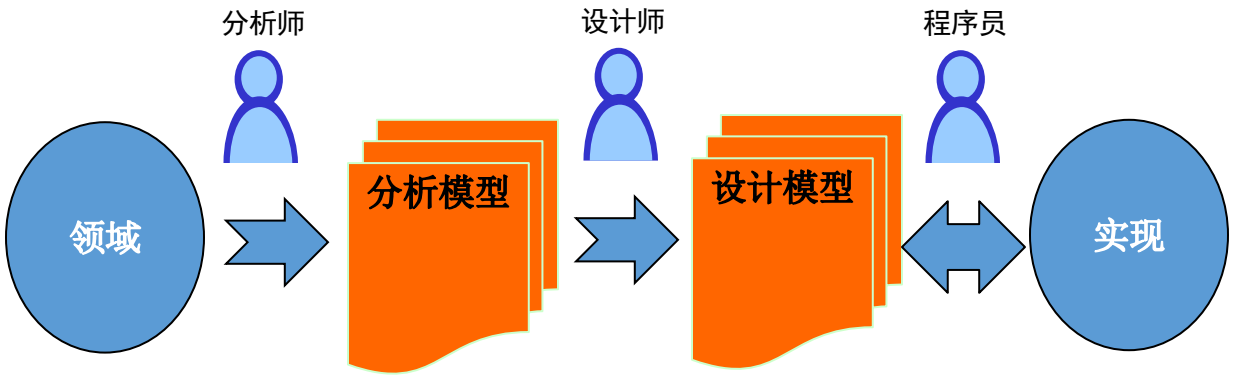
DDD中的康威定律-系统架构驱动组织架构

丁辉

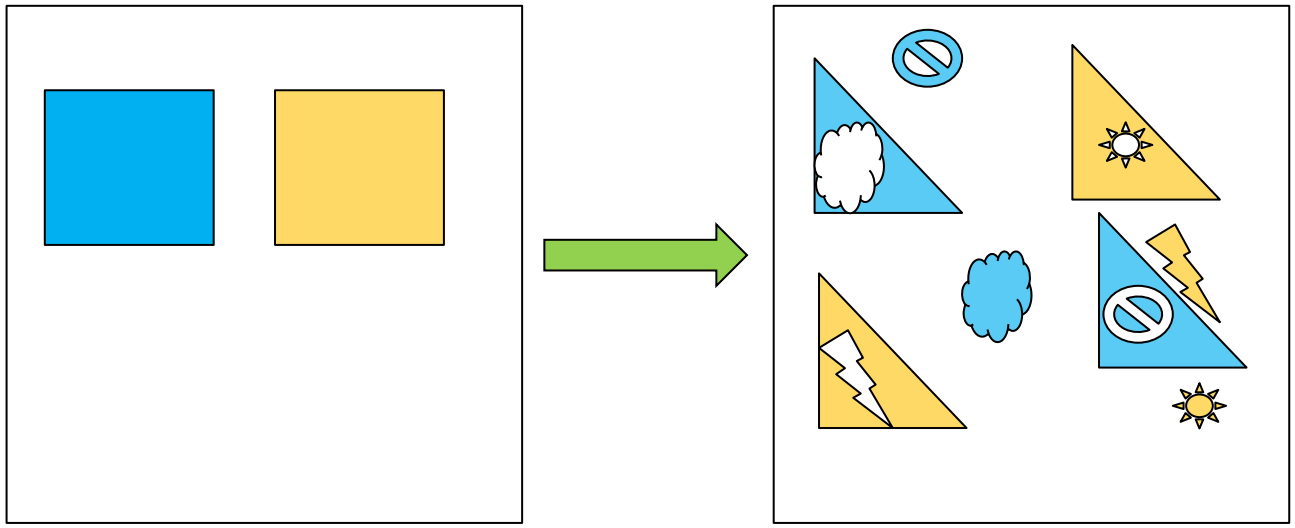
讲师简介

- 丁辉
  - 中兴通讯公司级敏捷教练和代码大全、代码设计训练营教练，13年软件开发经验，8年项目管理和流程改进经验
  - 指导并参多个团队由传统研发模式向敏捷研发模式转型（其中超过100人的大型团队成功项目级敏捷转型5个）。
  - 在敏捷导入、指导团队转型、CI、核心技术实践、自组织团队建设等方面具有丰富的实战经验；
  - 对如何提升员工代码设计能力和提升代码内在质量、遗留代码重构、DDD/DSL架构设计等方面也有较多理解和解决思路；
  - 精益创业教练，曾指导多个创业团队产品设计、团队运作、技术架构。
  - 人工智能平台+HPC专家

