DOMAIN DRIVEN DESIGN for MICROSERVICES ARCHITECTURE

领域驱动的微服务

Dec 2nd, 2017

ThoughtWorks®

领域驱动设计和微服务

刘传湘 & 笪磊

软件设计面临的挑战

产值达到**100**亿

首刚用了71年

联想用了13年

*小米用了**3**年*

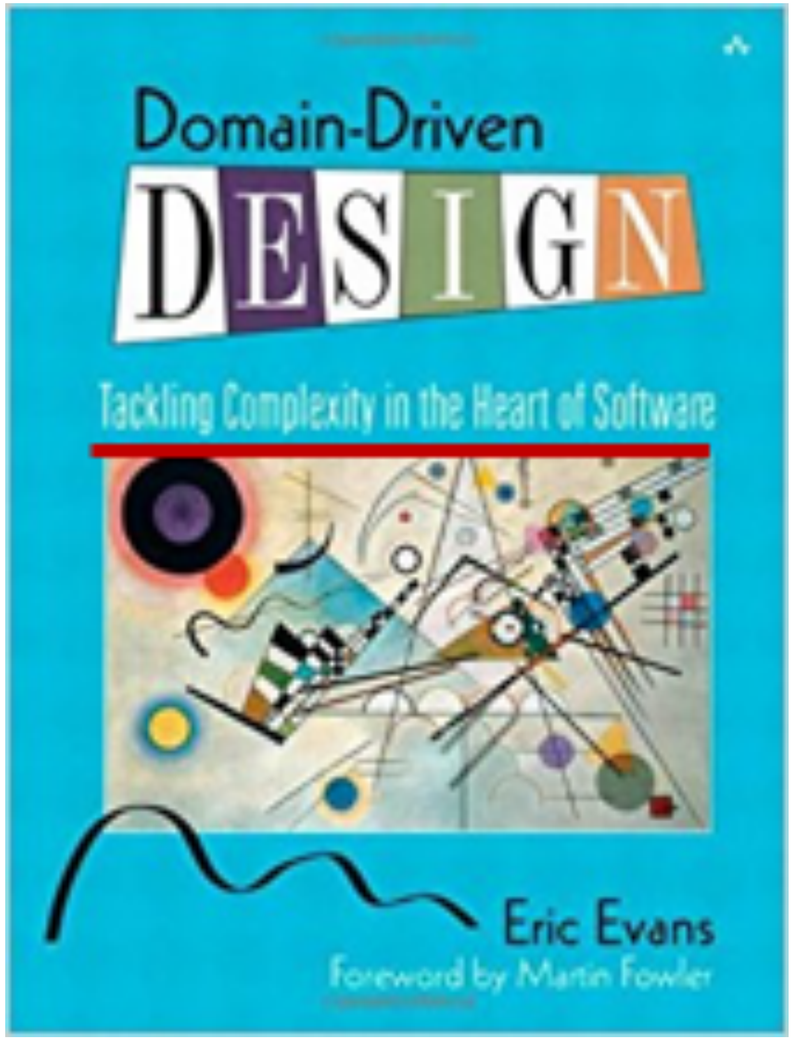
We Are All Software Companies Now

- WSJ Jun 9, 2016

软件设计面临的挑战

- *软件开发的不确定性贯穿了整个软件工程的生命周期。*
- *软件工程中不可能有任何"银弹"解决软件的复杂度问题。*
- *软件工程核心实质是社会工程，优秀团队的竞争力来源于互相的信任及良好的沟通。*

领域驱动设计



领域驱动设计是一种设计方法，试图解决的问题是软件的难以理解，难以演化. 领域驱动设计试图分离技术实现的复杂性，用围绕业务概念来构建领域模型的方式来控制业务的复杂性。

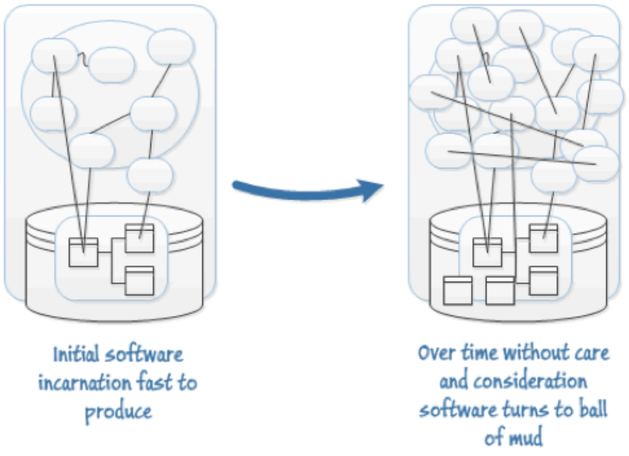
When you remember that DDD is really just “OO software done right”, it becomes more obvious that strong OO experience will also stand you in good stead when approaching DDD.

