Pursuing practices of Domain-Driven Design in PHP

Who am I

Giorgio Sironi
Bachelor in Computer Engineering
Advisor @ Allbus
Zone Leader @ DZone

Allbus

The long title

Pursuing
practices of
Domain-Driven
Design
in PHP

The DDD box of goods

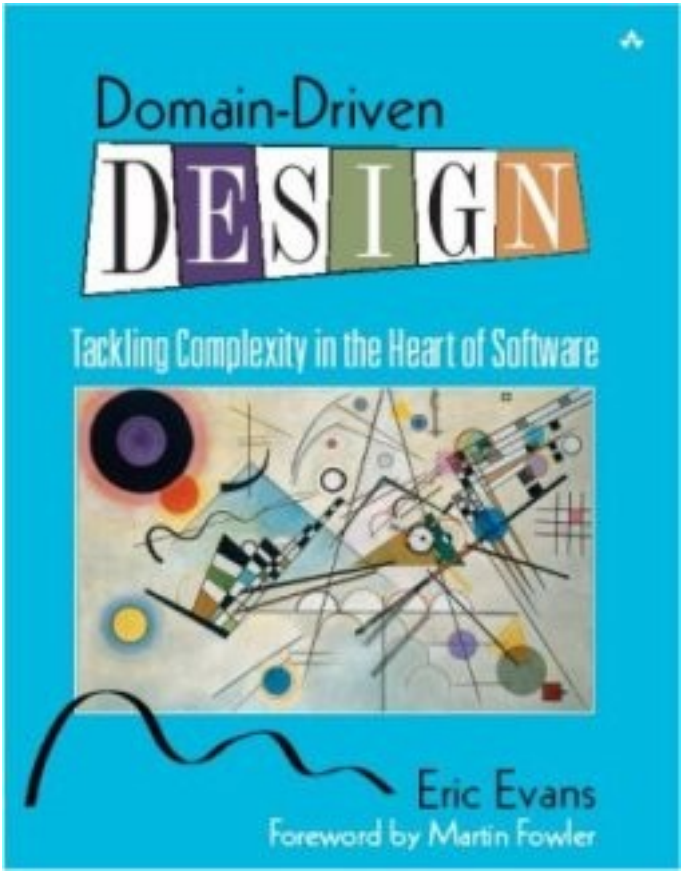
Knowledge crunching

Ubiquitous Language

Domain Model

Refactoring towards deeper insight...

Collaboration patterns (Anti-corruption layer, Separate ways, ...)



Why? (the most important slide)

- Close to business, to follow its changes
  - aids iterative development
- The code is the design
  - supported by blackboards and UML
- Test and develop with in-memory objects
  - no instantiation of Oracle connections
- It's also fun!
  - exercise creativity and learning skills

Domain Model

Reflects knowledge of the domain **more** than technology

- *While there is value in the item on the right, we value the item on the left more*
- Persistence-agnostic (UnitOfWork)
- In general, no outward dependencies

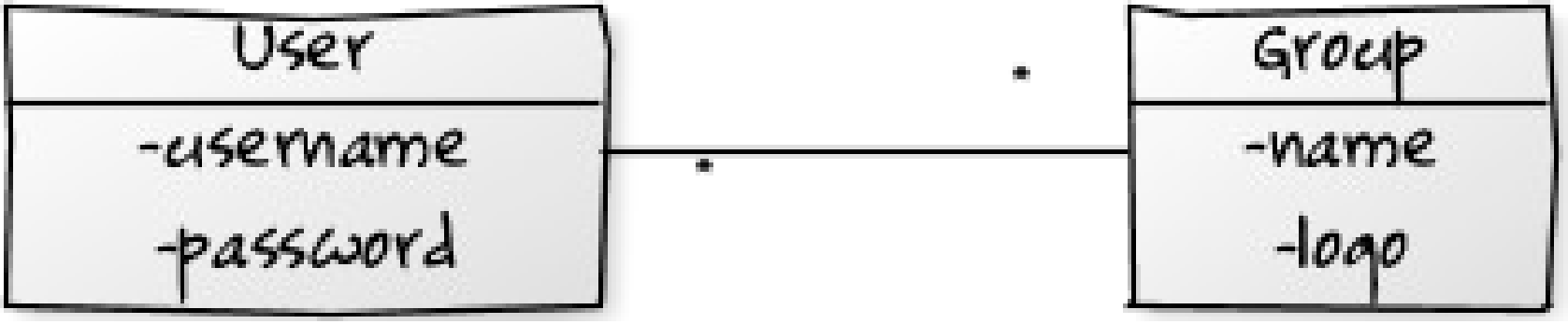
Step 1: from relational...