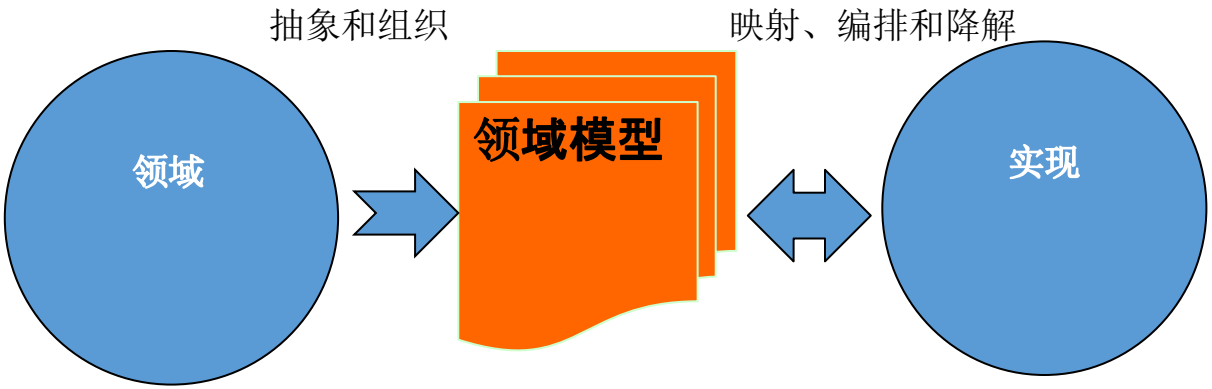
传统过程



Mapping



将模型和实现绑定



DDD的本质

- 分

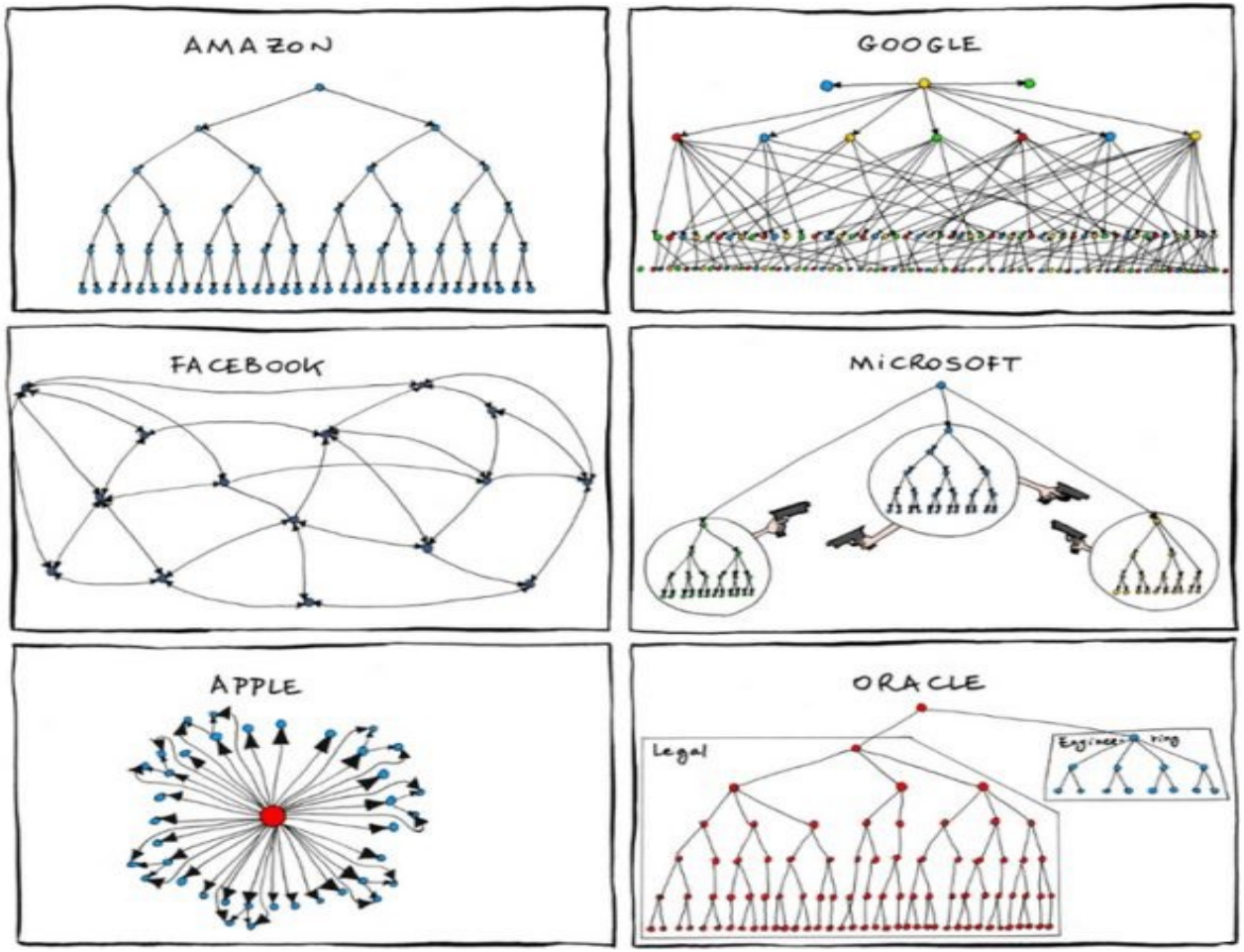
合

万事大吉了吗？？

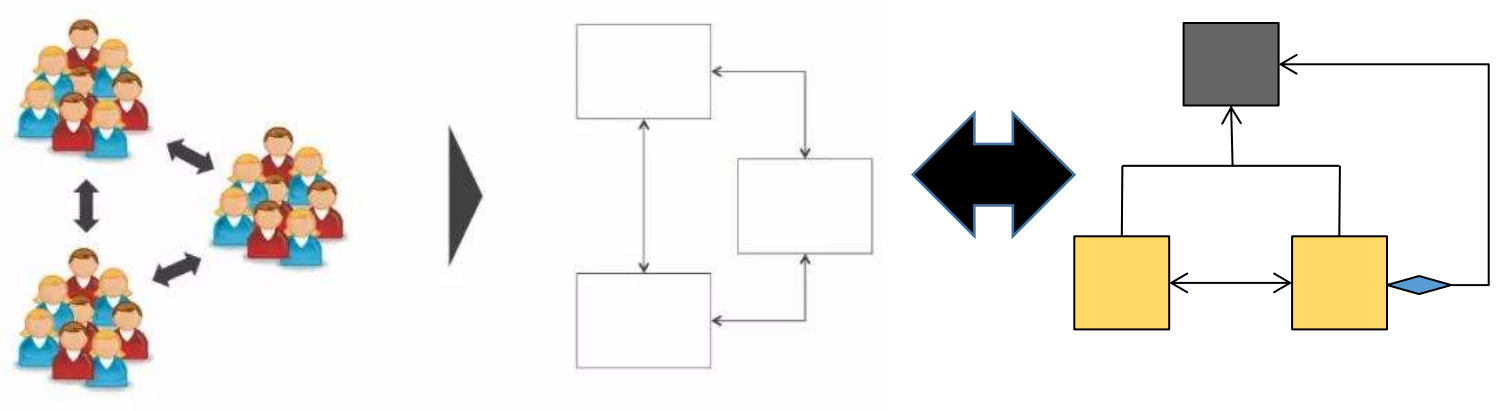
康威定律

- Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations. - Melvin Conway(1967)
- 设计系统的组织，其产生的设计等同于组织之内、组织之间的沟通结构

