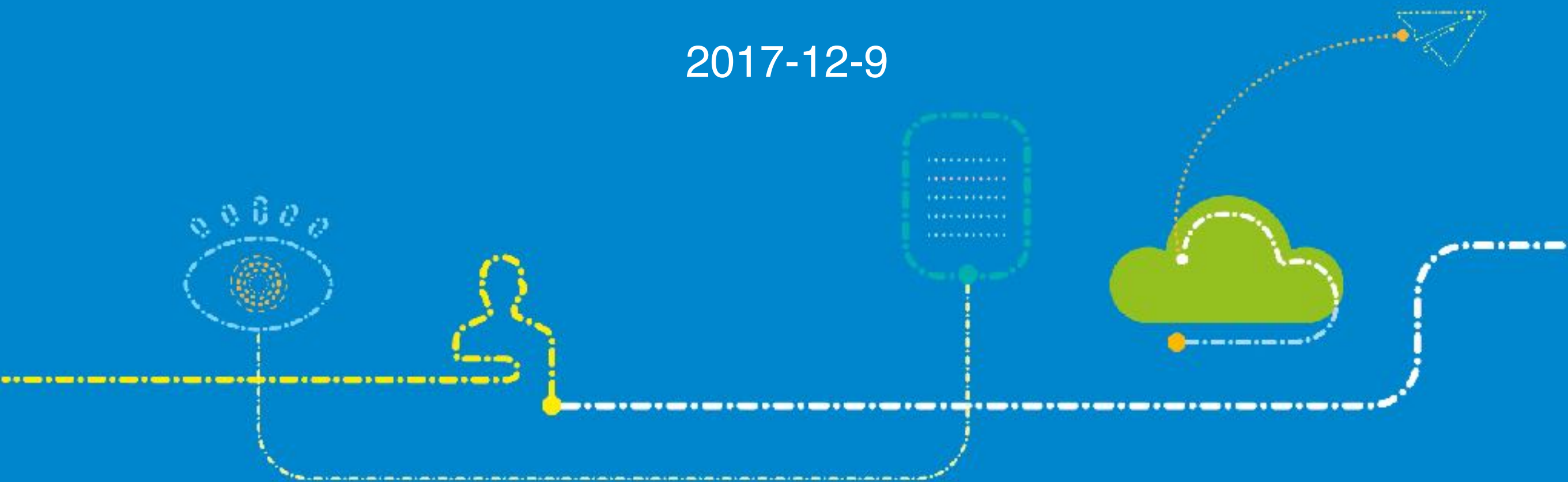




DDD驱动测试体系建设

张晔@ZTE

2017-12-9



测试体系



目录



自动化测试的窘境

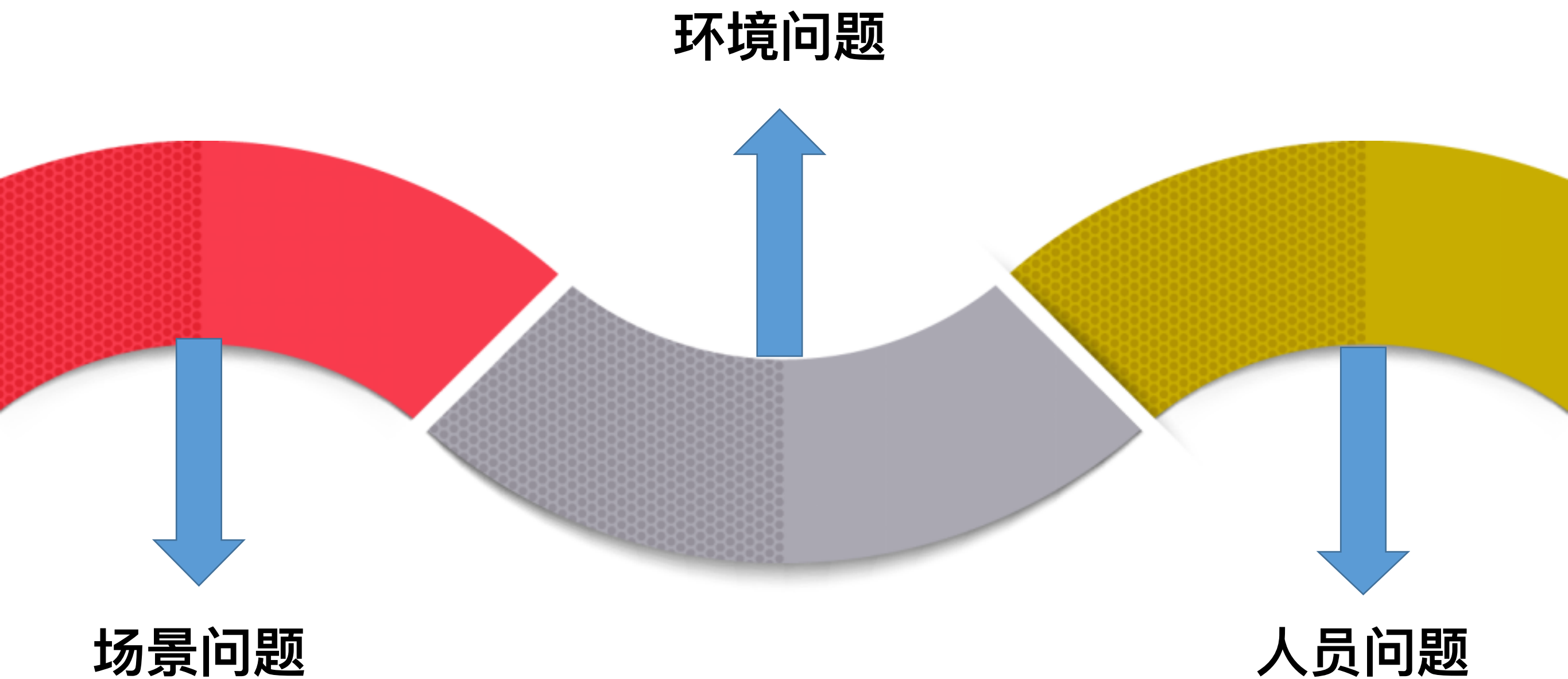


DDD驱动用例设计

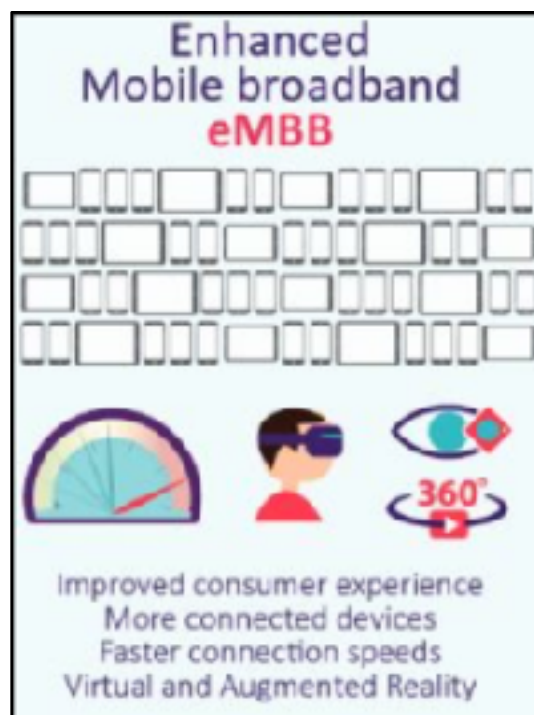


DDD驱动用例实现

典型问题



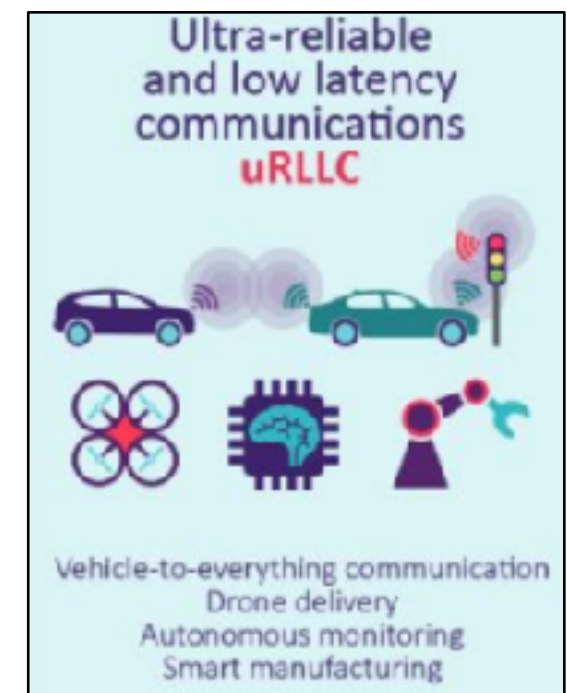
测试的场景问题



大容量

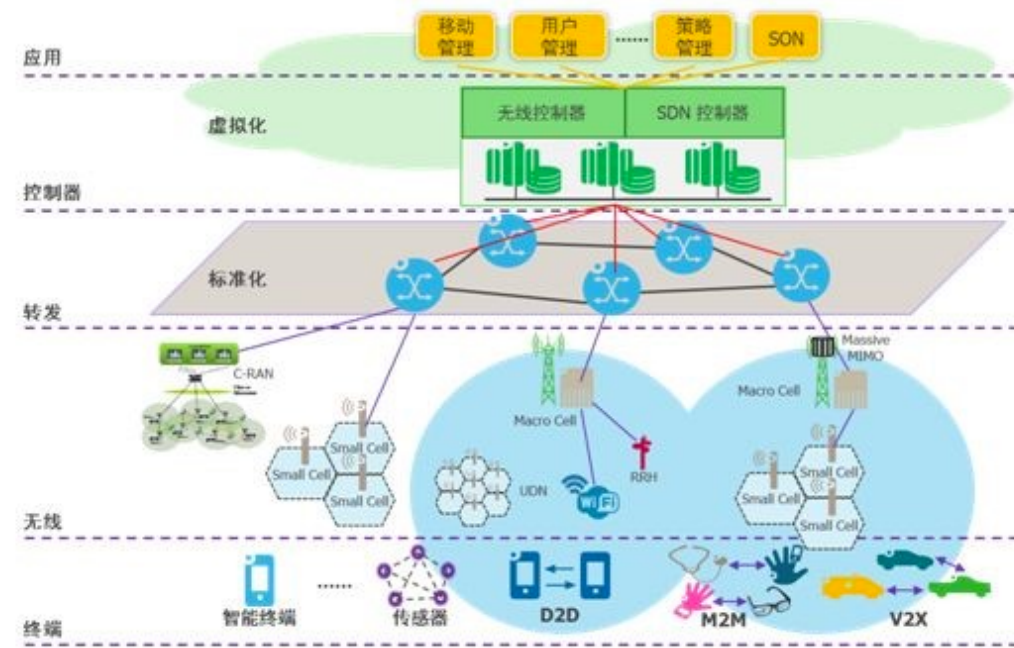


大规模互联



实时通讯

测试的环境问题



工具软件



系统设备



通信仪表

测试的人员问题



- 代码能力欠缺
- 不了解实现细节

- 不熟悉测试业务
- 不熟悉测试环境

自动化测试



数据驱动的用例设计

- 自研自动化测试平台
- 测试数据和测试过程分离
- 面向过程的脚本语言
- 一个用例一个文件



存在的问题

- 代码复用困难
- 难以理解，过大的用例文件
- 不易变化，难以扩展
- 平台影响用例实现效率
