



@codecentric

## Test Driving Your Domain Core

DDD Europe, 05.02.2020

Tobias Goeschel  @w3ltraumpirat

Please download the workshop test sources:

[https://github.com/weltraumpirat/ws\\_test\\_driving\\_your\\_domain\\_core\\_java](https://github.com/weltraumpirat/ws_test_driving_your_domain_core_java)

[https://github.com/weltraumpirat/ws\\_test\\_driving\\_your\\_domain\\_core\\_is](https://github.com/weltraumpirat/ws_test_driving_your_domain_core_is)

(Photo by Mirko Reinhardt)



# First of all: It's not about the tests.

*The objective of this workshop is to **gradually evolve the design of a domain core**.*

*This means we will start small, then grow. We will try things, then find out where we're wrong. And we will probably rewrite some stuff.*

*That's **OK**.*

## TDD Steps:

1. *Think*
2. *Add a **failing** unit test*
3. *Seriously, **see** it **fail***
4. *Write **just enough** code to make it **pass***
5. ***Refactor** towards better understanding / clearer code*
6. *Repeat*

## Inside-Out

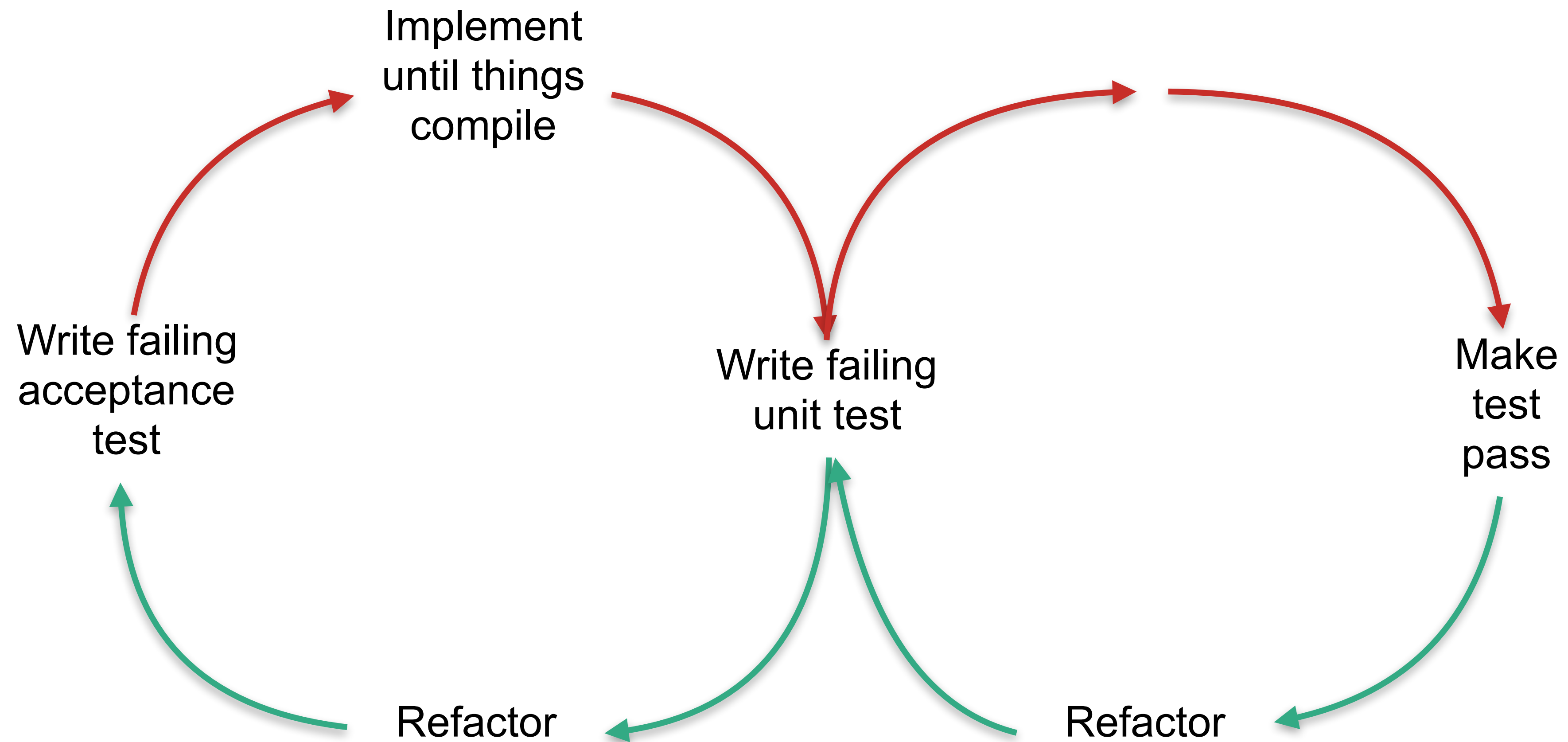
- *Start with the null / zero - case*
- *Add logic for processing a single item*
- *Add iterations / recurse*
- *Design **emerges***
- *“What’s the smallest test I can write next?”*