user
id
username
password
active

group
id
name
logo

user_group
id_user
id_group

Step 1: ...to object-oriented




```
class User
{
}
```

```
class Group
{
}
```

Step 2: from Active Record...

```
class Group extends Doctrine_Record
{
}
```

Hard dependency towards ORM

Inherits pollution from Doctrine_Record

Step 2: ... to Data Mapper

```
/**
 * @Entity
 */
class User
{
}
```

Building blocks

Entity

Value Object

Aggregate

Repository

Factory

Service

Other transient objects (Specification, Parameter Objects, ...)

Entity

More than a row

Equality is based on identity

e.g. Post #42, user 'giorgiosironi', …

The bread and butter of your Domain Model

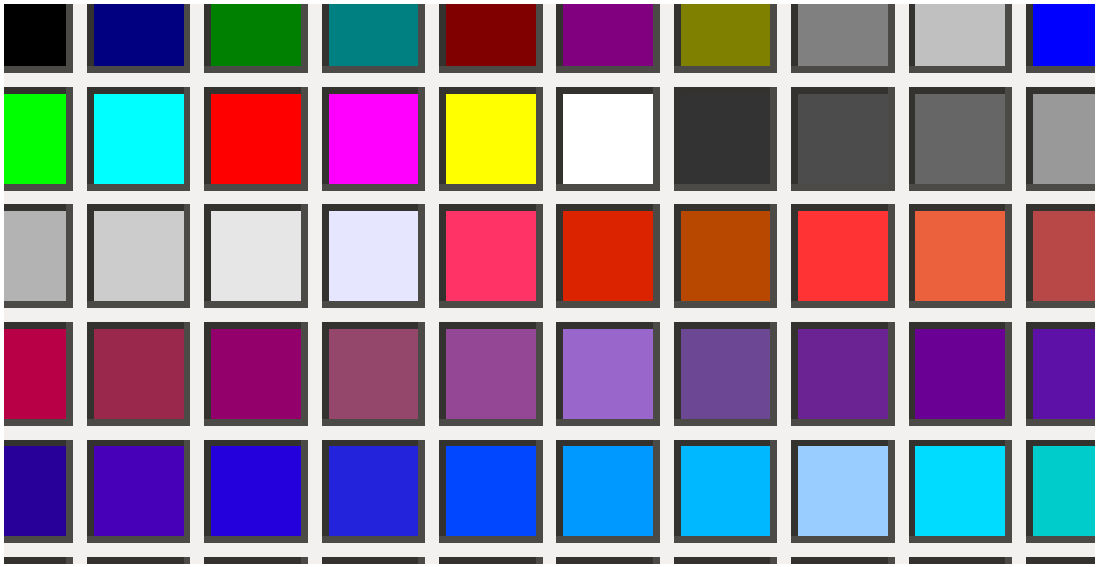