- 用例较开发速度低



关键字驱动的用例设计

- RobotFramework
- 表格式语法
- 关键字驱动
- Python
- 面向资源设计

手机接入		
接入成功		

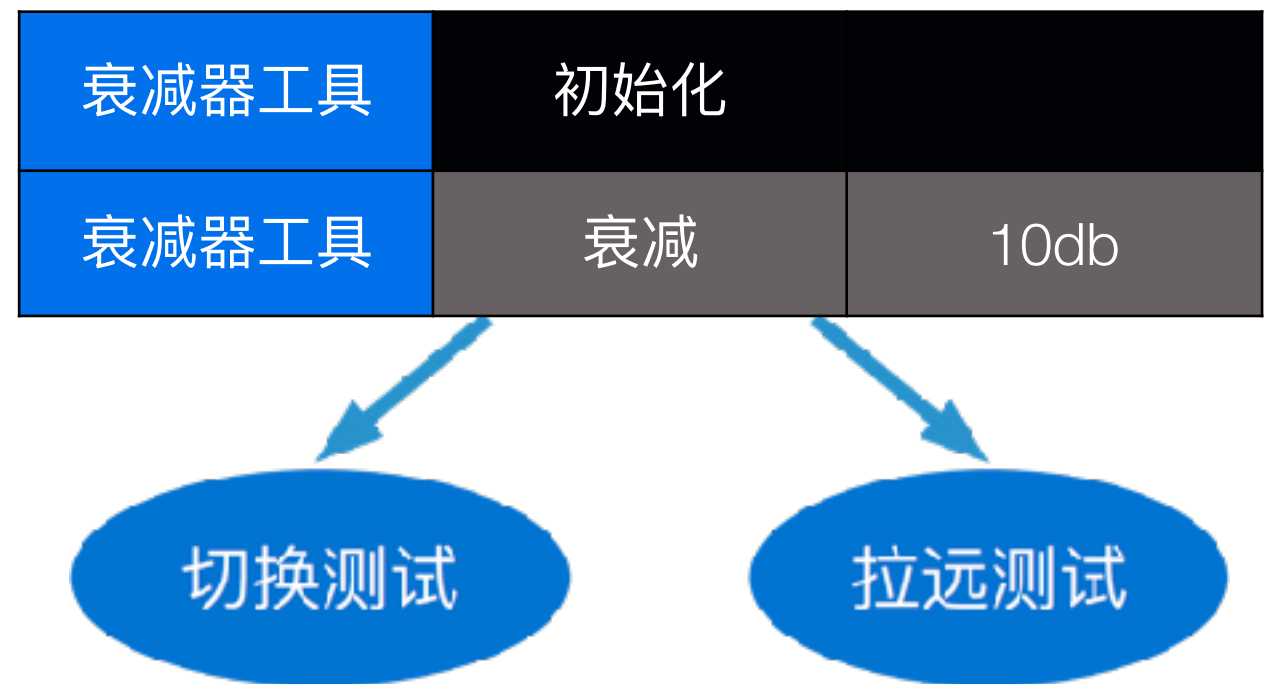
用例示意

单板	创建	
呼叫平台	拨号	

KW:手机接入

未解决的问题

- 抽象层次低
- 人员介入困难
- 稳定性差
- 可扩展性



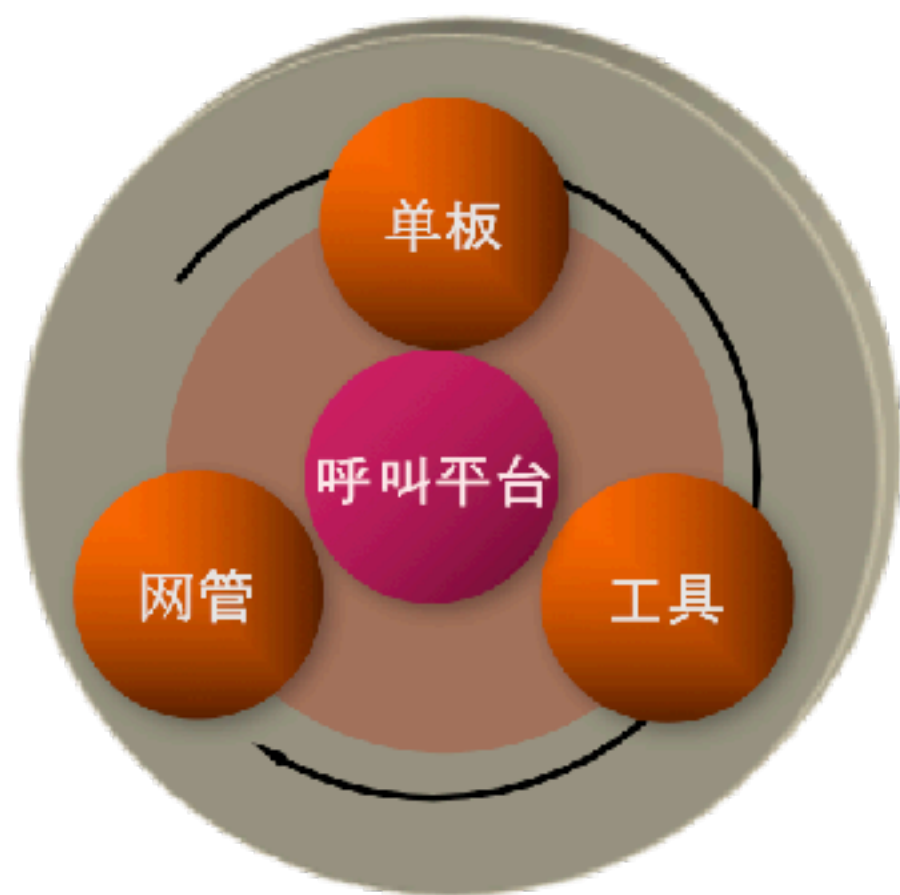
目录

● 自动化测试的窘境

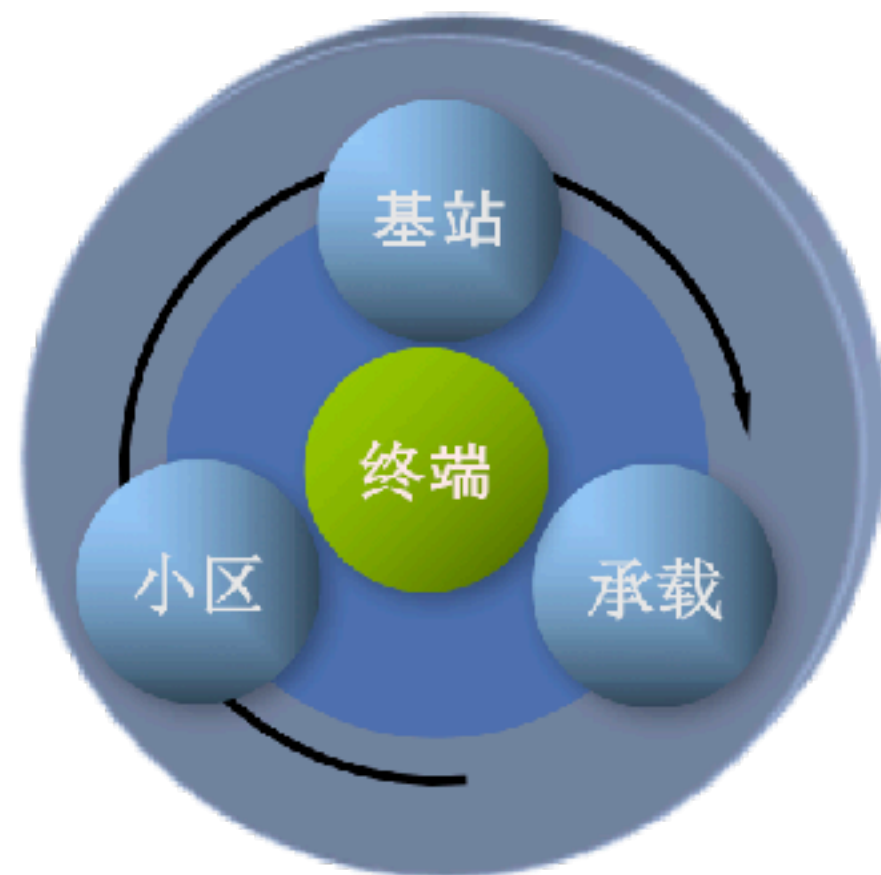
● **DDD驱动用例设计**

● DDD驱动用例实现

认知升级



资源模型

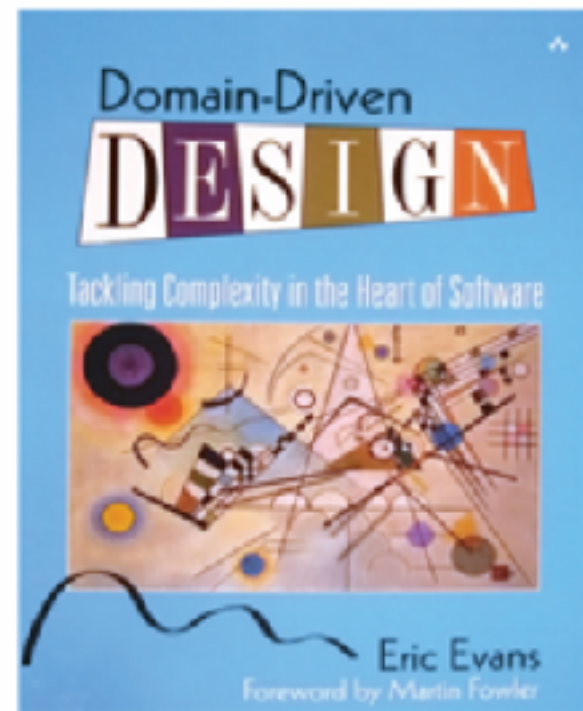


业务模型

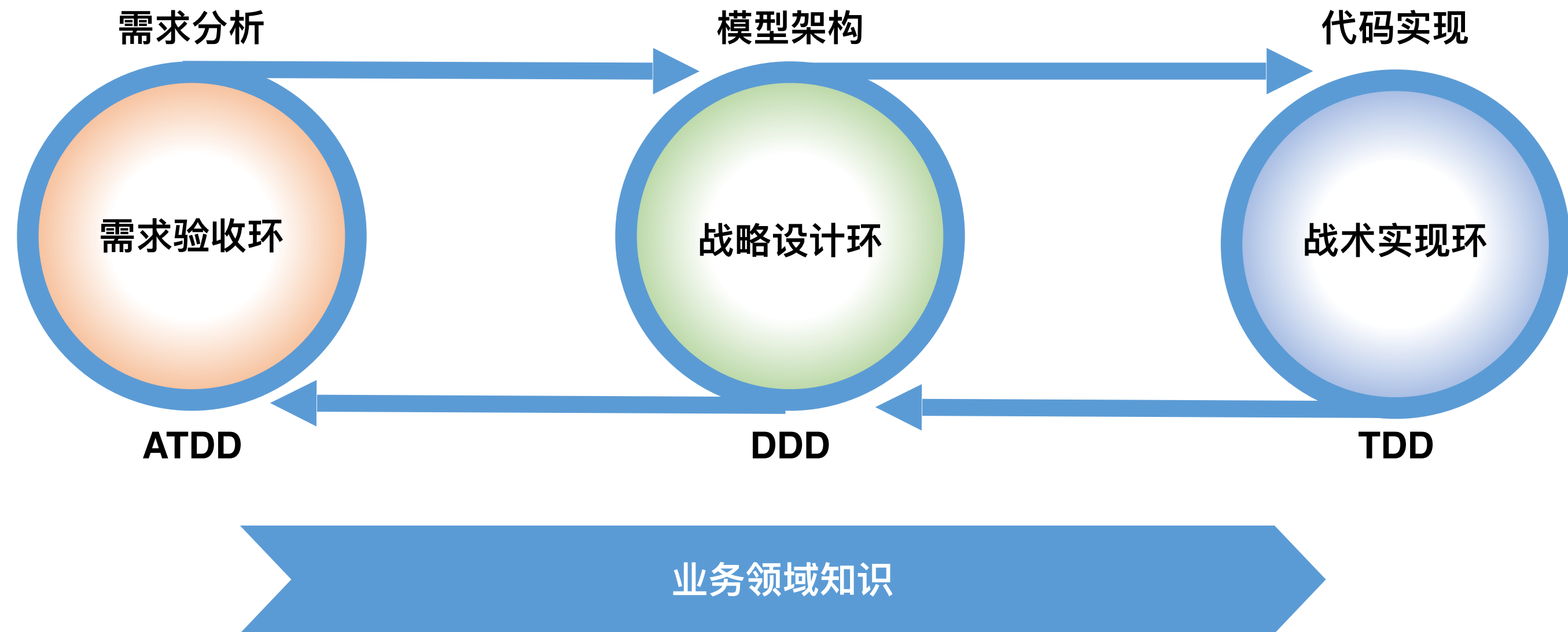
DDD领域驱动设计

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。

Evans



研发闭环

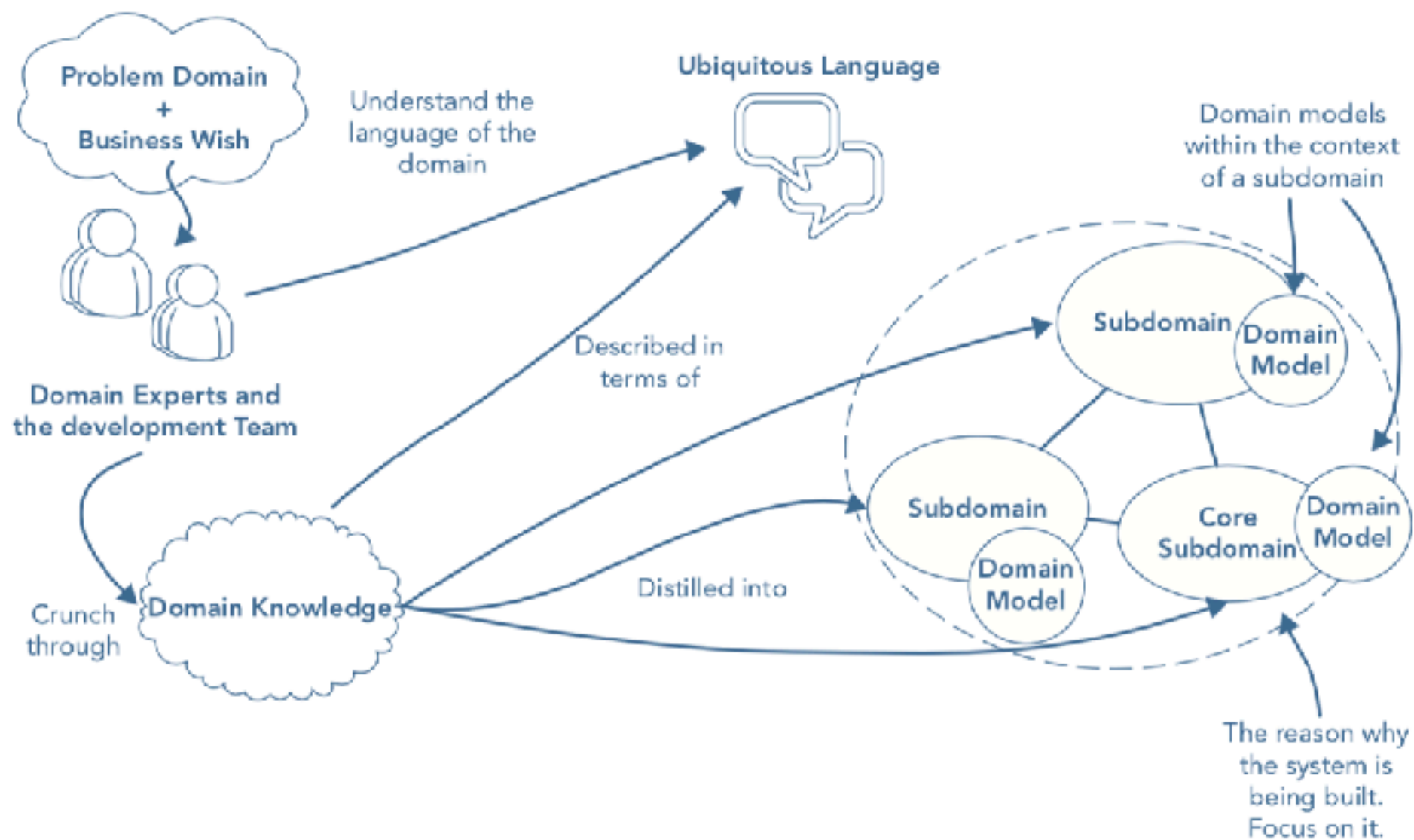