Source: http://www.developerfusion.com/article/9794/domain-driven-design-a-step-by-step-guide-part-1

微服务架构与领域驱动设计

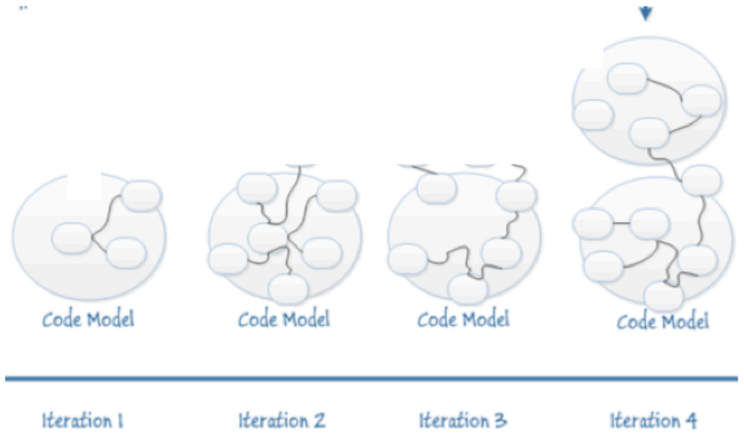
WHY? 领域驱动设计的提出距今已经十多年，但真正火热起来大约是在2013年微服务架构被提出之后。