Value Object

Previously known in the world as *value*

e.g. the number 42, Zip code 22031, #FF0000

Equality based on values

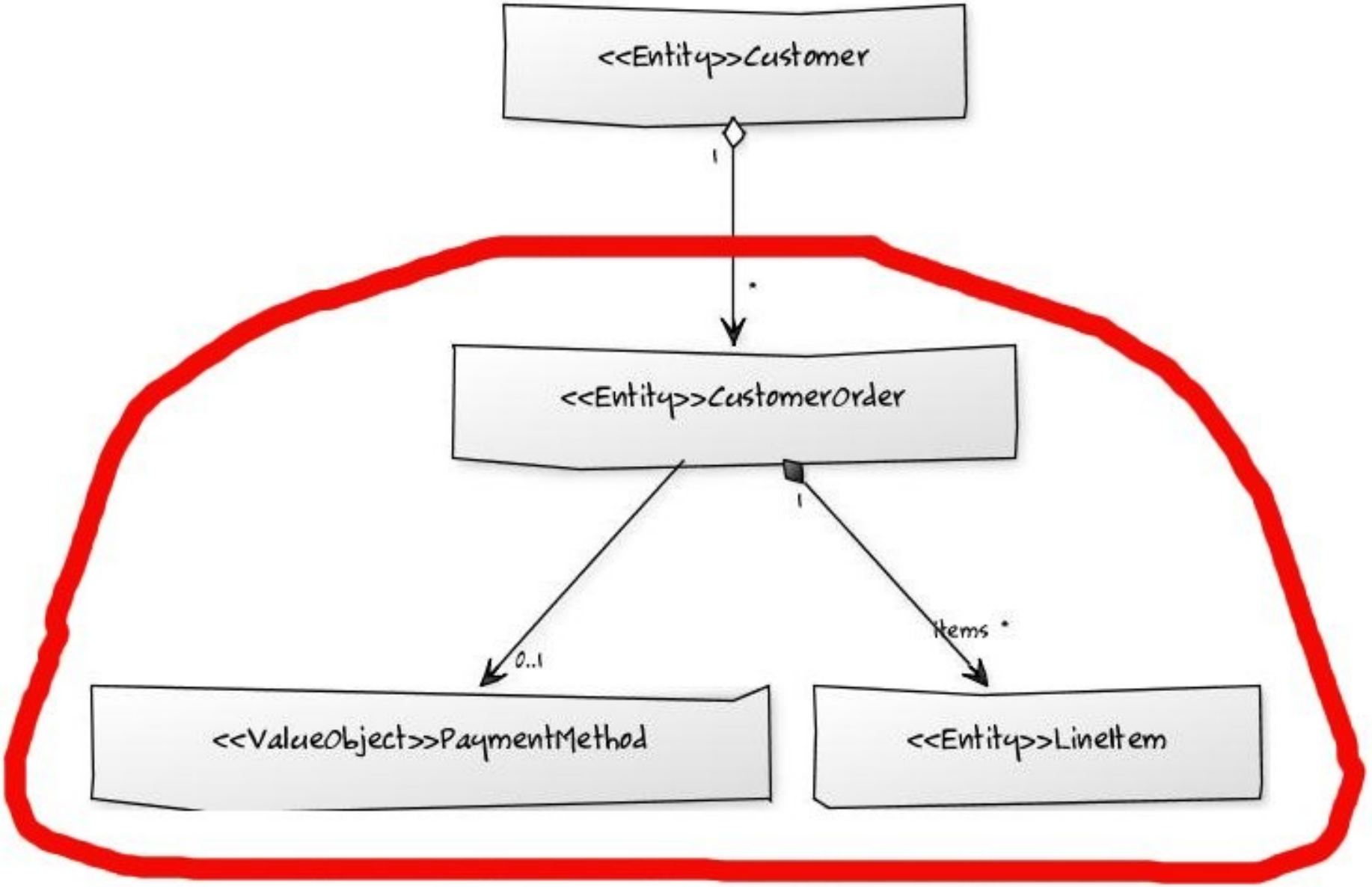
Immutable in certain implementations



Aggregate

- Subgraph of Entities and Value Objects
  - with a single entry: the root
  - possible multiple exits, mostly to follow during reading
- Unit of consistency: loose correspondency to database transactions
- Partitions the state of the application

Checkpoint: data modeling



Repository

- The gate to the database
  - One aggregate at the time
- The illusion of an in-memory collection of Entities
  - Fowler's definition
  - Here's a BookRepository