需求&验收



解决用例设计中的全面性和有效性问题

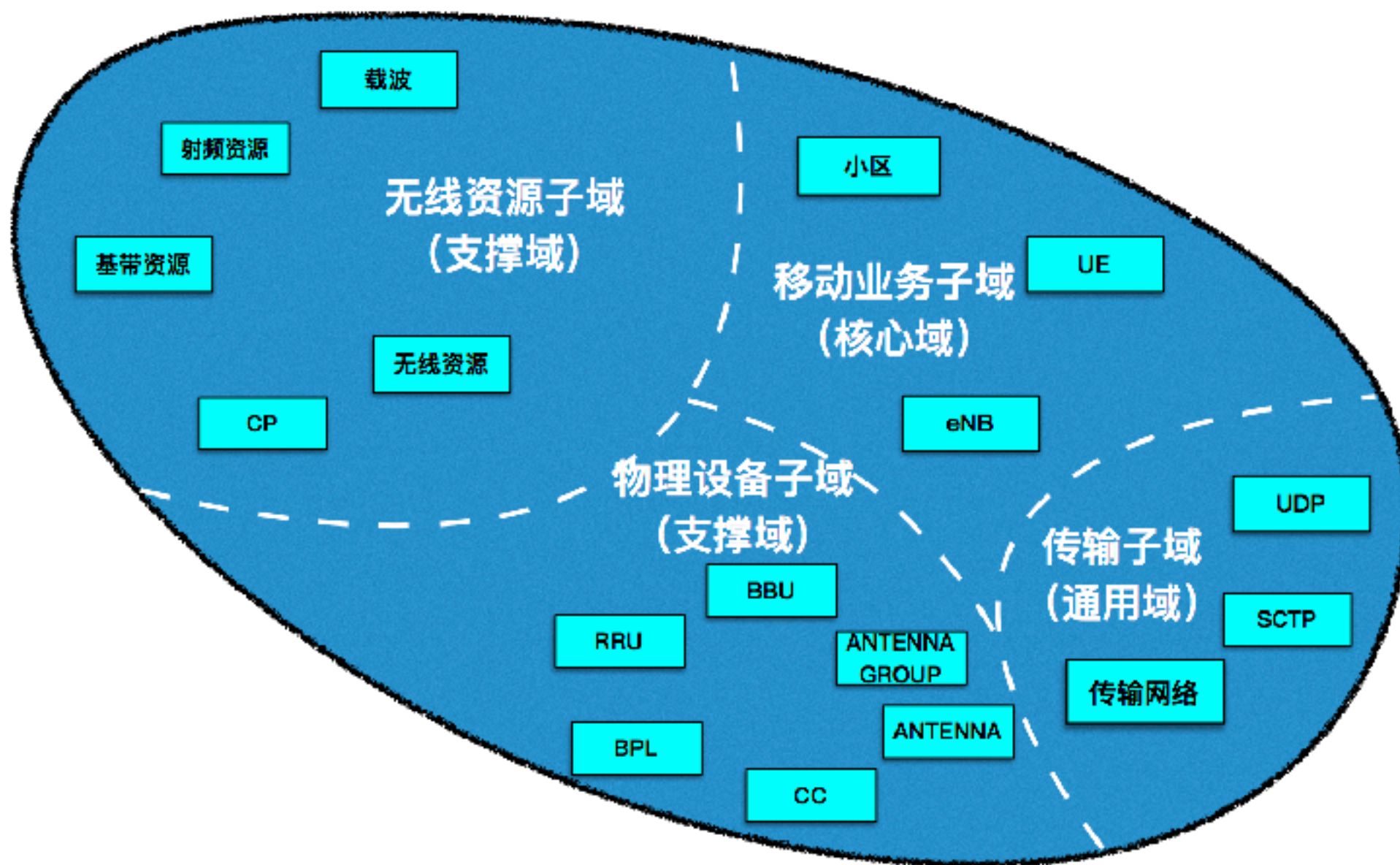
战略设计



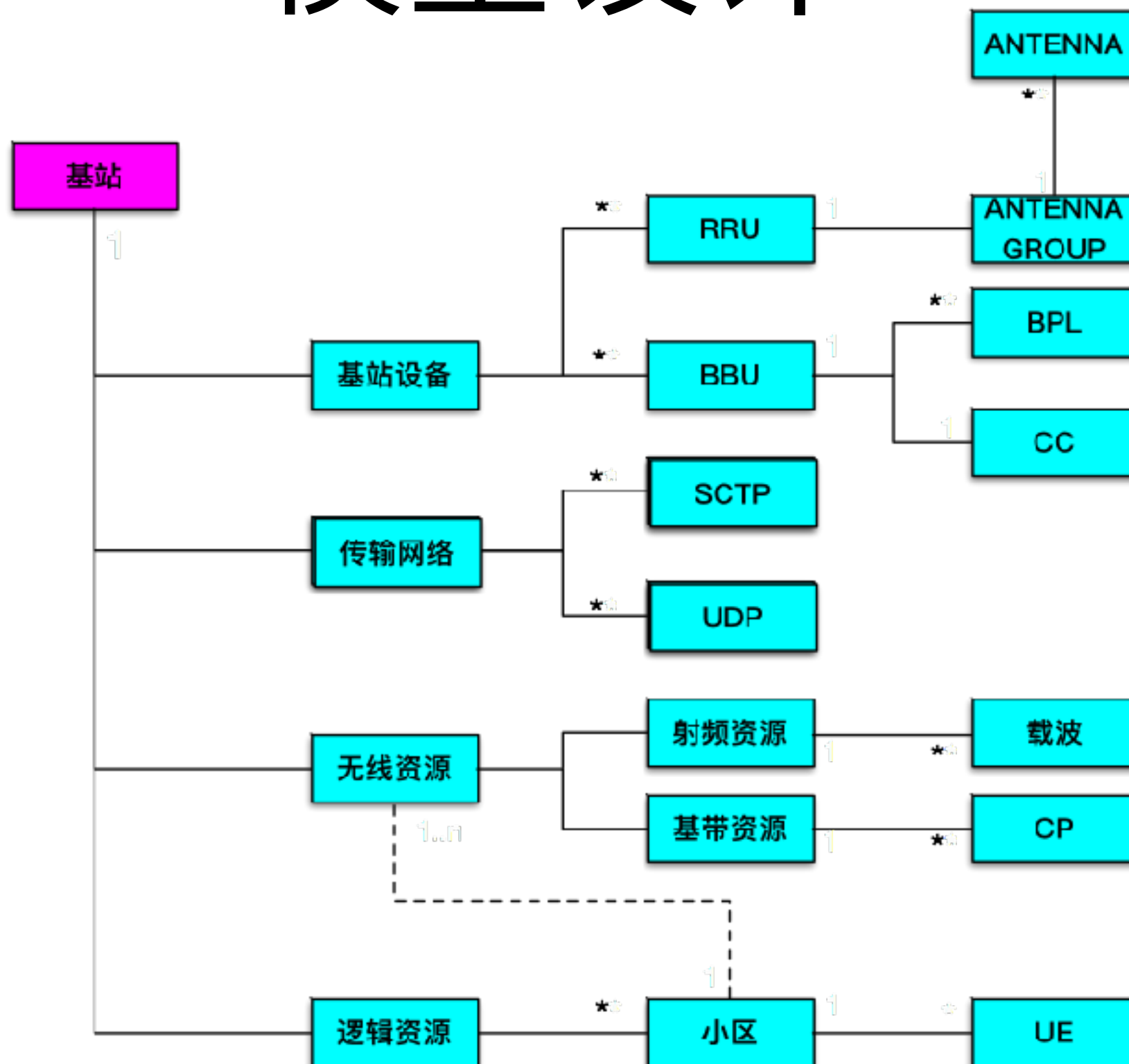
设计流程



识别核心域

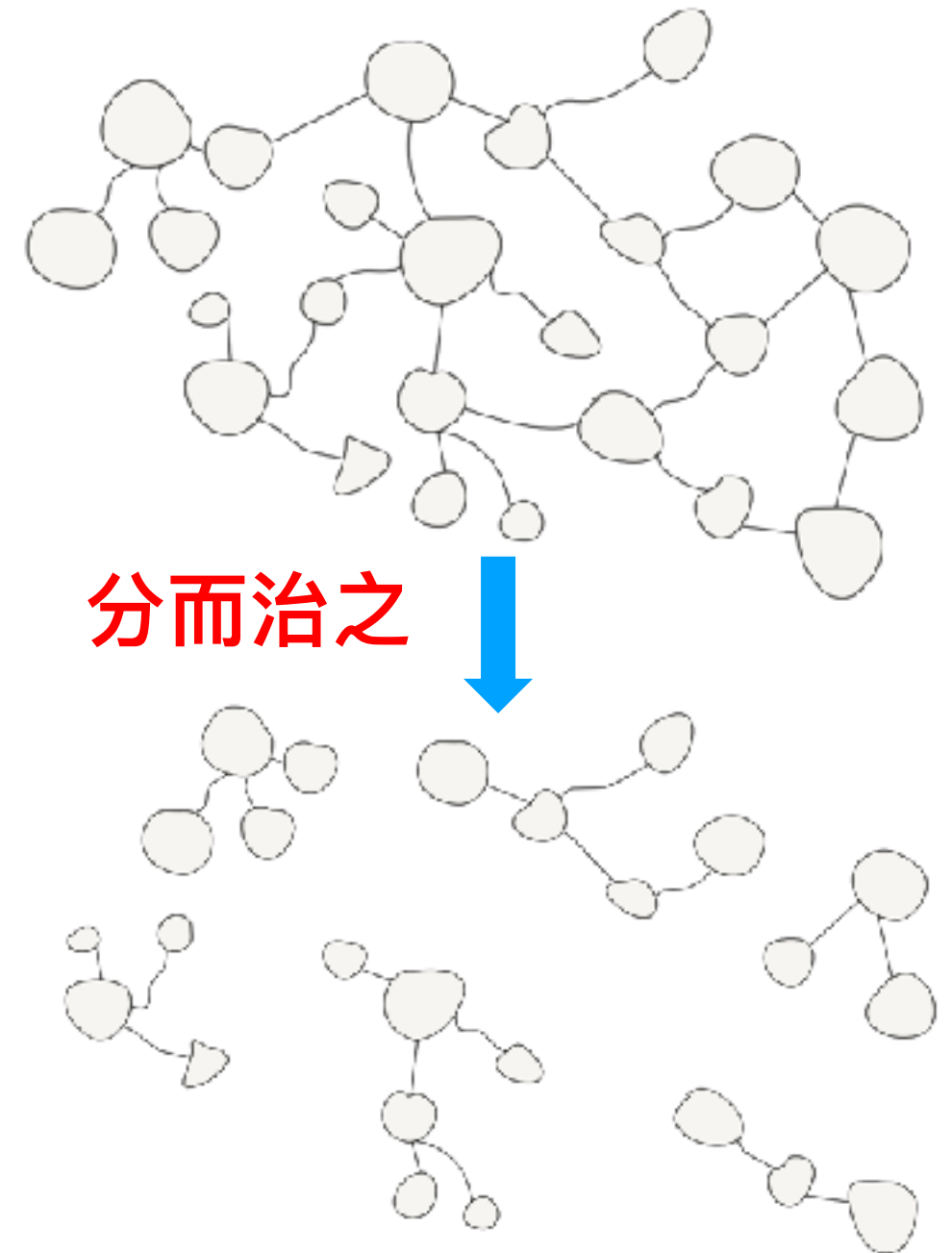


模型设计

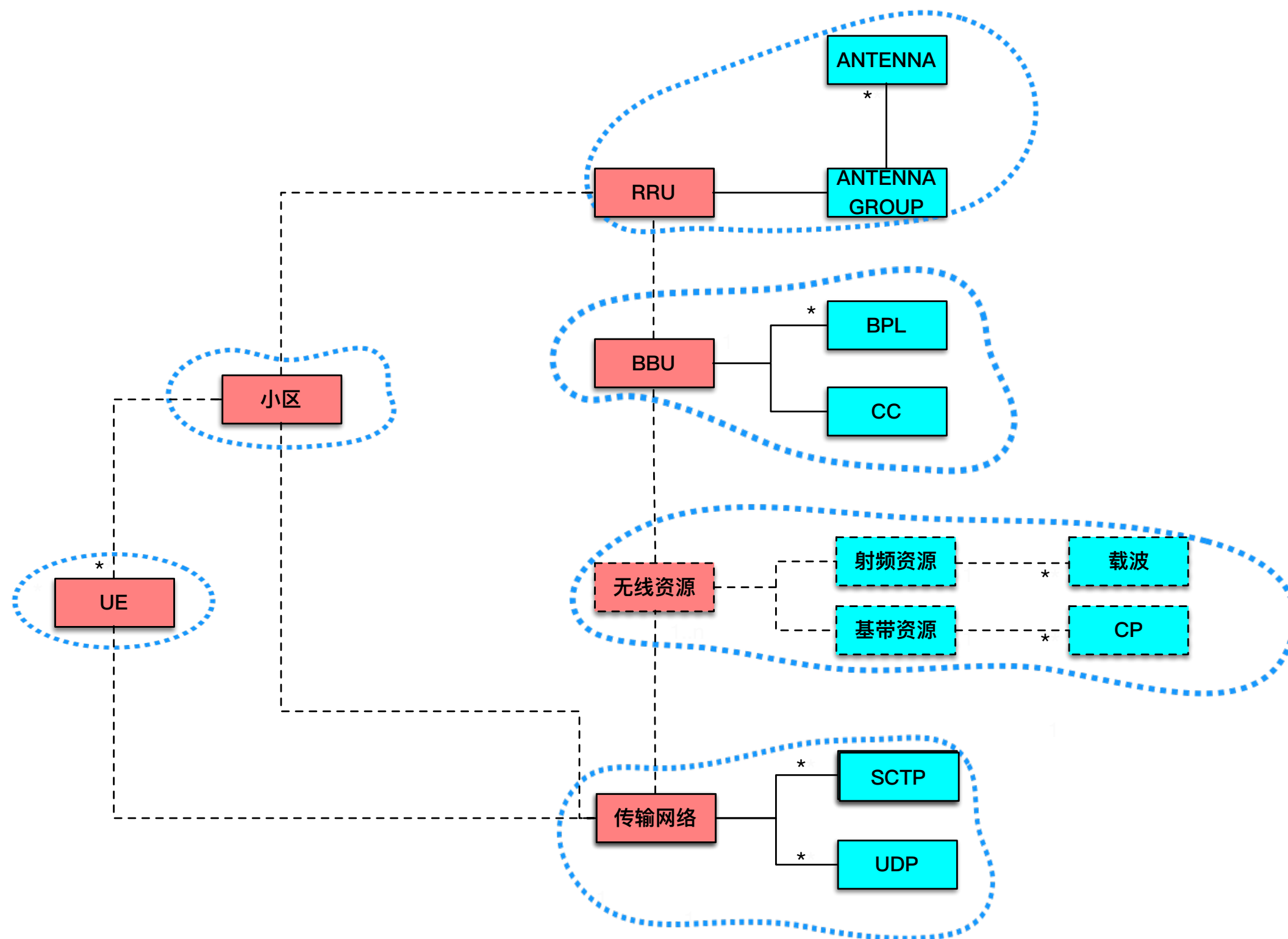


建模思索

- 聚合要尽可能小
- 对聚合内部的实体访问必须通过聚合根（可能与直观认知相违背）
- 聚合内部的实体可以持有其他聚合的聚合根ID

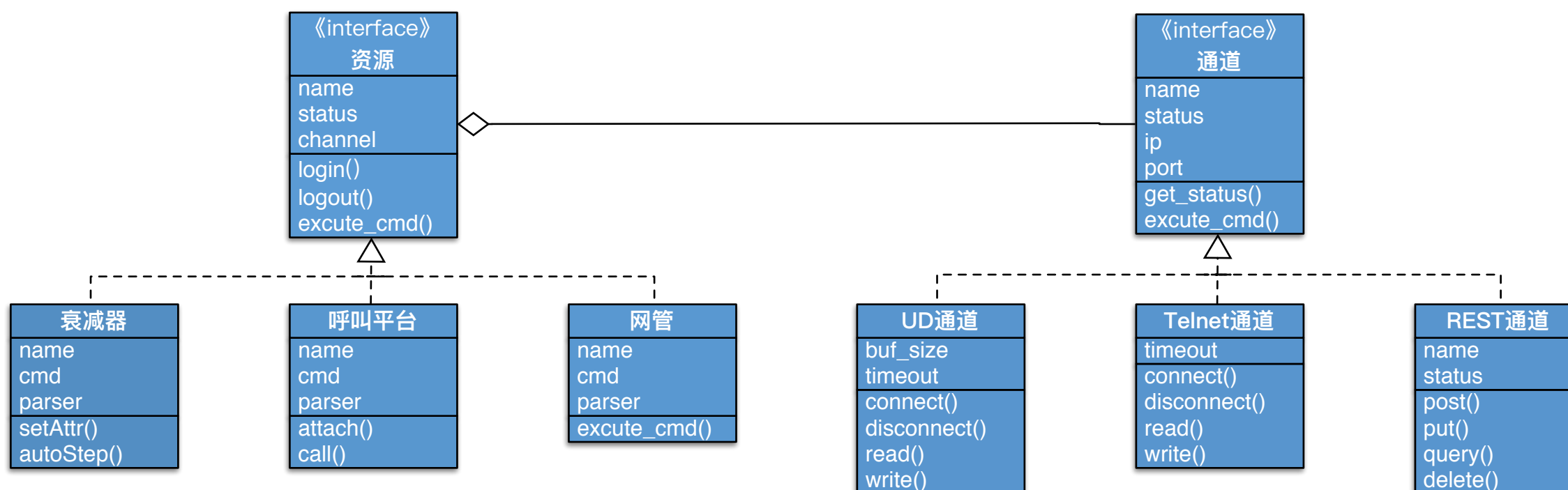


模型精炼

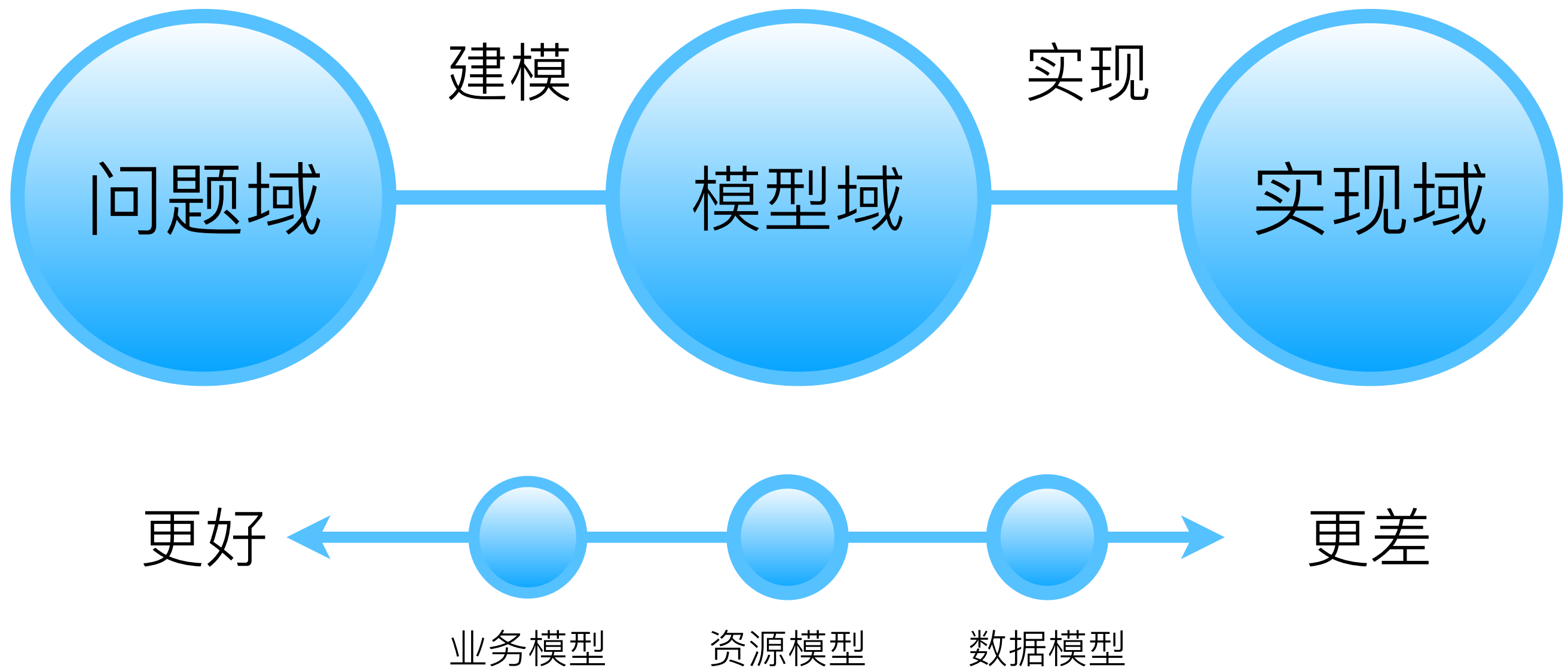


资源模型

- 封装复杂的资源对象操作逻辑
- 分离设备和通道两个变化方向
- 抽象命令处理过程



建模的本质



目录



自动化测试的窘境

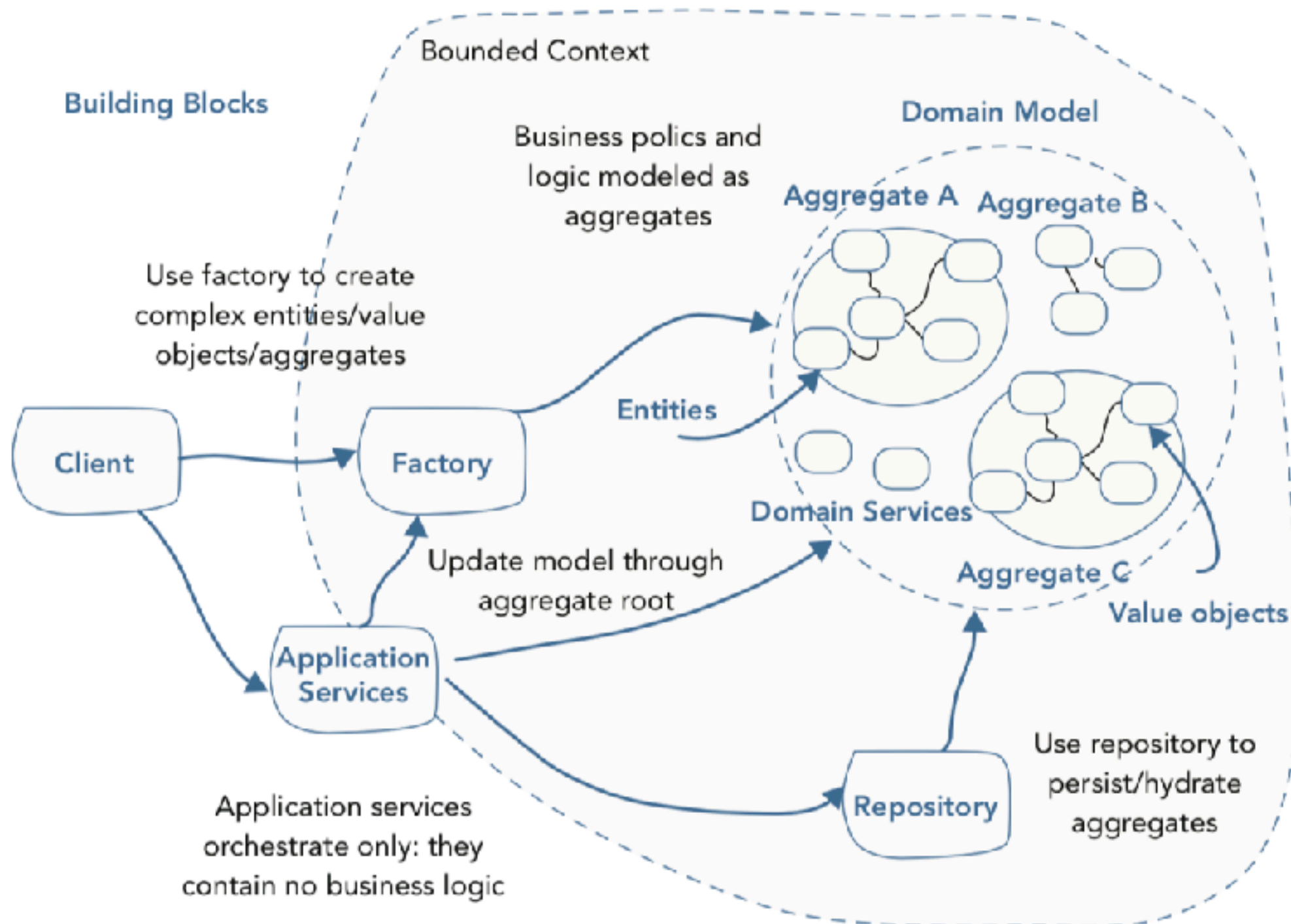


DDD驱动用例设计



DDD驱动用例实现