项目1：
没有使用领域模型，对设计没有重视
- ✓ 迅速完成第一个版本
- ✓ 一年后陷入困境

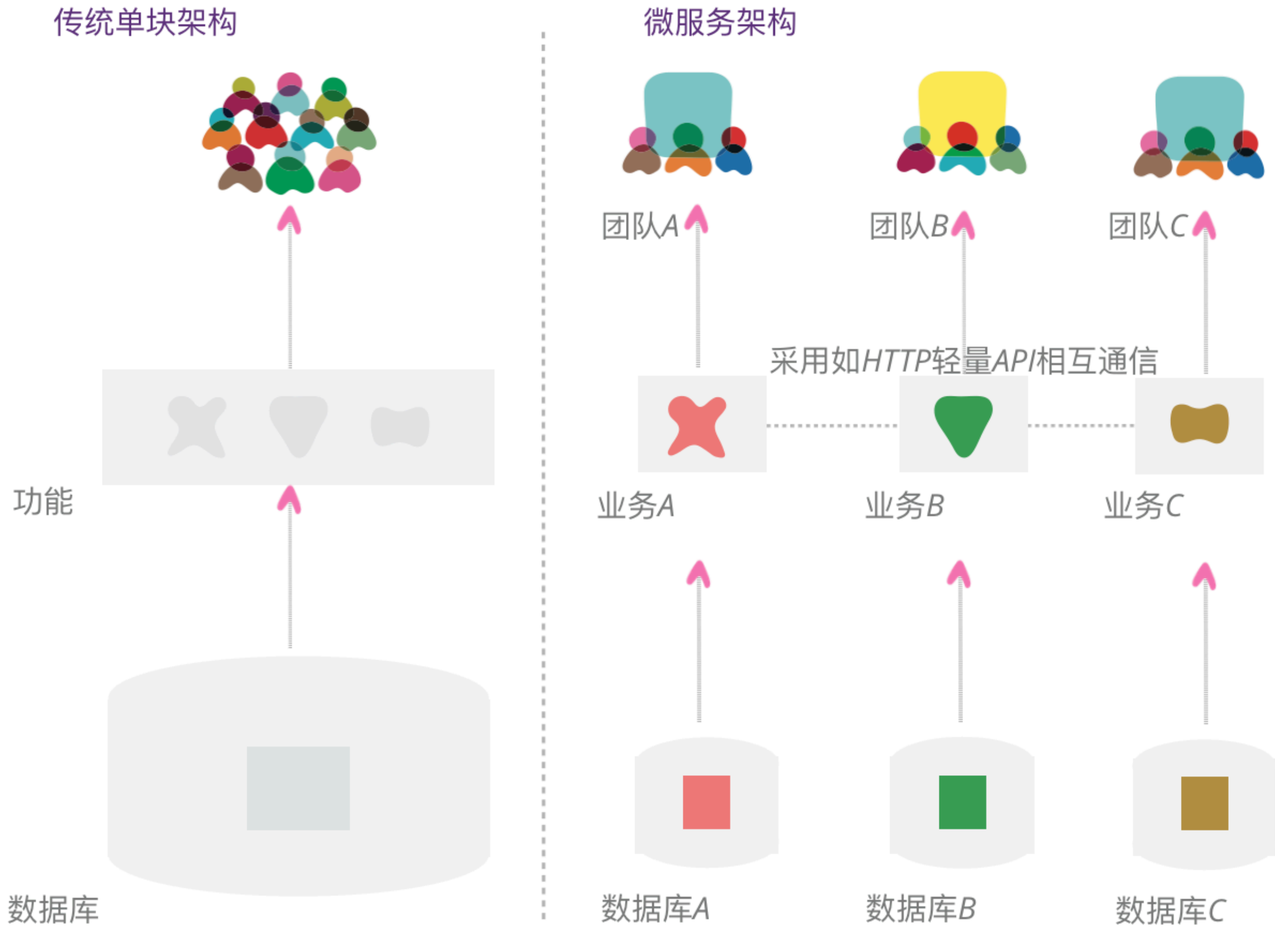


项目2：
业务专家进行高瞻远瞩的全局分析建模
- ✓ 分析模型过于复杂
- ✓ 分析模型与程序设计脱节

项目3：
重视领域模型，迭代开发完善模型
- ✓ 提供灵活性和扩展性
- ✓ 根据团队对领域的不断理解，深化领域模型



微服务架构



> 微服务架构将业务和数据剥离

每条业务做到4个独立：

独立进程

以开发一组小型服务的方式来开发一个独立的应用系统，其中每个小型服务都运行在自己的进程中。

独立部署

服务围绕业务功能进行构建，并能通过全自动的部署机制来进行独立部署

独立技术

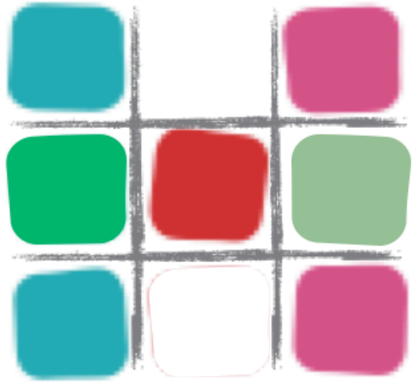
每条业务线可以使用不同的开发语言、数据存储技术，并保持最低限制的集中式管理

独立团队

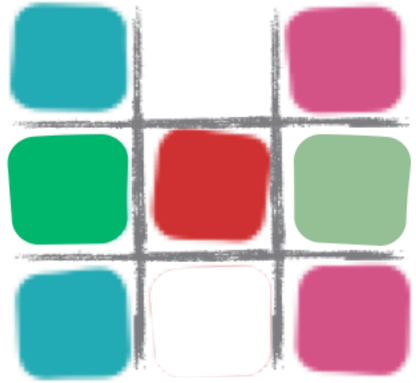
每条业务线均配备开发、测试、运维、DBA等更具生产力、更对结果负责的全功能团队

和传统架构设计方法相比...

业务架构



系统架构



技术架构

DDD要解决的两个核心问题：

1. **业务架构如何合理的设计划分？**
2. **如何使系统架构与业务架构保持一致？**

领域驱动设计全景图

- DDD作为一个设计方法提供了**迭代梳理**业务架构和系统架构的模型语言。
- DDD强调业务和系统的绑定关系，明确了**跨职能**（业务和技术）**协作**在架构设计中的重要性。