Factory

Encapsulate creation of complex Aggregates

- a new() is often all you need

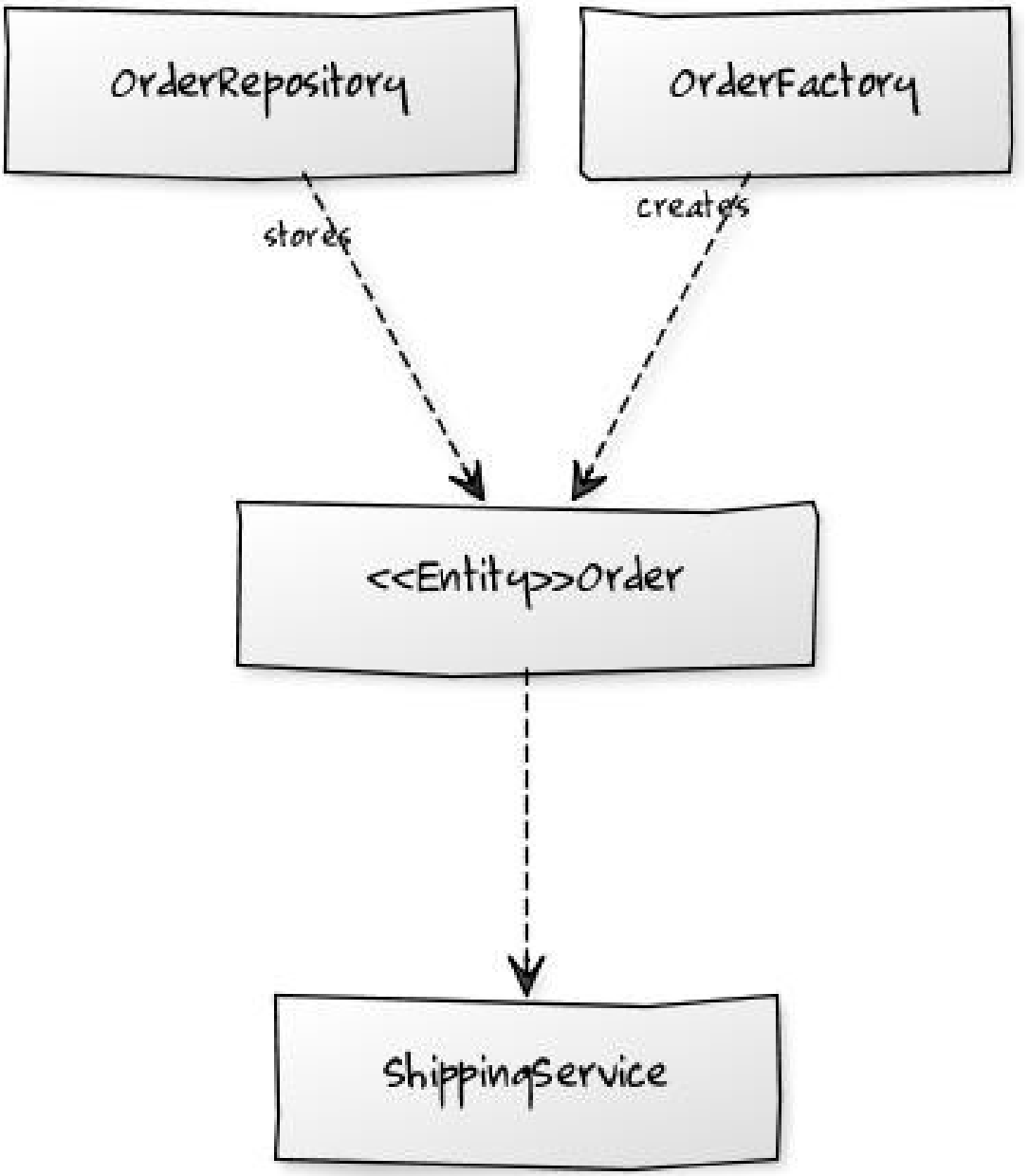
Service

At the **domain** level

To help Entities and Value Objects

- to avoid mutual dependencies
- or field references to strangers
  - optimal for isolation from libraries

Checkpoint: lifecycle classes



Meanwhile, in PHP...

On frameworks

Only one suggestion: build your Domain Model like if the framework didn't exist

Active Record vs. Data Mapper

Doctrine 2 for persistence (see tutorial by @juokaz)

By default for Doctrine 2 object === row

All the tricks are at http://github.com/giorgiosironi/ddd-talk

Entity

- @Entity annotation
  - no *extends*
- @Column for fields
  - private fields
  - types are PHP types: *string*, not *varchar*

Value Objects

Do you want an (id, date) table?

- Serialization of the whole object
- Conversion into a custom string/numerical format via a custom DBAL data type
- Deconstruction/reconstruction with lifecycle hooks
- Combined approaches: serialization and mirror fields

Aggregate

Relationships: **@OneToMany, @ManyToMany, @OneToOne, ...**

- `@OneToMany(..., cascade={“persist”, “remove”})`
- `@OneToMany(..., orphanRemoval=true)`
- `@ManyToMany(targetEntity="Phonenumber")`
  `@JoinTable(..., inverseJoinColumns={name=”phonenumber_id", referencedColumnName="id", unique=true)`

Repository

- Plain Old PHP Object
- Composing EntityManager
  - It's possible to define EntityManager custom repositories: quick and dirty
- Encapsulates queries
- Typical methods: add($root), remove($root), find($id), findByStrangeCriteria()
  - It's a collection!

Factory, Service

- POPO
  - Sometimes composing services or infrastructure objects (e.g. generating new progressive number for invoices, calculate current taxes, sending mails...)
- Often decoupled with an interface

References

The code shown in this talk

http://github.com/giorgiosironi/ddd-talk

Four books

http://domaindrivendesign.org/books

Domain-Driven Design mailing list

Google that :)

Q/A

Feedback

Testing in isolation tutorial: http://joind.in/3216

DDD talk: http://joind.in/3224

Thanks for your attention