战术设计



实体

- 实体具有唯一ID
- 完整的生命周期
- 通过ID比较

Focus on who ,not on what.

```
class Entity(object):
    """the entity meta class"""
    __metaclass__ = ABCMeta
    _instance_id_generator = count()

    def __init__(self, id):
        self._id = id
        self._instance_id = next(Entity._instance_id_generator)

    def __eq__(self, other):
        return self._id == other.id

    def __ne__(self, other):
        return not self == other

    @property
    def instance_id(self):
        """the instance id of entity"""
        return self._instance_id

    @property
    def id(self):
        """the property of id"""
        return self._id
```

值对象

- 不具有身份
- 不可变型
- 通过所有属性比较

Focus on what ,not on who.

```
class ValueObject(object):
    ... __metaclass__ = ABCMeta

    def __new__(cls, *args, **kwargs):
        self = super(ValueObject, cls).__new__(cls)

        args_spec = ArgsSpec(self.__init__)

        check_class_are_initialized()
        assign_instance_arguments()
        check_invariants()

        return self

    def __setattr__(self, name, value):
        raise CannotBeChangeException()

    def __eq__(self, other):
        return self.__dict__ == other.__dict__

    def __ne__(self, other):
        return self.__dict__ != other.__dict__

    @property
    def hash(self):
        return hash(self.__class__) and hash(frozenset(self.__dict__.i
```

聚合

- 拥有一个聚合根
- 聚合根拥有全局唯一ID
- 通过聚合根修改聚合内对象
- 销毁聚合根时销毁所有对象

工厂

- 封装复杂的领域对象创建逻辑
- 创建失败需抛出异常通知调用方

```
from abc import ABCMeta, abstractmethod

class Factory(object):
    """Factory abstract class"""

    __metaclass__ = ABCMeta

    @abstractmethod