实例



DDD架构和团队之间Mapping



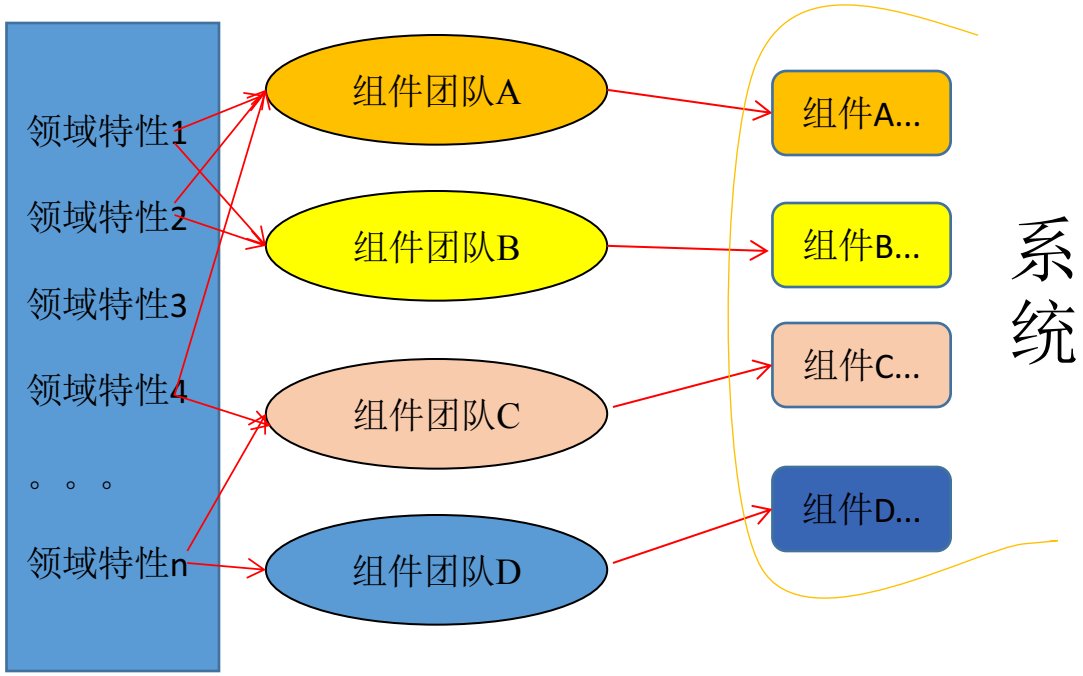
实例

- 团队40+
- 拆分为5个领域方向

阶段一

- 按专业方向组件团队

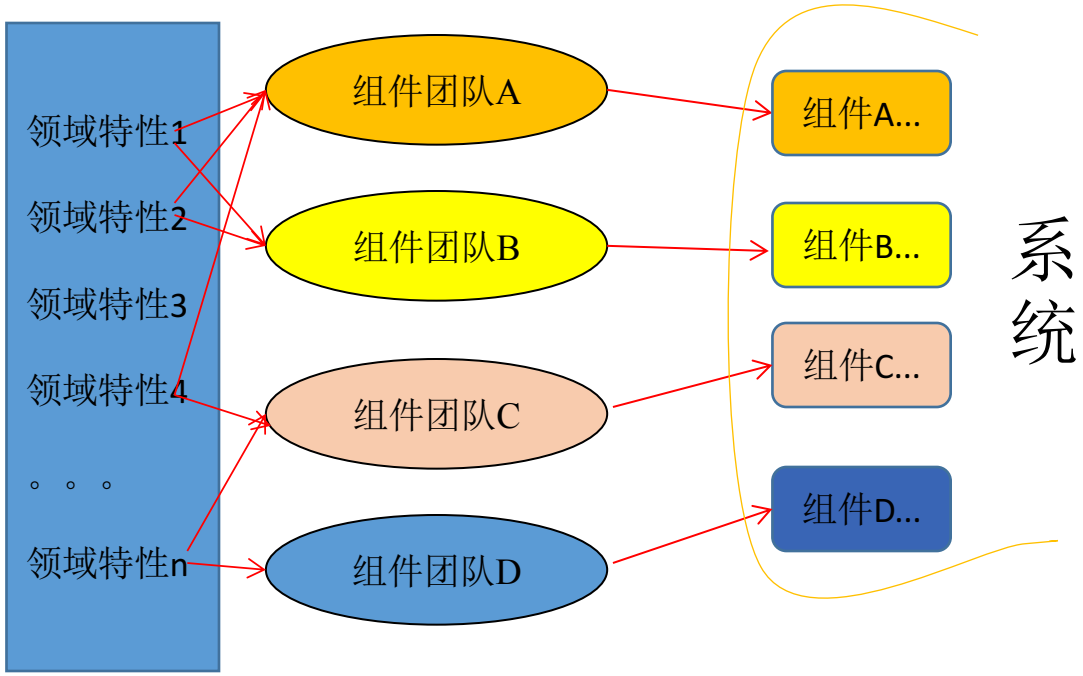
