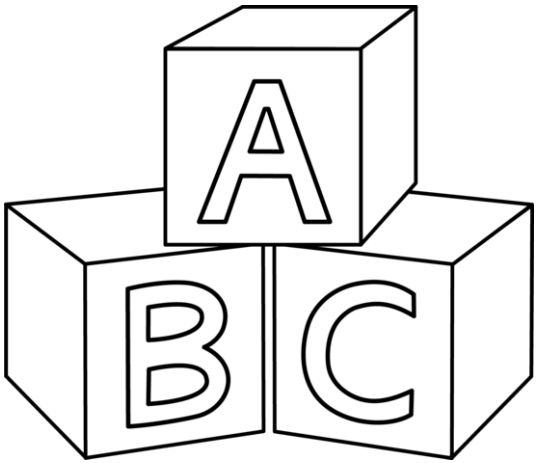
of Simple Design



Four Rules

of Simple Design

- ✓ **Passes the tests**
- ✓ **Reveals intention**
- ✓ **No duplication**
- ✓ **Fewest elements**



Four Rules

of Simple Design

- ✓ **Passes the tests**
- ✓ **Reveals intention**
- ✓ **No duplication**
- ✓ **Fewest elements**

Okay, but ...

... that's not enough!

It's a long way to go

It's a long way to go

Design Patterns

SOLID Principles

Clean Code

Domain Driven Design

...

So ...

... how do we get started ?

Some guidelines

for the TDD cycle

Some guidelines

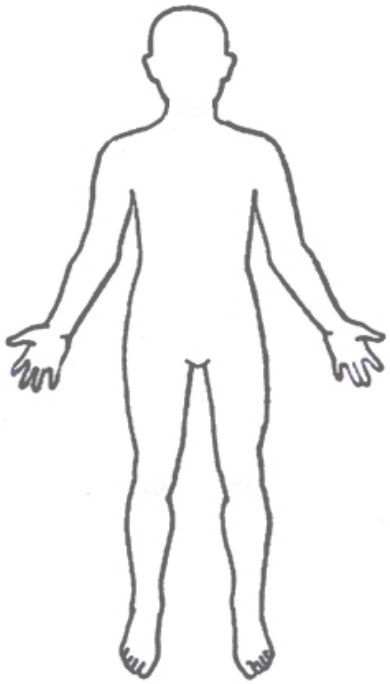
for the TDD cycle

Add one (and only one) new test while in « Green »

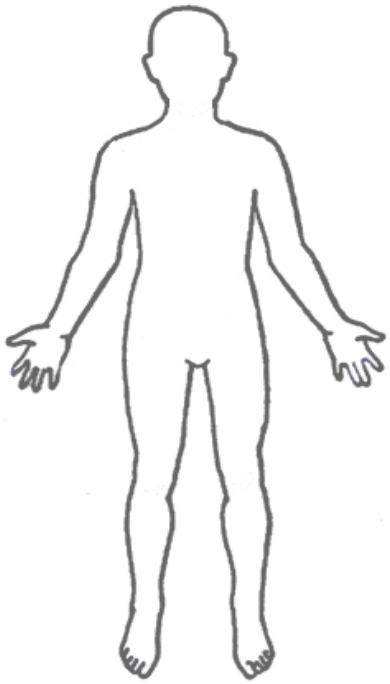
Watch the test fail before coding corresponding solution

Code in order to return as soon as possible to « Green »