    def create(self, *args):
        """the create method of factory"""
        raise NotImplementedError
```

仓储

- 存储聚合的容器
- 封装与数据库的交互逻辑
- 接口根据业务需要来构建
- 由Domain定义，实现交给具体类
- 延迟加载是种异味

```
@Singleton
class CellRepository(ObjectRepository):
    def __init__(self):
        ObjectRepository.__init__(self)

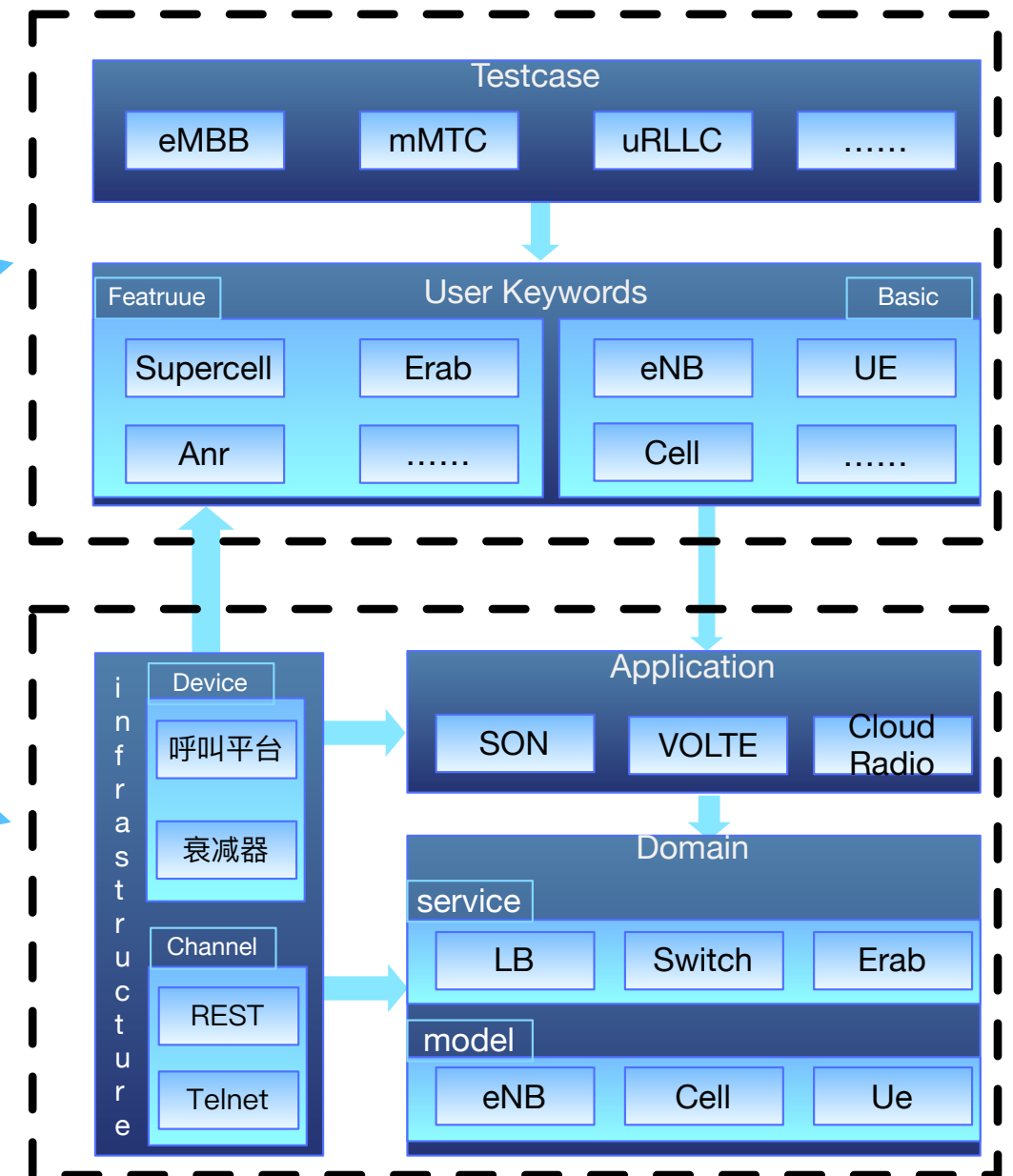
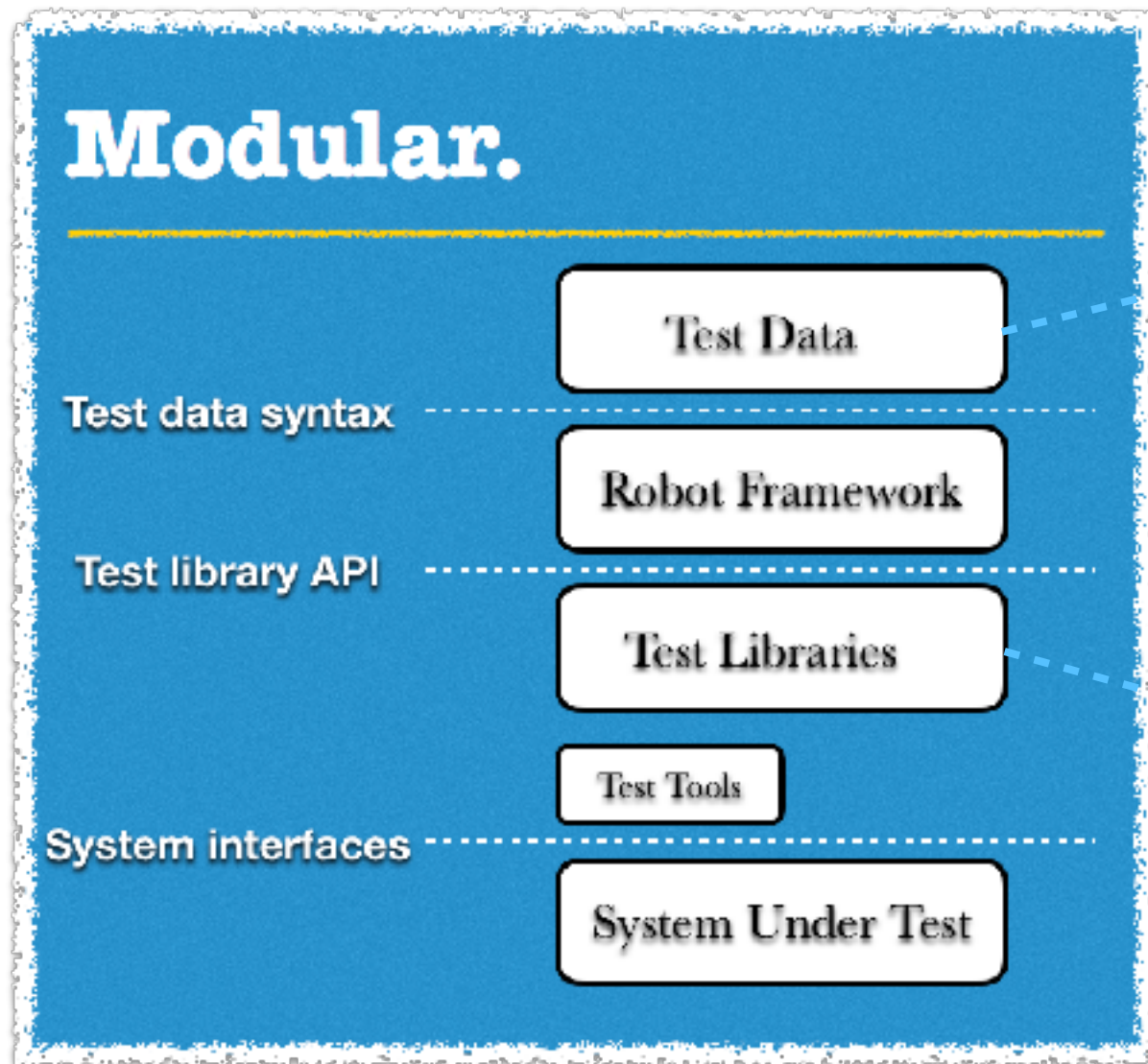
    .....

    def find_all_available_cells(self):
        return filter(lambda x: x.get_cell_state() == 'available',
                       self._objDict.values())
```