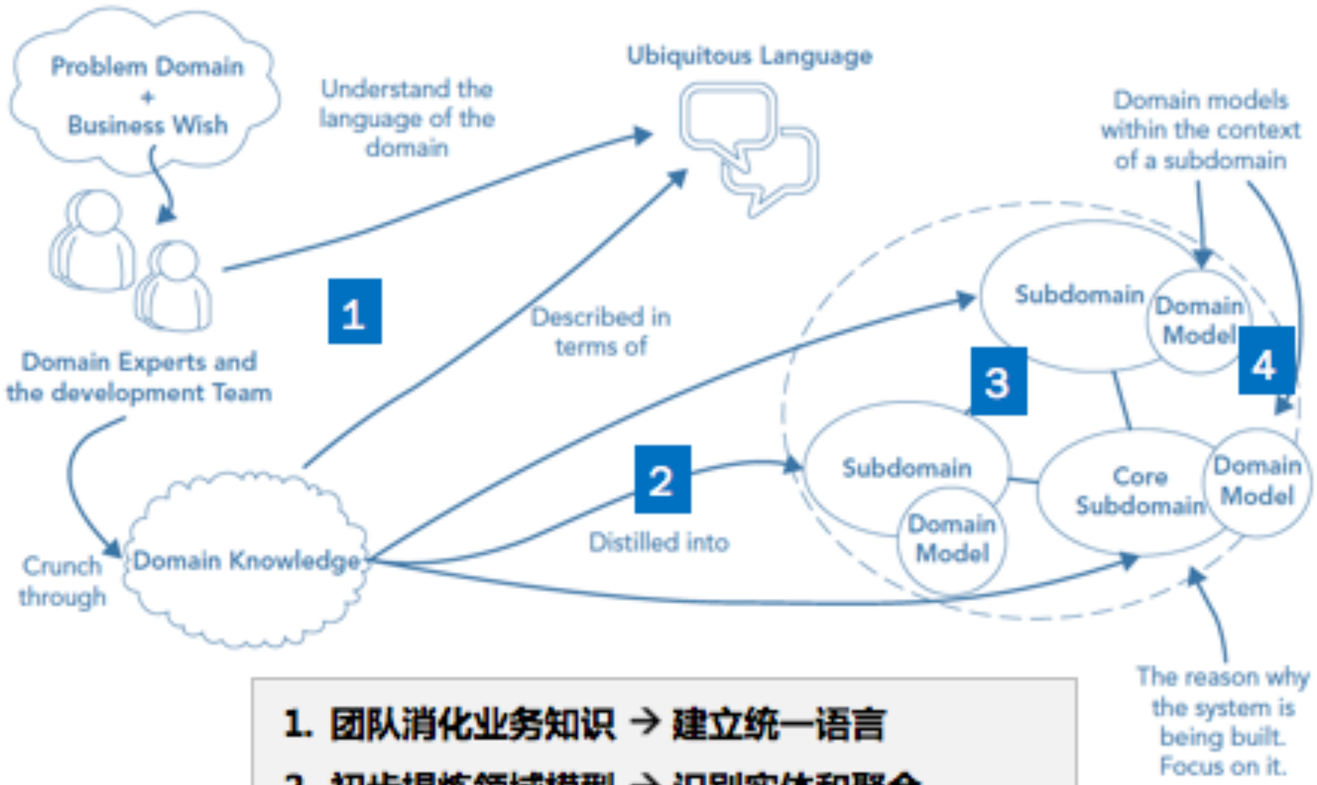
领域驱动设计三原则

P1: Focus on your core domain.

P2: Iteratively explore models in a creative collaboration of domain practitioners and software practitioners.

P3: Speak a Ubiquitous Language within an explicitly Bounded Context.

