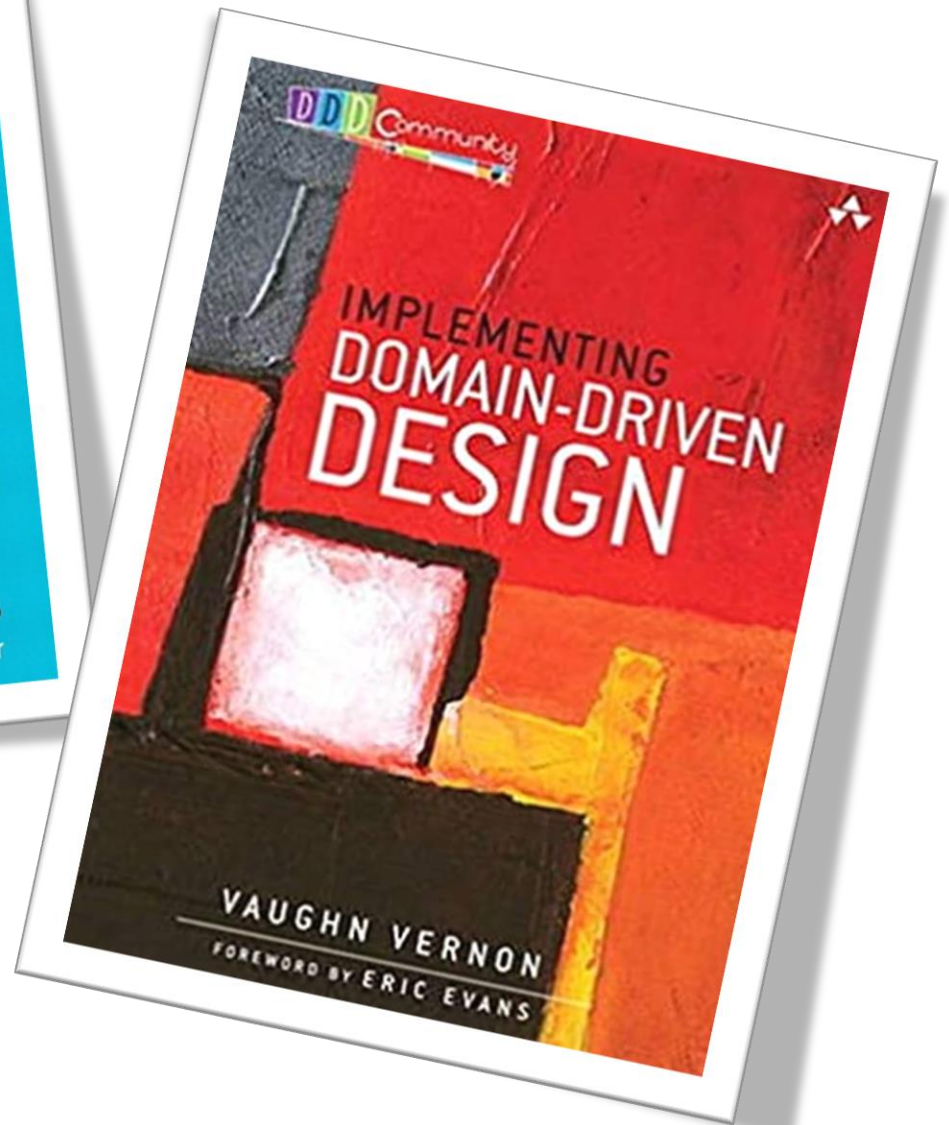
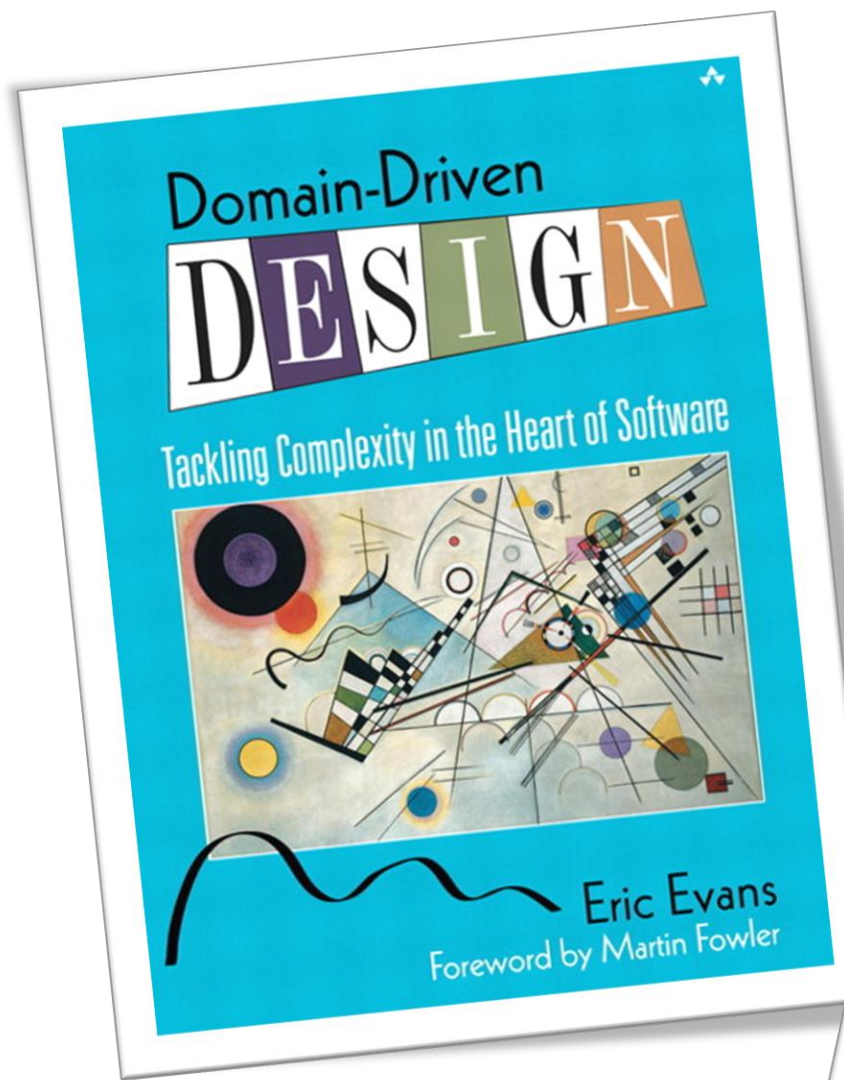