Refactor code or test at any one time, not both



Anatomy of a Test

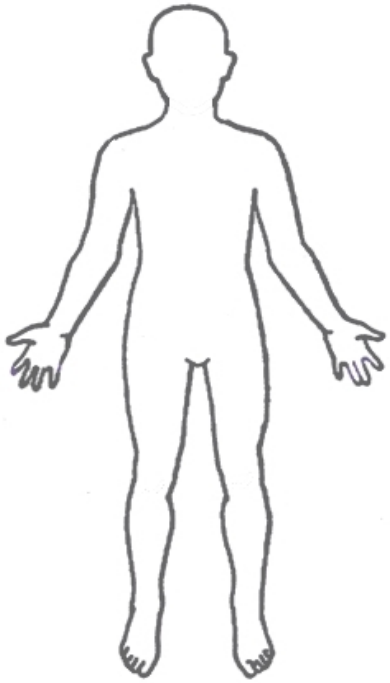


Anatomy of a Test

GIVEN

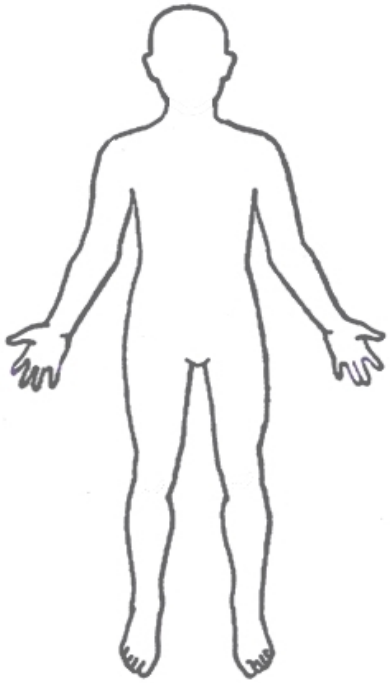
WHEN

THEN



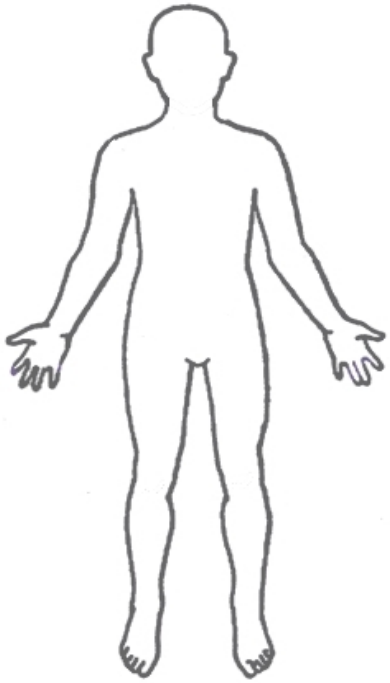
Anatomy of a Test

GIVEN	**Context**
WHEN	**Event**
THEN	**Expectation**



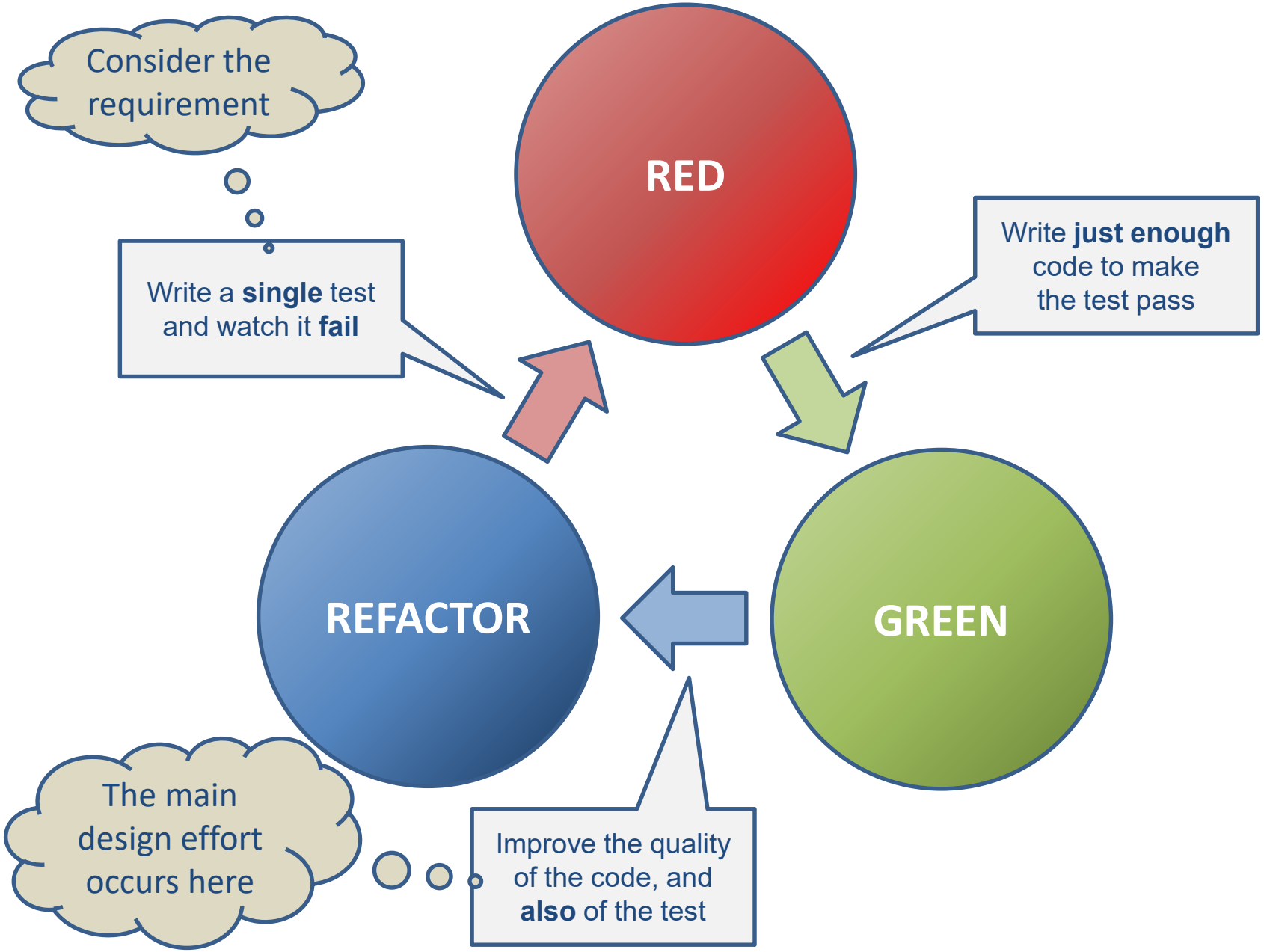
Anatomy of a Test

GIVEN	Context	= states / data
WHEN	Event	= what is being tested
THEN	Expectation	= solution to the requirement



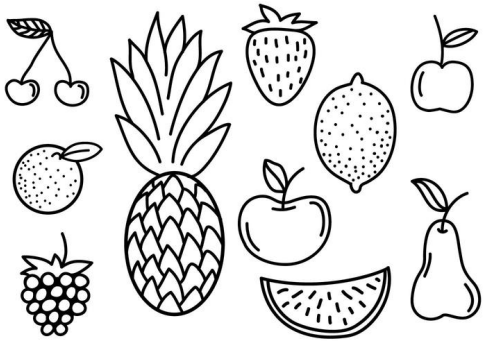
Anatomy of a Test

3	GIVEN	Context	= states / data
2	WHEN	Event	= what is being tested
1	THEN	Expectation	= solution to the requirement


Consider the requirement
Write a **single** test and watch it **fail**
RED
Write **just enough** code to make the test pass
GREEN
REFACTOR
The main design effort occurs here
Improve the quality of the code, and **also** of the test