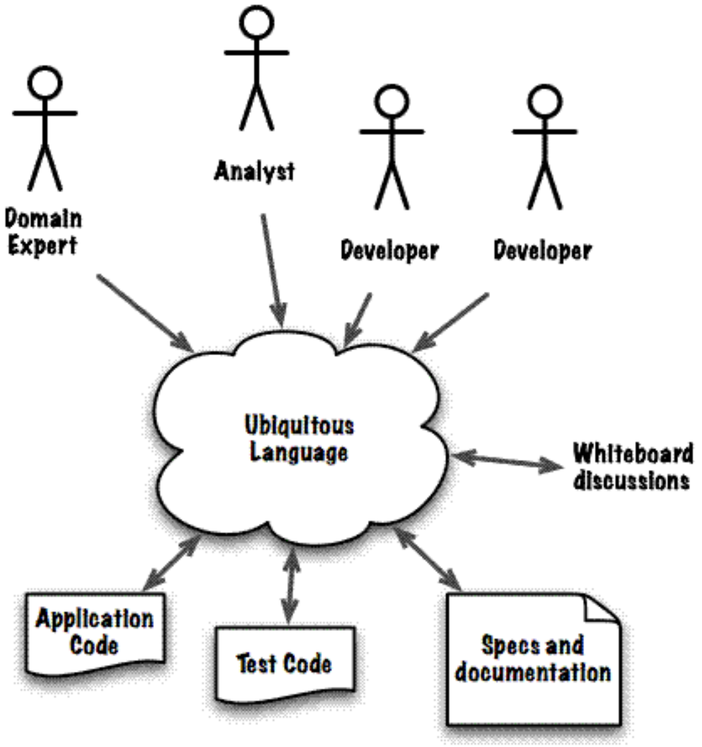
领域驱动设计



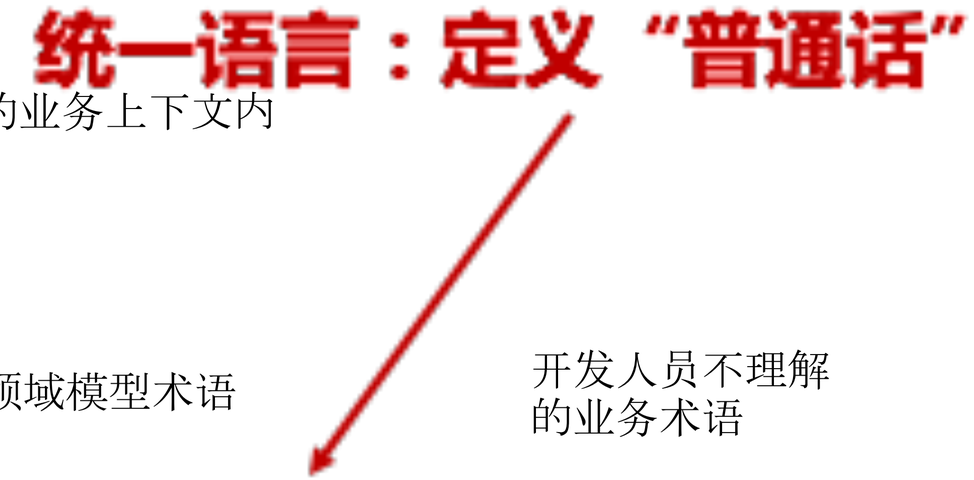
1. 团队消化业务知识 → 建立统一语言
2. 初步提炼领域模型 → 识别实体和聚合
3. 分解领域模型复杂度 → 识别限界上下文
4. 细分领域模型内部元素 → 识别事件和命令