延迟价值交付



•不是每个团队都工
作在最高价值特性上

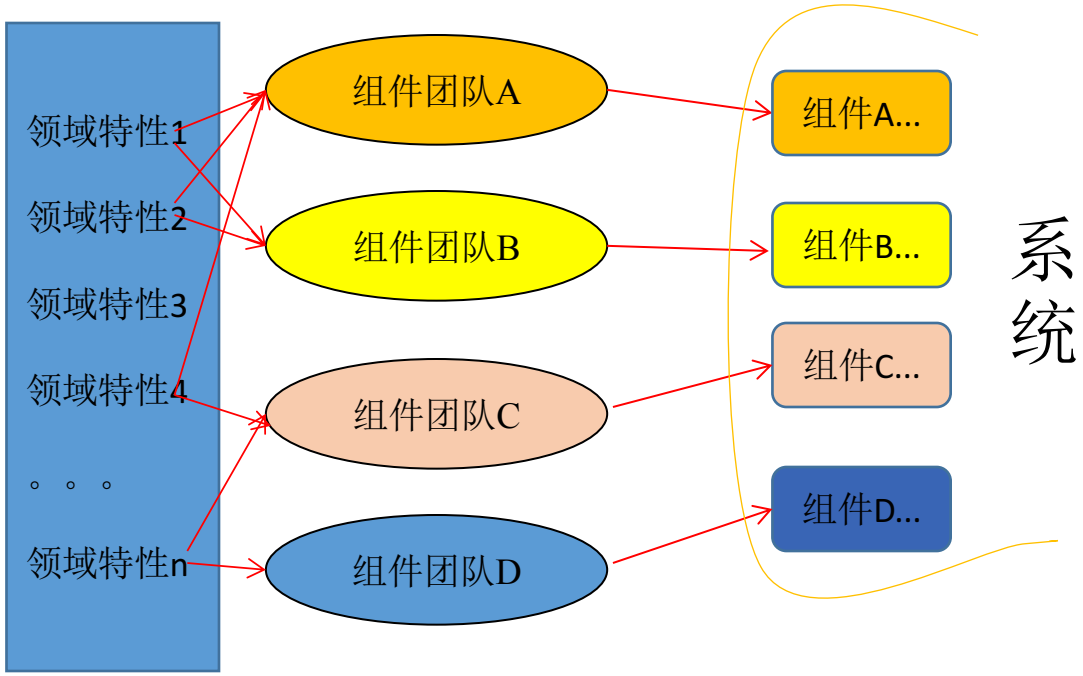
•团队A忙于特性1的工作，团队C的目标是特性4，但由于组件A还没有完成，所以特性4还无法集成、测试和交付，需要等组件A的工作完成。