## Outside-In

- *Decide on a design **first***
- *Run tests through the API*
- *Mock collaborators*

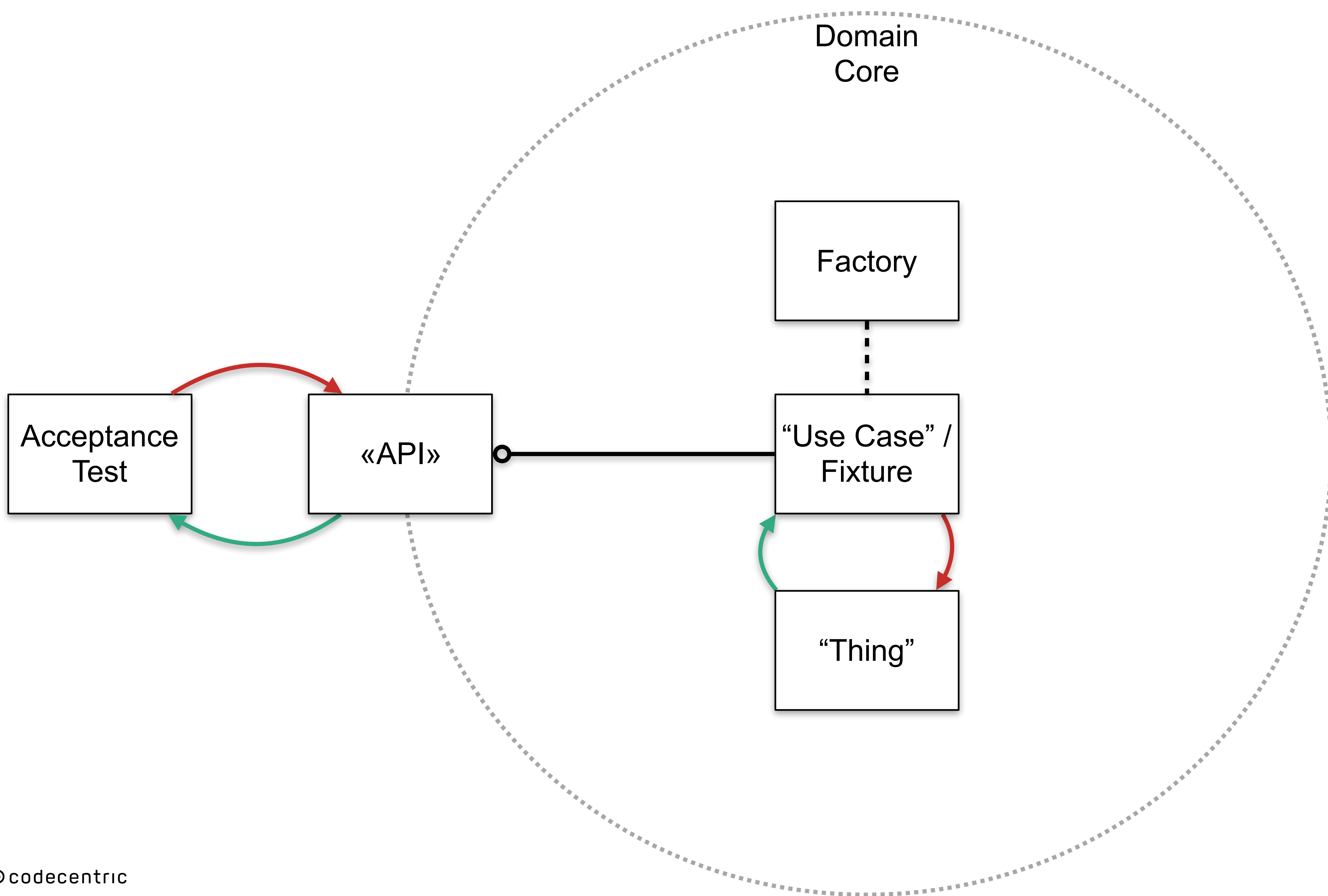
## The Double Loop of BDD + TDD

- Write a **failing** acceptance test
- Implement steps until things **compile**
- Switch to **TDD** mode
- Periodically run all tests until **green**
- **Refactor** fixture steps, as needed