统一语言

http://openprtspecs.blogspot.com



统一语言



对象: 实体与值对象

- 账户是不是实体？
- 转账是不是值对象？

- 假如有个体育场提供一个web应用让用户在线订票。 存在**观众**和**座位**这两个概念，他们是不是实体？

对象: 实体与值对象

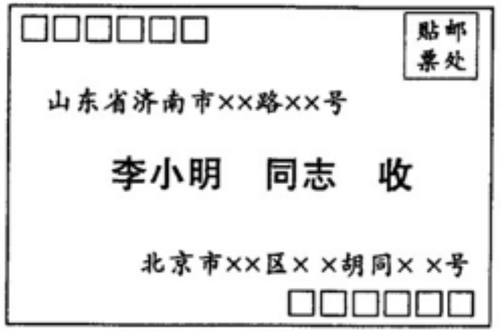
实体

- ➢具有生命周期
- ➢有唯一标识
- ➢通过Id判断相等性
- ➢增删改查/持久化
- ➢可变
- ➢比如Order/Car

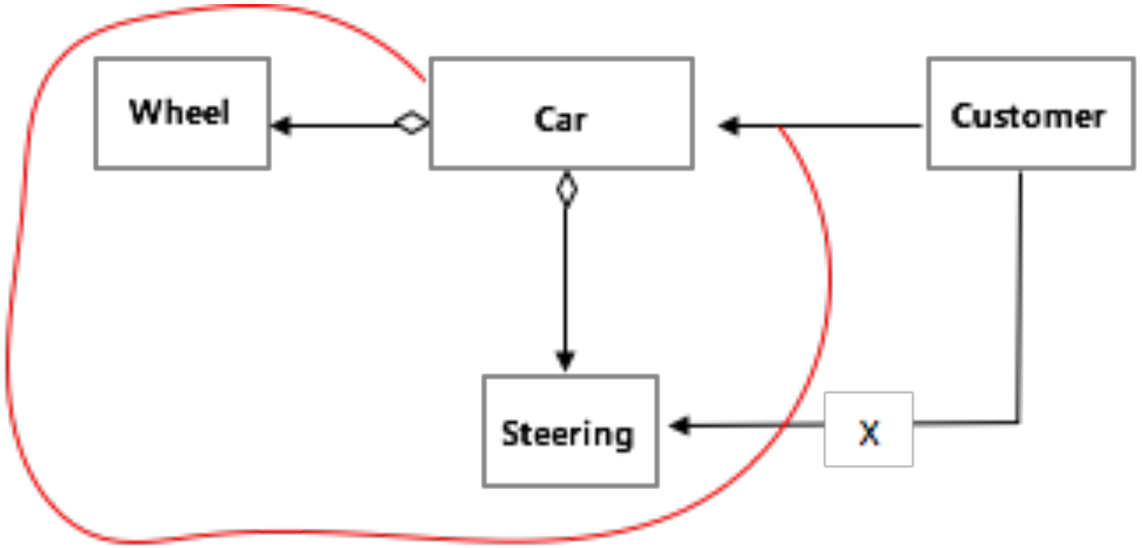
VS

值对象

- ➢起描述性作用
- ➢无唯一标识
- ➢通过属性判断相等性
- ➢实现Equals()方法
- ➢即时创建/用完即扔
- ➢不可变(Immutable)
- ➢比如Address/Color