鼓励代码产出最大化，而非价值最大化



- 工作的挑选是基于组件团队擅长的（专业分工）而不是客户价值
- 局部优化，每个人表现的都很忙
- 迫使一些团队进行“人为创造的”工作。

计划与协调复杂



•所有团队在迭代计
划中相互依赖，需要同步工作

•一个团队的延误会
对所有组件团队产生连锁反应。

持续上升的人力成本

- 现有人员分配与高优先级工作间很难匹配
  - 现有版本，高优先级的任务主要集中在组件A，现有人员无法满足其需求，因此招聘了更多的开发人员。
  - 下个版本，高优先级的任务包含了更多的组件C。组件C成为瓶颈，需要招聘更多的开发人员。
  - 很难通过在不同组件团队之间转移开发人员来解决
- 另外的组件团队都很忙（追求资源高利用率），不想人员流失
- 将某一组件的专家转移到另一组件团队是一种浪费
- 害怕需要很长时间才能掌握另一项技能

帕金森定律：只要管理人员的威望和权利与他的预算紧密相关，那么他总会想办法扩展自己的组织。

限制人的潜力

- 限制学习
  - 大部分开发人员只了解系统中的某个小片段，他们看不到或学不到更多新的东西
  - 开发人员所掌握的知识与他们的效率有着强有力的联系
  - 例子： 复杂的问题往往是跨模块的，经常需要求助于一些非常了解系统的人
- 人为的瓶颈（学习负债）
  - 举个极端的例子：