EAPLI

# Introdução DDD

Paulo Gandra de Sousa  
pag@isep.ipp.pt



**Table 1.4. Analyzing the Best Model for the Business**

*Which is better for the business?*

*Though the second and third statements are similar, how should the code be designed?*

Possible Viewpoints

Resulting Code

*“Who cares? Just code it up.”*

Um, not even close.

```
patient.setShotType(ShotTypes.TYPE_FLU);  
patient.setDose(dose);  
patient.setNurse(nurse);
```

*“We give flu shots to patients.”*

Better, but misses some  
important concepts.

```
patient.giveFluShot();
```

*“Nurses administer flu vaccines  
to patients in standard doses.”*

This seems like what we’d like  
to run with at this time, at least  
until we learn more.

```
Vaccine vaccine = vaccines.standardAdultFluDose();  
  
nurse.administerFluVaccine(patient, vaccine);
```

Persistence decisions  
**must not**  
dictate constraints  
on the  
domain model

# Anemic Domain Model

*Procedural thinking where Objects are just Plain data structures devoid of any behaviour, and behaviour is concentrated in "service" and "controller" classes.*



# Entities

- Objects **with rich behaviour** in the real world which we would like to track its **identity**
- Example:
  - **Student** identified by **Student Number, NIF, email**
  - **Product** identified by **Product Reference**
  - **Sale** identified by **Invoice Number**

# Entity: example

```
Class Product{  
    private ProductID id;  
    // other attributes of product,  
    //e.g., designation, description, et
```

Always  
constructed in a  
valid state

```
    private Product(){  
    public Product(String sku, Money price) {...}
```

```
    public ProductID getProductID() { ... }
```

It's ok to return  
one's identity

```
    private void setProductID(string sku) { ... }
```