实体与聚合

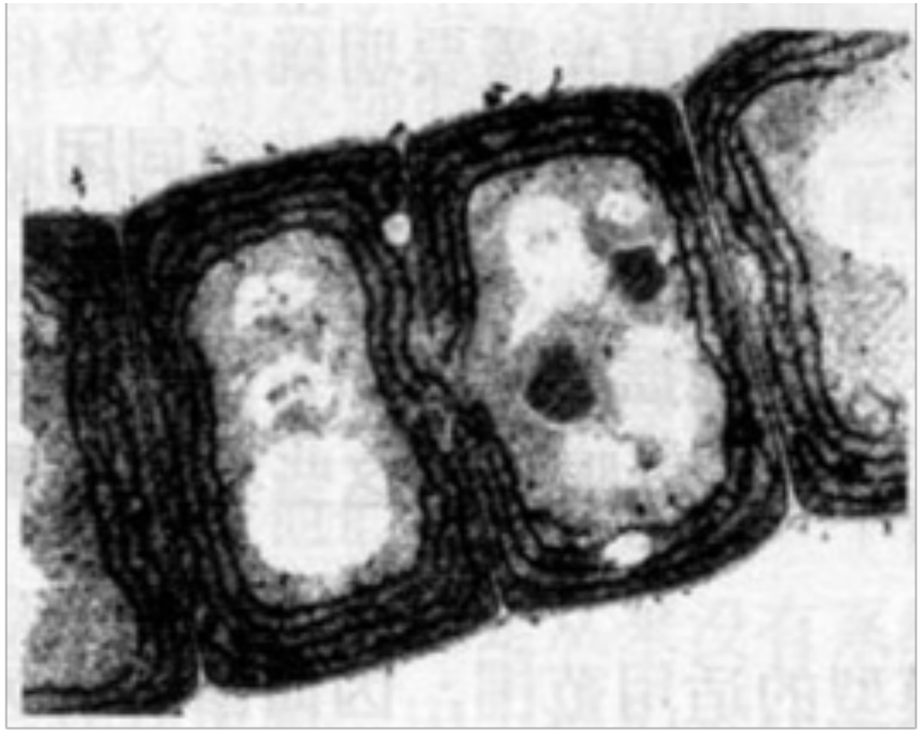


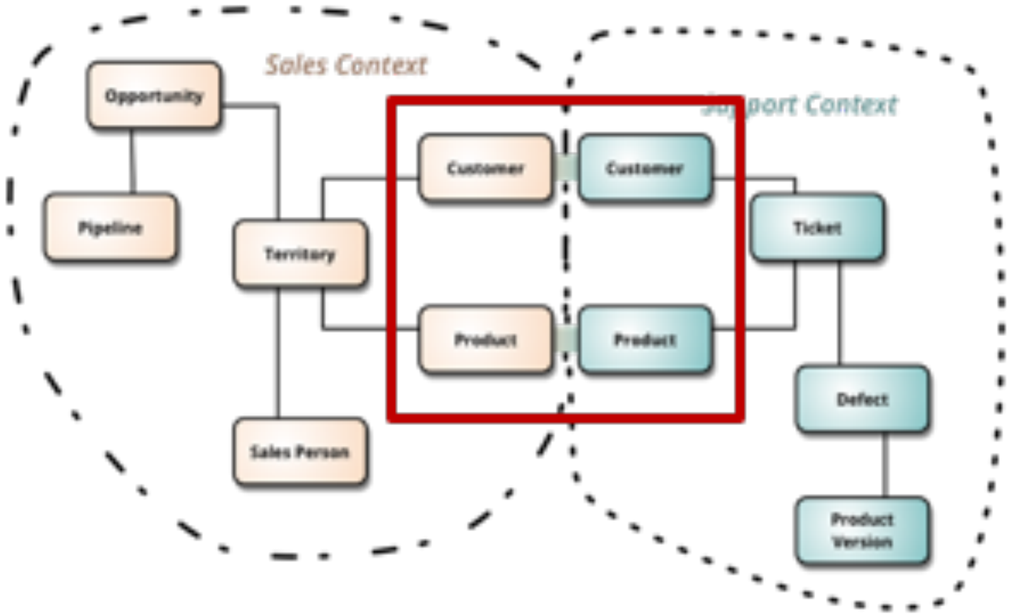
在给定的业务上下文里一辆汽车有4个轮子和一个方向盘。

聚合是一组可以被视为单个领域对象的集合。聚合通过定义清晰的所属关系和边界，并避免错综复杂的对象关系网来实现模型的**内聚**。

考虑一个包含多条记录的订单，每条记录都是单独的对象，但订单（连同所有记录）一起视为单个聚合是有用的（封装业务规则）。

界限上下文





限界上下文主要用来封装通用语言和领域模型，显式地定义了领域模型的**边界**（所应用的上下文）。

https://martinfowler.com/bliki/BoundedContext.html

事件与命令

(Domain) Events

*It really became clear to me in the last couple of years that we need a new building block and that is the **Domain Event**.*

Eric Evans

*An **event** is something that has happened in the past.*

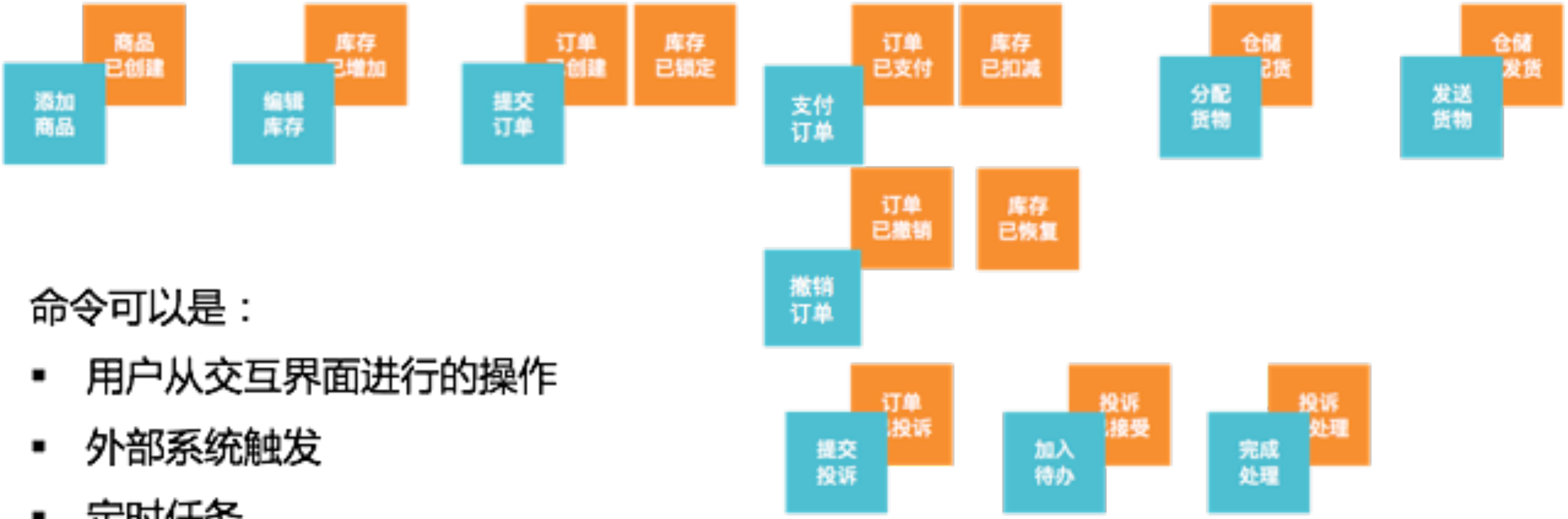
Greg Young

*A **domain event** … captures the memory of something interesting which affects the domain*