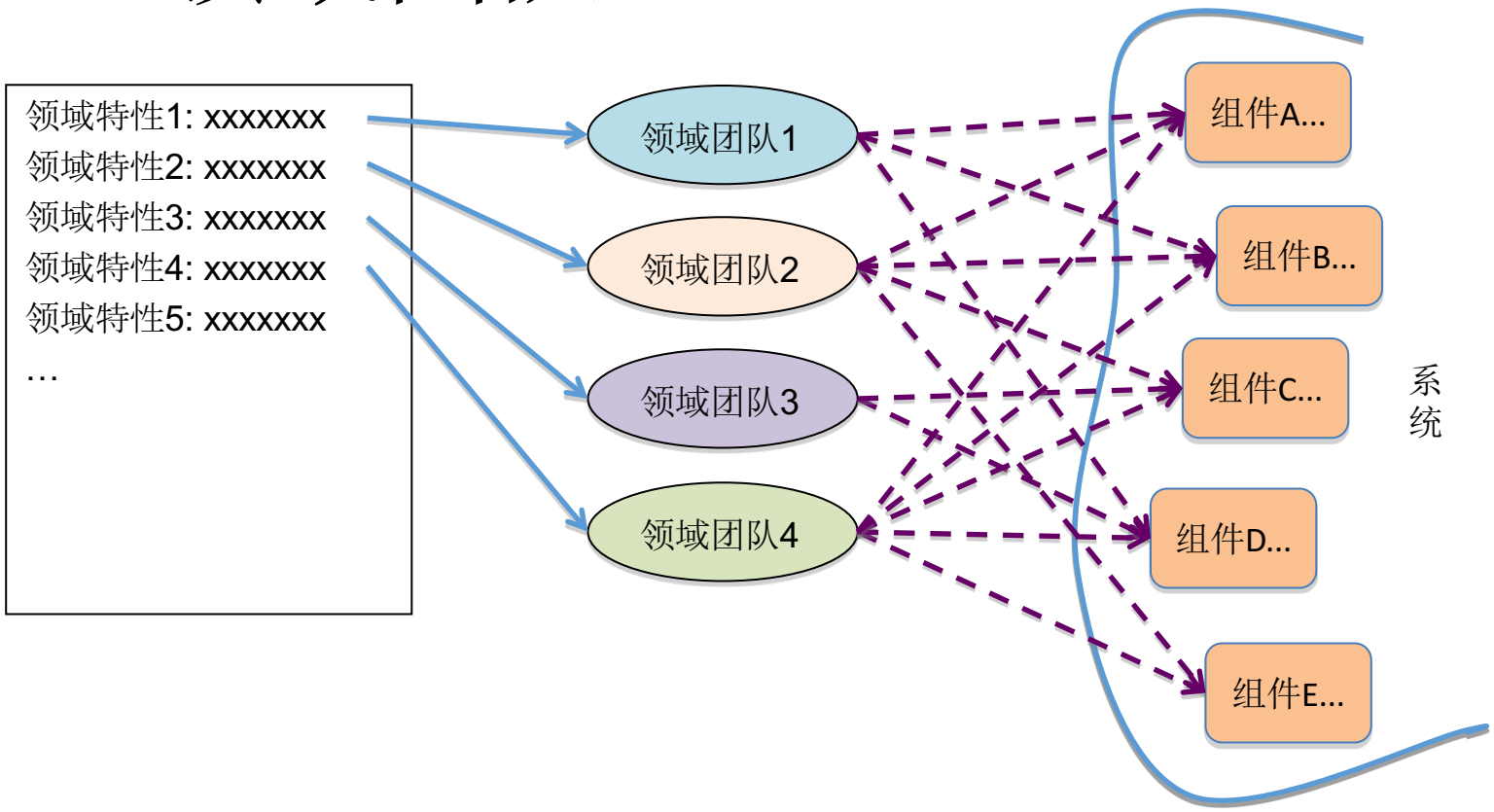
  选项一、每个人都参与产品开发并能把所有的事做好。

  选项二、每个人可以出色的完成一项（仅此一项）任务，除此之外没有别的贡献。哪个会出现更多的瓶颈？哪个可以更快的交付特性？

阶段二

- 领域团队

领域团队


领域特性1: xxxxxxx
领域特性2: xxxxxxx
领域特性3: xxxxxxx
领域特性4: xxxxxxx
领域特性5: xxxxxxx
…
领域团队1
领域团队2
领域团队3
领域团队4
组件A...
组件B...
组件C...
组件D...
组件E...
系统

领域团队的好处

- 增加价值流量
  - 优先开发高价值的特性
- 变更更加容易
  - 不再需要多个团队的重新协调与计划
  - 同时进行的工作减少（Work In Progress）
- 简化计划
  - 给予团队整个特性开发的工作，组织和计划工作变得相对简单
- 减少交接产生的浪费
  - 整个特性团队协同工作（分析、设计、编程、测试），交接工作大幅度减少了

领域团队的好处

- 减少等待时间，周期加快
  - 排除了工作交接
  - 完成客户特性不需要等待不同部门的同步工作，减少了等待产生的浪费
- 自管理，提高成本效益
  - 团队有责任完成端到端的工作，承担了绝大部分与其他团队协调的工作
- 增进学习，发挥人的潜力
  - 责任范围扩大
  - 不同领域专家坐在一起共同协作，增加了个人和团队的学习机会

领域团队的好处

- 提高工作动力
  - 团队意识到完成了一项直接面向客户的端到端特性时，容易产生更高的动力和工作满意度
- 使软件模块间协作、接口定义更容易
  - 特性团队更新所有模块的代码，协调更容易