## Exercise #1:

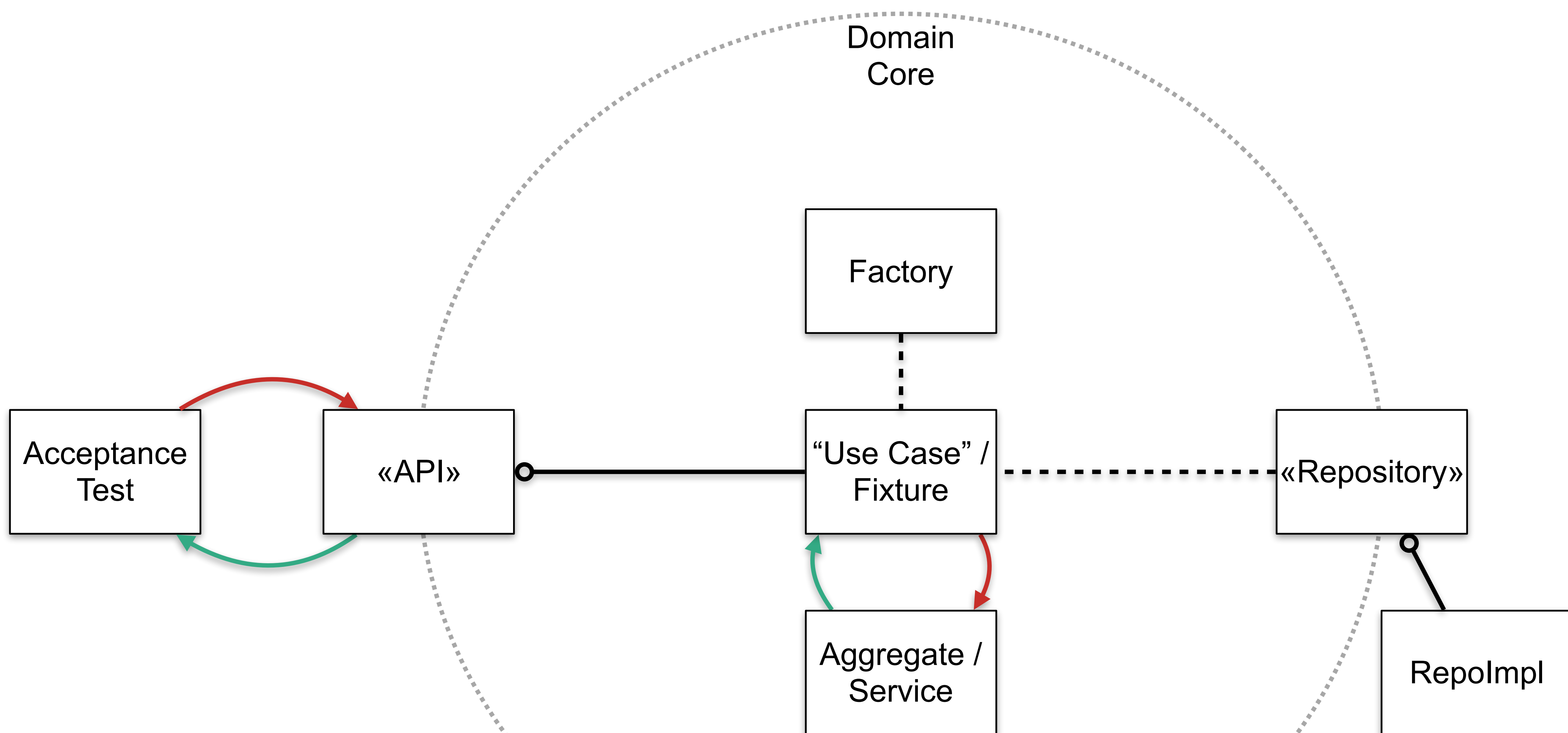
- *Choose one: Outside-in / Inside-out*
- *Start from BDD test*
- *No `new` - use factories!*
- *`CheckoutService` or `cart.checkOut()`?*
- *Where do things “live”? What can tests “see”?*
  - Is there an explicit API?
  - We probably need enclosing “Use Cases” / Fixtures to hold objects