Martin Fowler

事件与命令

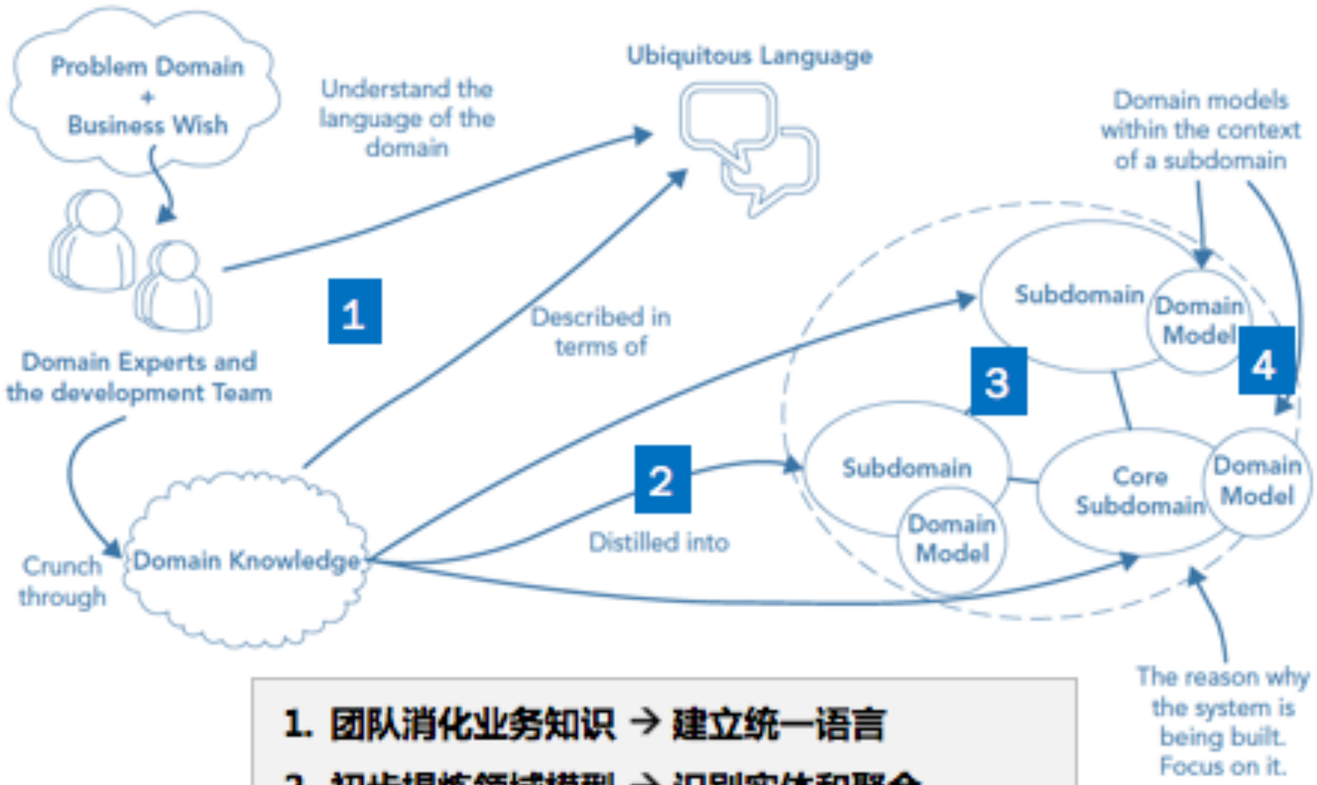
根据前面识别出来的领域事件，对可能产生事件的命令进行建模：



命令可以是：

- 用户从交互界面进行的操作
- 外部系统触发
- 定时任务

领域驱动设计



1. 团队消化业务知识 → 建立统一语言
2. 初步提炼领域模型 → 识别实体和聚合
3. 分解领域模型复杂度 → 识别限界上下文
4. 细分领域模型内部元素 → 识别事件和命令


THANK YOU