领域服务

- 协调多个领域
- 无业务状态
- 封装业务逻辑，避免领域逻辑外泄的操作
- 以动词开头来命名

分层架构



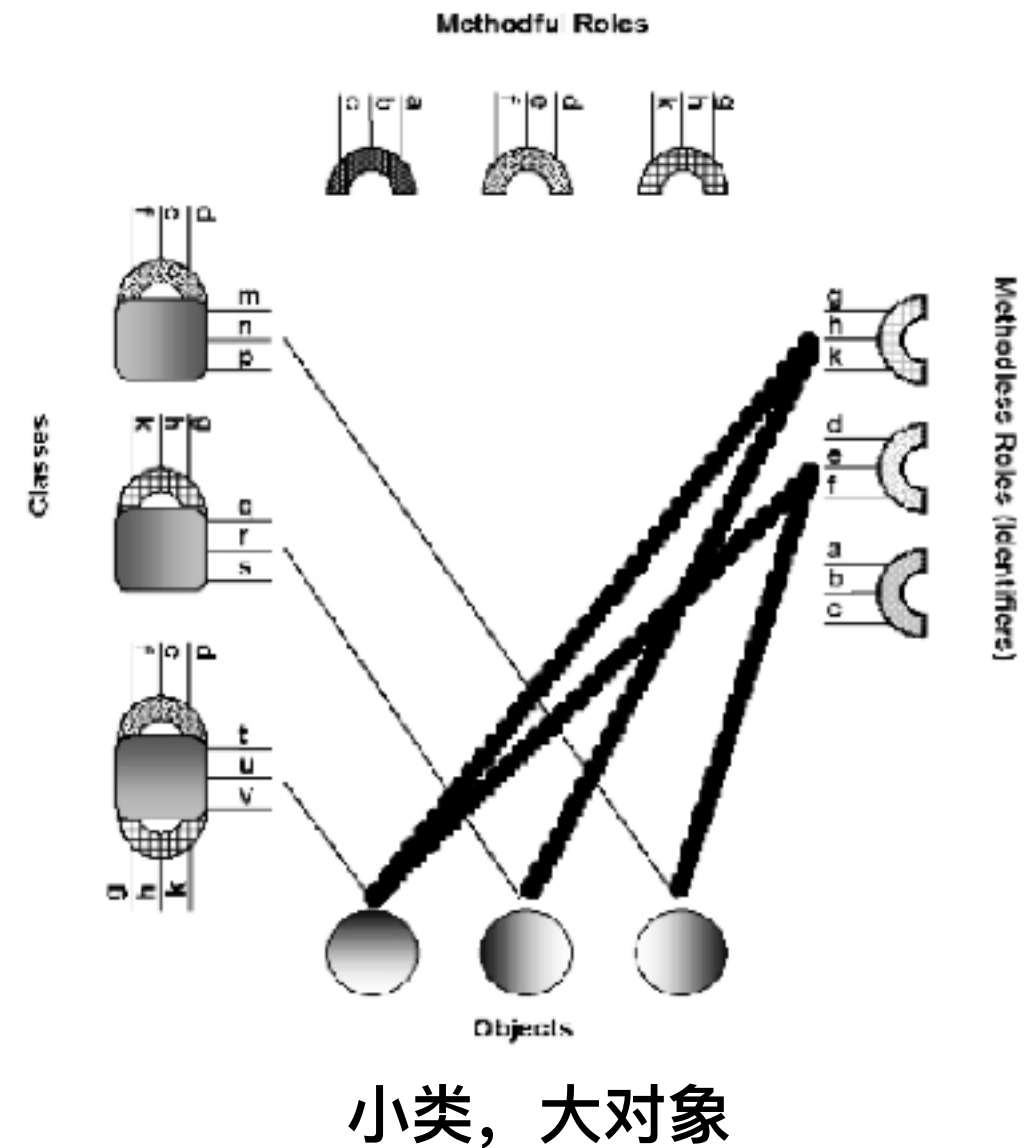
演进过程中的问题

- 充血模型
- 上帝类
- 单场景只使用少量方法



DCI

- 通过给予**系统行为一等公民**的地位来提升面向对象的代码的**可读性**；
- 干净地**分离**高频变化的**系统行为**（描述系统做什么）和低频变化的**领域知识**（描述系统是什么），而非将两者混合在一个类中；
- 帮助理解**系统级状态和行为**，而非仅有对象状态和行为；
- **基于对象**而非基于类的思维方式，前者更接近于人类心智模型，而后者，在面向对象编程语言中常导致过早地屏蔽了基于对象的思考。



整体效果

- 用例开发效率大幅提升
- 在整个公司范围推广基于DDD的测试方法
- 共计实现用例数超过10W

“ST领域”是DDD可全面落地的领域！

相关资源

代码示例：

- https://github.com/kaelzhang81/python_ddd_sample

动态DDD：

- <https://github.com/kaelzhang81/ddd-dynamic>

Thank You