In the end, the key is ...

... practicing!



Fruit Shop

Charge the right amount when the customer goes to the checkout.

Business rules (prices in cents):

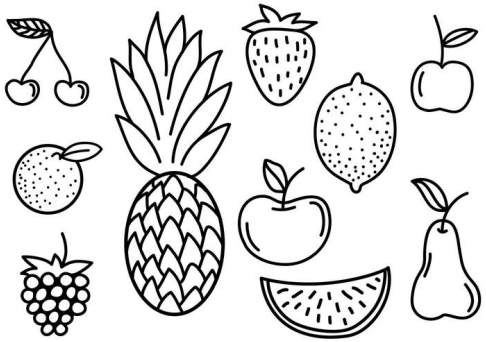
- **1 apple costs 100**
- **1 banana costs 150**
- **1 cherry costs 75**

```
@Test
public void noCheckoutForEmptyCart() {

}
```

```java
@Test
public void noCheckoutForEmptyCart() {

    // THEN
    assertThat(totalAmount).isEqualTo(0);
}
```

```java
@Test
public void noCheckoutForEmptyCart() {

    // WHEN
    int totalAmount = cart.computeTotalAmount();

    // THEN
    assertThat(totalAmount).isEqualTo(0);
}
```

```java
@Test
public void noCheckoutForEmptyCart() {

    // GIVEN
    Cart cart = new Cart();

    // WHEN
    int totalAmount = cart.computeTotalAmount();

    // THEN
    assertThat(totalAmount).isEqualTo(0);
}
```