## Exercise #2: Aggregate and Repository

- *We want persistence, but at little cost:  
Use In-Memory Repositories (HashMap)*
- *Pass **only** immutable objects*

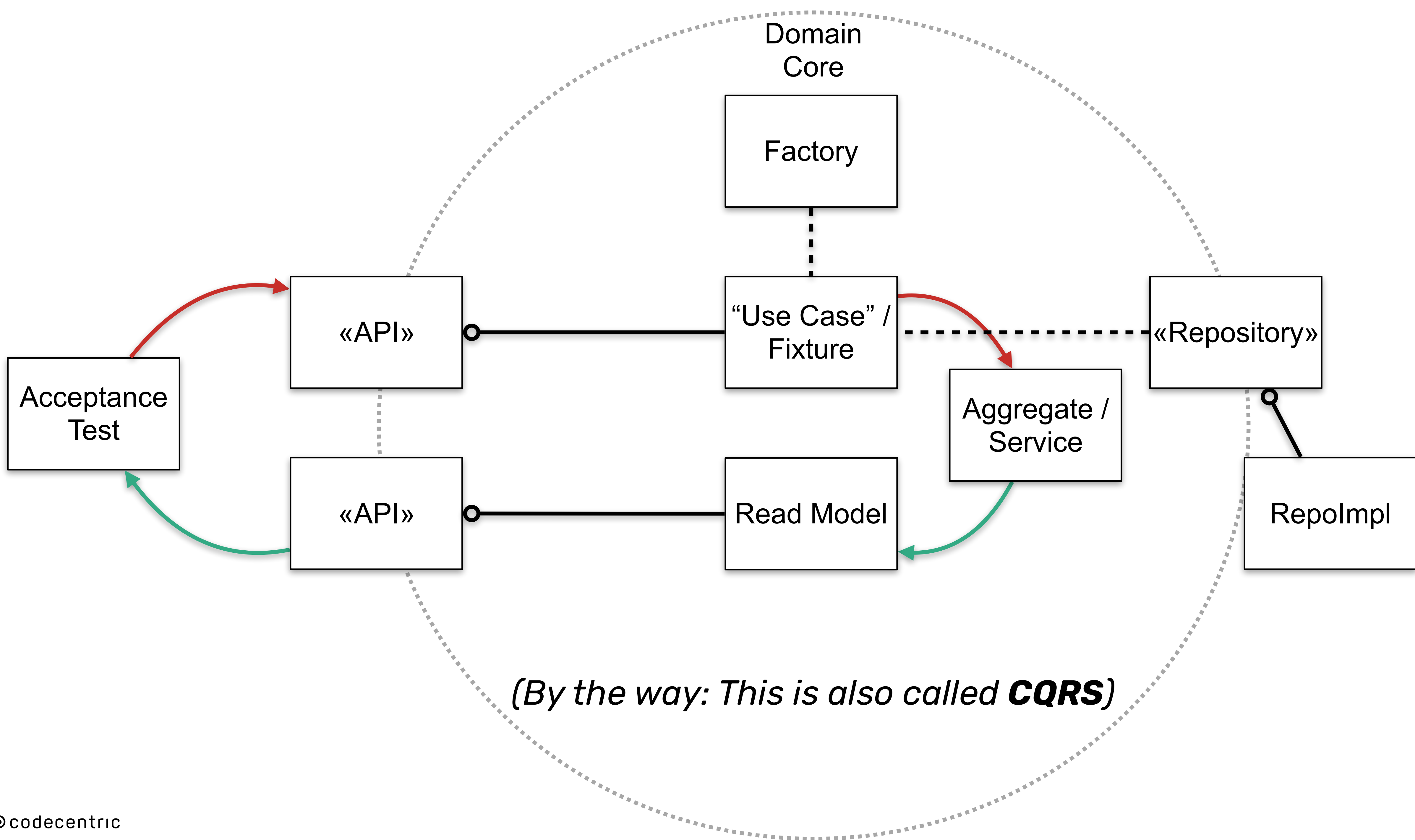


*(By the way: This is also called **Hexagonal Architecture**)*

## Exercise #3: Tell, Don't Ask

- **No** getters
- **No** return values
- *How do we output things?*
- ***Outside-in*** is a good match. Why?