No one can change  
one's identity

```
    public boolean equals(Object other) {  
        if (other==this) return true;  
        if (!(other instanceof Product)) return false;  
        return this.getProductID().equals(  
            (Product)other.getProductID());  
    }
```

```
}
```

Object instances refers to the same real world  
entity, if they have the same identity

# Value objects

- Problem
  - Some objects matter for the value of its attributes, e.g., Color
  - Serve to describe, quantify or classify an Entity
- Solution
  - Create **immutable** objects which are identified by the equality of its attributes and do not need an identity



# Value object: example

No setters

Always  
constructed in a  
valid state

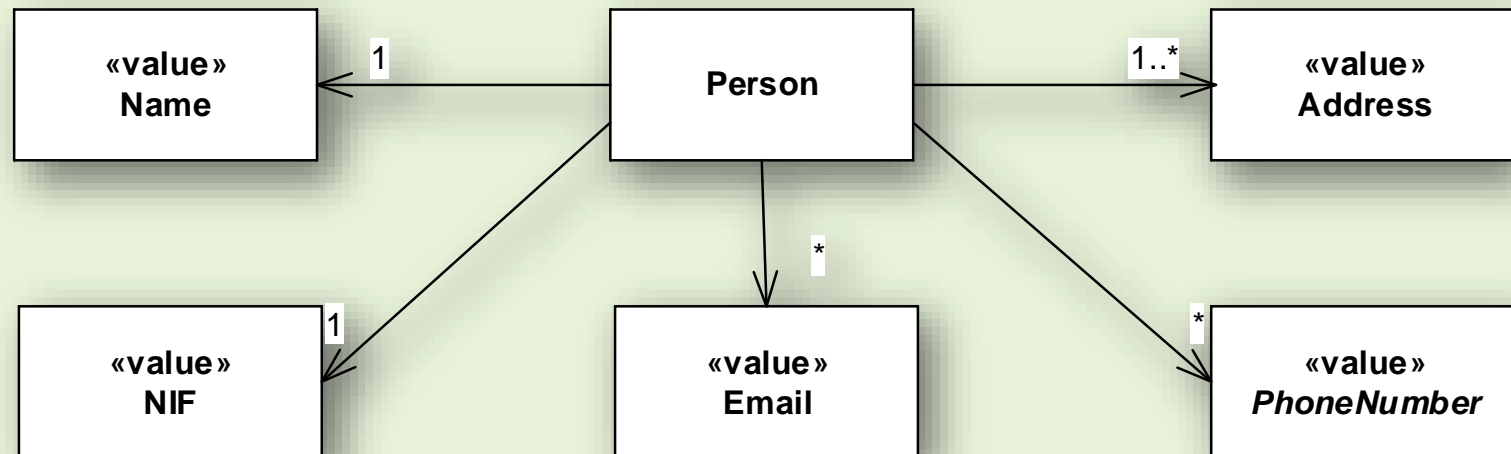
```
class Color {  
    private final int red;  
    private final int green;  
    private final int blue;  
  
    public Color(int r, int g, int b) {...}  
  
    public int redComponent() {...}  
    public int greenComponent() {...}  
    public int blueComponent() {...}  
    public boolean isPrimaryColor() {...}  
  
    // immutable; creates a new object  
    public Color combinedWith(Color other) {...}  
  
    // equality by value  
    public boolean equals(Object other) {...}  
}
```

Getters might be  
ok, but try to avoid  
“get” syntax

# The Domain, SRP and Value Objects

*Primitive types are not the best option to represent domain concepts!*

*Favour imutability of your objects.*



# Value Objects are immutable, but they characterize mutable Entities...

```
// two red cars
Color red = new Color(1, 0, 0);
Car c1 = new Car("AA-10-24", ..., red);
Car c2 = new Car("BB-28-85", ..., red);

assertEquals(c1.color(), red);
assertEquals(c2.color(), red);

// we cannot change "the color" red, but
// we can change the "color of car" c1
Color blue = new Color(0, 0, 1);
c1.repaint(blue);

assertEquals(c1.color(), blue);
assertEquals(c2.color(), red);
```

Notice the name of the methods. No get/set semantics ☺