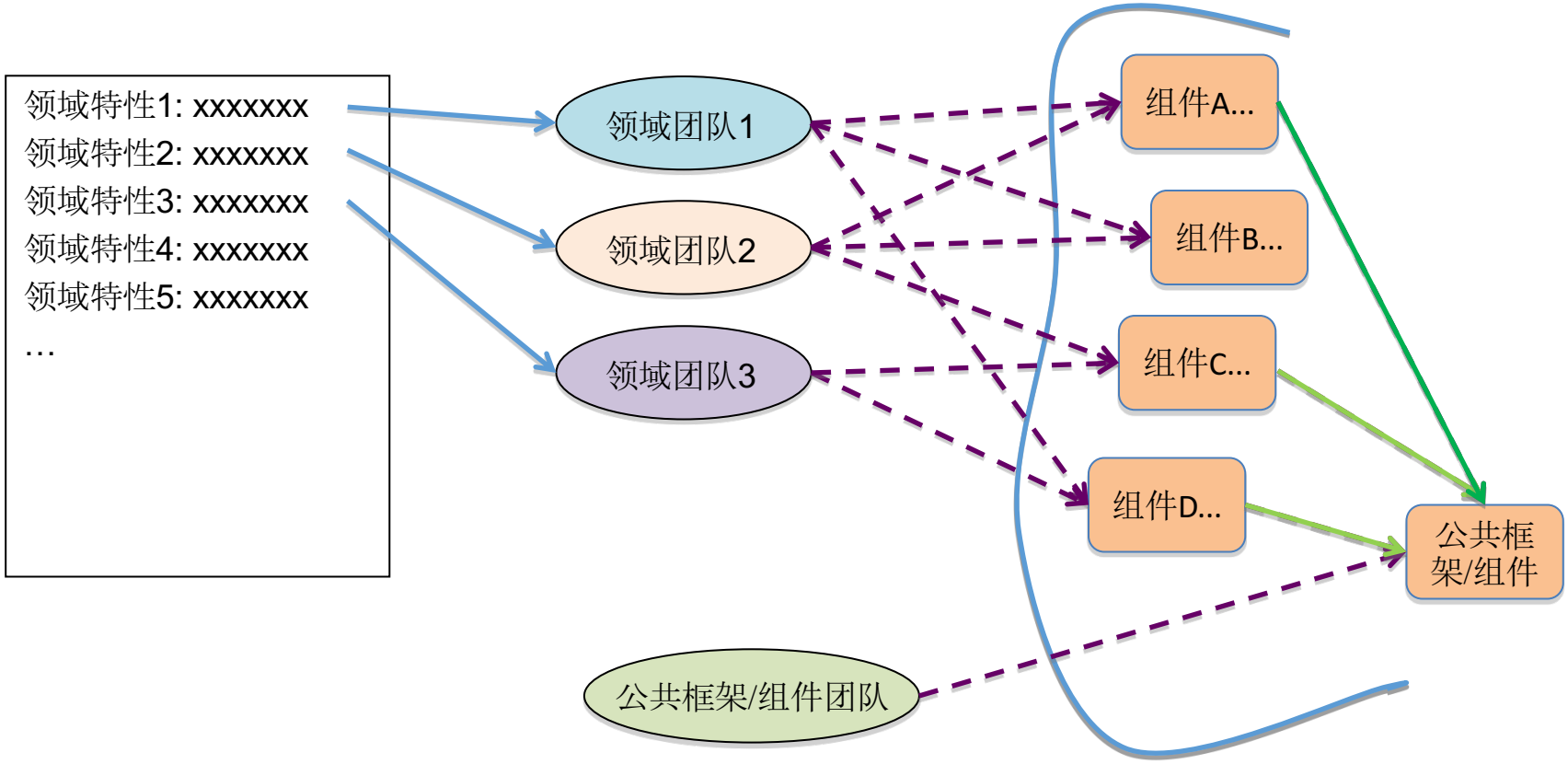
问题

- 原来的组件专家如何分配到领域团队
- 架构、知识、技能复用问题

阶段三

- 领域团队+功能复用公共框架组件团队

功能复用公共框架/组件团队



优势

- 公用功能框架和组件集中开发维护
- 形成复用

挑战

- 功能复用度
  - 各个项目复用比例不一致
- 逐步出现公用组件的定制化的情况
- 需求响应不及时
- 某些功能，工作范围和质量牵扯不清

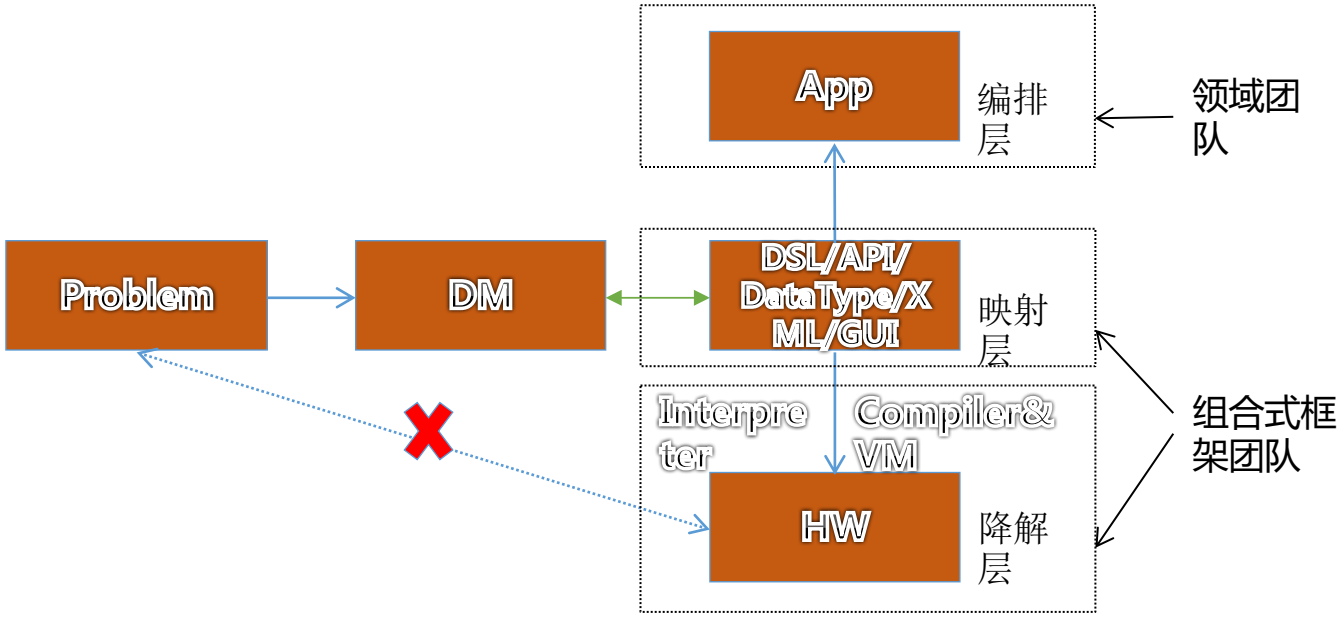
阶段四

- 领域团队+组合式框架团队

思路

- 功能复用公共框架/组件团队的根本问题
  - 公用组件的抽象粒度过大，调用方调用不够灵活
  - 公用组件的间组合方式不够，通过api调用组合后，还需要较大代码量才能实现新的组合
- 对领域层语义进行平滑降解
  - 复用层框架/组件进一步抽象
  - 抽取原子
    - 规则+关系
  - 提供组合手段
    - 规则通过关系组合，组合后的东西还具有原子性，仍然可以和其他原子和关系进一步组合

基于原子组合的复用思路



实例 告警屏蔽

- rule原子：

has
not
screen

- relation原子：

squential
optional
or
and

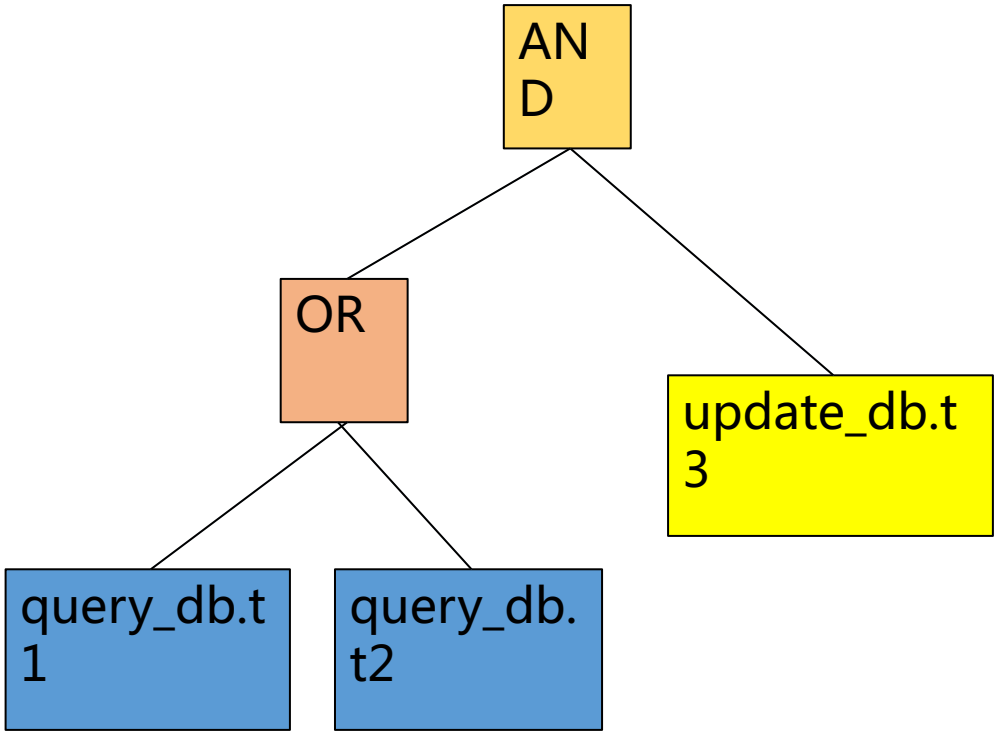
```
__def(ScreenRule) __as
(
    __sequential
    (
        __optional(__has(loflom), __sequential
        (
            __screen(ais),
            __screen(lck),
            __screen(tim),
            __screen(bbe),
            __screen(bdi)
        )),
        __optional(__or(__has(lck), __has(ais)) , __sequential
        (
            __screen(tim),
            __screen(bbe),
            __screen(bdi)
        )),
        __optional(__has(tim), __sequential
        (
            __screen(bbe),
            __screen(bdi)
        ))
    )
);

__def(GenerateRule) __as
(
    __sequential
    (
        __optional(__or(__has(loflom), __has(ais), __has(lck), __has(tim)), __generate(aais)),
        __optional(__and(__has(bbe), __not(__has(bdi))), __generate(arei))
    )
);
```

实例规则配置

rule原子：
query_db、update_db

relation原子：
AND、OR



现状

- 前三阶段的问题得到了有效解决
- 组合式框架团队工作聚集，原子的质量和易用性和性能大幅提升；新增原子和修改原子的响应速度较快；
- 领域特性组开发难度降低，需求响应速度加快，人员分流
- 自动化测试性价比提升，功能覆盖率增大

Q & A

谢谢大家！☺