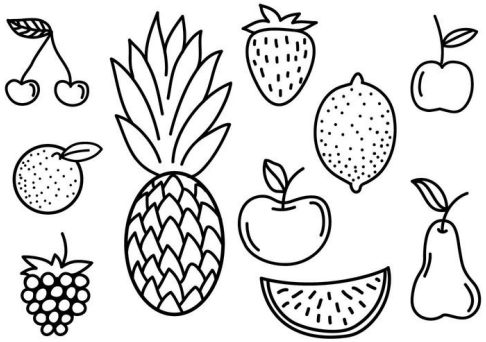


Fruit Shop

Charge the right amount when the customer goes to the checkout.

Business rules (prices in cents):

- **1 apple costs 100**
- **1 banana costs 150**
- **1 cherry costs 75**

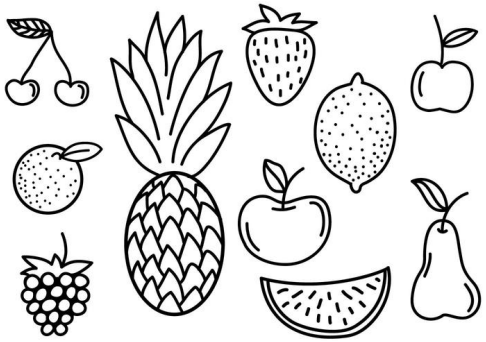


Fruit Shop

Charge the right amount when the customer goes to the checkout.

Business rules (prices in cents):

- **1 apple costs 100**
- **1 free apple when two bought apples**
- **1 banana costs 150**
- **1 cherry costs 75**

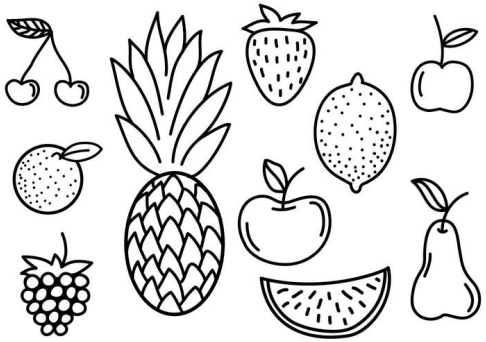


Fruit Shop

Charge the right amount when the customer goes to the checkout.

Business rules (prices in cents):

- **1 apple costs 100**
- **1 free apple when two bought apples**
- **1 banana costs 150**
- **1 cherry costs 75**
- **You have to be able to sell « des pommes » in France**

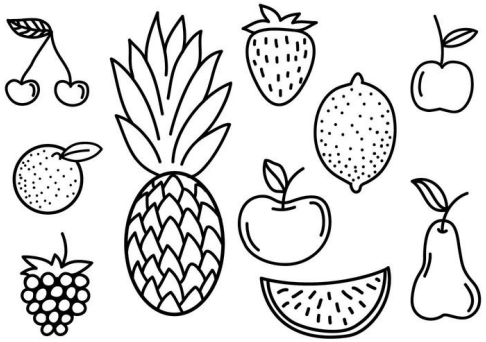


Fruit Shop

Charge the right amount when the customer goes to the checkout.

Business rules (prices in cents):

- **1 apple costs 100**
- **1 free apple when two bought apples**
- **1 banana costs 150**
- **The second banana is half price**
- **1 cherry costs 75**



Fruit Shop

Charge the right amount when the customer goes to the checkout.

Business rules (prices in cents):

- **1 apple costs 100**
- **1 free apple when two bought apples**
- **1 banana costs 150**
- **The second banana is half price**
- **1 cherry costs 75**
- **A loyalty program customer is entitled to a 10% discount**


THANK
YOU