1. **当前开发模式概要：**
   1. 结构：controller -> service -> repository + framework + view。必须由controller调用service，service调用repository，service和repository可调用framework内容。
   2. 从设计模式来看，类似于SOA，以service为核心，虽然实际开发中大量的业务逻辑泄露到controller和repository中（甚至是view和前端js中）。
   3. 此设计对传统简单的MVC（即框架示例所描述的那样，其中的Model其实是用作ORM和业务逻辑载体的混合体）主要做了如下改动：将Model拆分成负责业务的Service和负责持久化的Repository，舍弃了ORM而采用Active Record。此种设计的一个优点是以下关注点分离：业务逻辑和持久化（基础设施）的分离。
   4. 然而在实际应用中，出现了以下问题：
      1. 业务泄露。本应属于Service的业务逻辑泄露到其他各层中（Controller、Repository、View等），而原本内容丰富的Service反而变成了贫血类。
      2. 职责划分不清。主要表现在以下几点：
         1. 全能Service，主要表现是超多的代码（如vshop的商品和订单的Service代码都在1000行以上）和多方面的功能。例如OrderService，几乎只要是跟订单相关的都在此Service中，而没有进行进一步的精细建模。
         2. 重复功能，表现在（特别是在跨模块时）重复的Service（如MemberService）和重复的方法。这些重复Service大部分地方一样，少数地方有区别。
         3. 一些Service中充斥着各种各样的查询功能（列表和单记录查询），让这个Service看起来很怪异。
      3. 贫血模型。基本都出现在Service层。多出现在查询的地方（列表、单记录）。因为我们默认约定Controller必须通过Service进行读写，而不能直接访问Repository，而实际上很多查询操作只需要Repository直接返回的数据即可，Service不需要做任何操作。
      4. 以技术视角划分模块和系统。这将直接导致4.2.2的问题。例如，所有的作业任务都放在一个系统（如message-center），而作业必然会出现各自业务逻辑（如会员合并），因而同一套业务逻辑必然会出现在多处。
      5. “前后台分离”。此处的分离是指按照前后台站点分离出完全独立的两套系统（如member-center何v-member，虽然商城是在一起，但基本也是两个完全独立的模块，而且v-shop这个名字起得很怪异，估计后本来准备分成两个，后来偷懒了搞成两个模块）。
      6. 经典的“framework问题”。这个问题几乎困扰着全公司人，而且还会继续困扰下去。framework问题的实质是敏捷团队/公司和传统架构思维的阻抗，它实际上正印证着康威定律：组织结构决定着软件架构。该主题将在后面详细讨论。
      7. OO无用论。一般在应聘时，我们都会在简历上写上“良好的OO基础”，但实际上我们骨子里是对OO持有怀疑和抵触的，而且实际开发中也是自觉不自觉地在面向对象框架中进行着面向过程的开发。问题是，为何我们那么抵触OO，那么喜欢过程式开发？另一个问题是，为何我们需要拥抱OO，相比于过程式开发能带来哪些好处？
      8. 二维设计。或者说：“当我拿着MVC这把羊角锤时，全世界都是钉子”。当我们用MVC这一单一设计模式去解决一切问题时，就陷入了二维设计。我们无法立体地看待问题，无论是思考（设计）还是编码都是在平面上进行着（如Controller -> Service -> Repository的形式化调用）。这种思维方式往往导致形式化的流程和约束，而形式化的东西又往往束缚了人们的思想，进而不再去思考。
      9. 耦合与内聚的纠缠。用辩证法的术语说，低耦合和高内聚是矛盾（辩证）统一的，高内聚的内部一定是高耦合的，低耦合的两个东西之间一定是低内聚的。当我们无法辨识出哪些元素之间应当是高内聚，哪些之间又应当是低耦合时，整体的高耦合便产生了（同时整体的低内聚也产生了，如分散在多处的同一套业务逻辑）——“牵一发而动全身”与“牵多发以动一处”。
      10. 对重构的偏见。“推到重来”是每个程序员的梦想，也是很多程序员对重构的理解。之所以出现“重构=推到重来”正是因为以上诸多问题导致的重构的困难性。实际上，重构的精髓在于局部改良。重构应当是改革，而不是革命。纵观千年封建史，一次次的革命换来的不过是一个个的王朝——现实中的重构也往往是这样。（当然改革论是有欠妥的时候，后面再讨论）