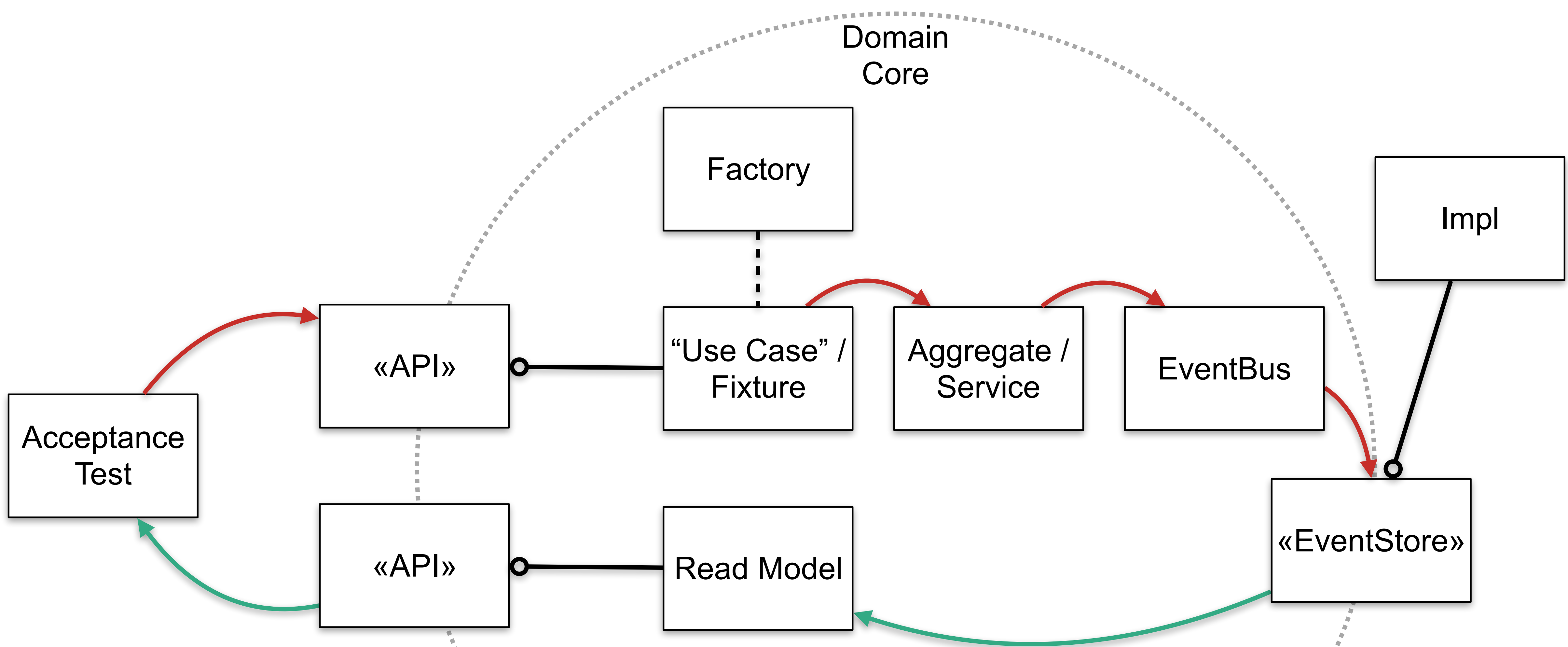
## A word or two about test doubles:

- Dummy, Stub, Spy, Mock - *learn the differences!*
- *Setup is often extensive*
  - *Sometimes it's easier to just extend / create dummy classes*
  - *Frameworks should be chosen carefully, and used sparsely*
- *Mocks couple your test to the implementation*
  - *Tests become more brittle*
  - *Refactoring becomes harder*
  - *Rule of thumb: Use mocks **only** at system boundaries*

## Exercise #4: Eventbus

- *“Eventbus” translates to **Publish/Subscribe***
- ***Only** aggregates may publish events*
- *What happens to our tests?*
- ***Advanced:** Replace repositories with “**event store**”*
  - *That’s really just a list of past events*
  - *Create / update read models via **projections***







*(By the way: This is also called **Event Sourcing**)*

# Closing Round



 codecentric AG  
Kreuznacher Str. 30  
60846 Frankfurt / Germany

 Tobias Goeschel  
Principal Consultant  
[tobias.goeschel@codecentric.de](mailto:tobias.goeschel@codecentric.de)  
[www.codecentric.de](http://www.codecentric.de)

 Phone: +49 (0) 151.14965469

