乐学 Team 分享系列之

重 构 修改代码的学问

Jun.xiong 2013.3.14

“挖坑”也要系统化进行

Roadmap

- 什么是重构？
- 重构的意义
- Smell Code 与常见重构方法
- 重构与设计
- 重构的正确时机
- 参考资料

任何一个傻瓜都能写出计算机可理解的程序，唯有写出人容易理解的代码，才有望成为一个优秀的程序员。

---Martin Fowler

什么是重构？

代码写好后改进它的设计

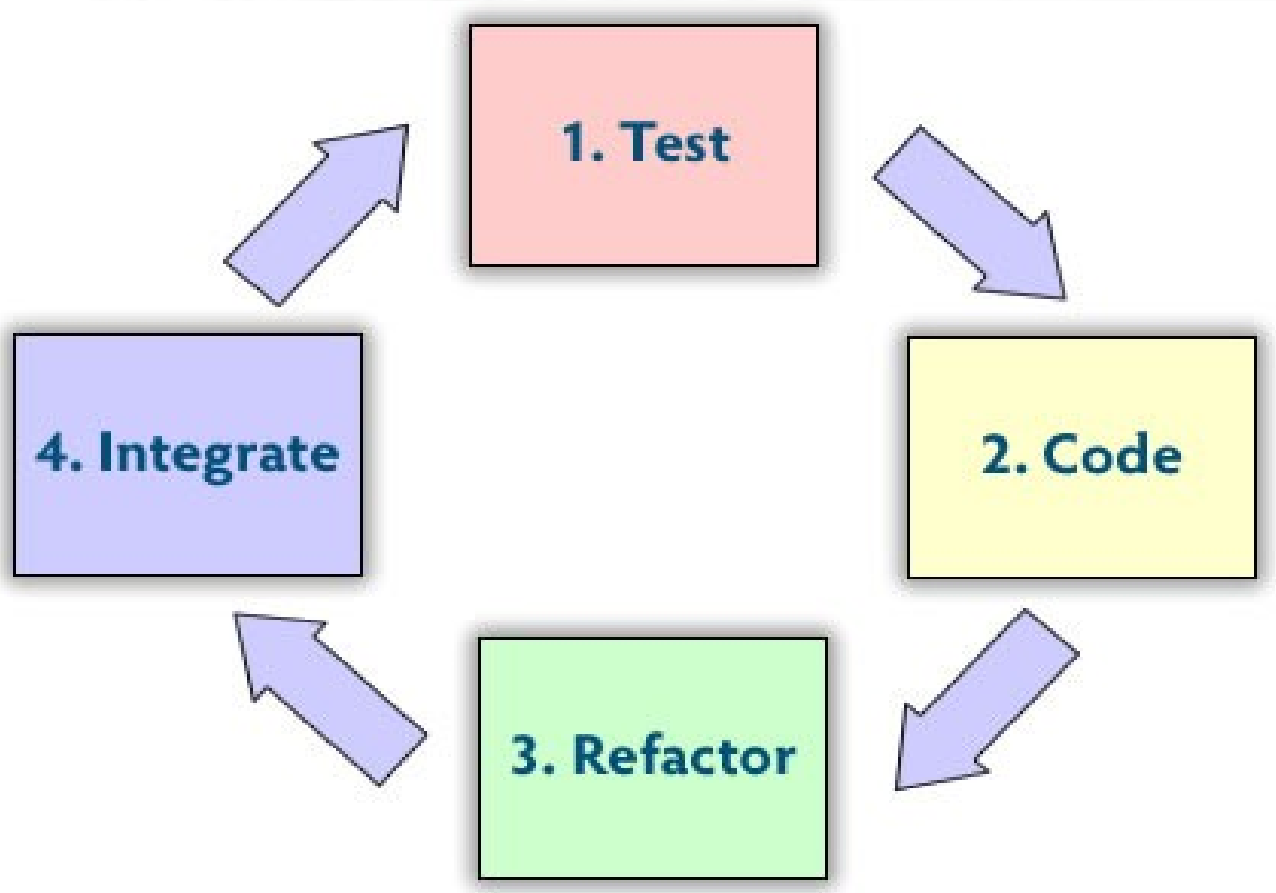
三要素：

调整代码的内部结构 (What)

不改变代码原有功能与行为 (Not What)

提高代码可理解性，降低维护与修改成本 (Goal)

什么是重构？



重构的意义

好代码是修改出来的

“擦干净窗户，你可以看得更远”

软件是有机生命体，设计会腐化，代码会膨胀

重构之道，防患未然．问题代码是债务

如果它没有坏，就别动它？（古老格言）

Smell Code 与重构方法

- 重复代码
  - 代码中的变化扩散，不利于扩展
  - 代码长度增加
  - 修改与阅读难度增加

- Tips: 封装变化，提取方法或类

Smell Code 与重构方法

过大类 / 过长函数

做得太多

提供了客户程序不需要的功能

理解与复用困难

Tips: 单一职责原则 : 一次只做一件事

提取函数与类

Smell Code 与重构方法

过多注释

代码命名不合理

逻辑或算法太复杂

注释与代码修改同步困难

Tips:

注释时机 : 为什么这样做 , 不知道做什么

提取方法 , 规范命名

Smell Code 与重构方法

- 过长参数列表

  4 个以上参数

  不利于变化扩展

  *args, **kwargs 需要额外的注释

Tips: 将参数封装成对象

Smell Code 与重构方法

- 函数依恋 (Feature Envy)

  函数对某 class 兴趣大于自己所处的 class

  > profile.addComment()

  > activity.addComment()

Tips: 将数据与操作这些数据的方法放在一起

Smell Code 与重构方法

- 过多复杂的算法分支

```
def hello():
    ## place one
    if salary <= 2000:
        calTax1()
    elif salary > 2000 and salary <=10000:
        calTax2()
    elif salary > 10000:
        calTax3()
```

```
def execute():
    ## place two
    if salary <= 2000:
        foo1()
        # some other code1
    elif salary > 2000 and salary <=10000:
        foo2()
        # some other code2
    elif salary > 10000:
        foo3()
        # some other code3
```

Tips: Replace switch-case with Strategy

Smell Code 与重构方法

- 不完善的程序类库 (Incomplete Library Class)

  已有包或库功能不能满足需求

Tips:　继承 -Subclassing

　　委托 -Wrapping

Smell Code 与重构方法

Other Tips:

- If-else 卫语句
- 查询与修改分离
- 重构一开始是小步进行的
- 不要对参数赋值

典型重构动作

变量，函数，类，模块等重命名

提取方法

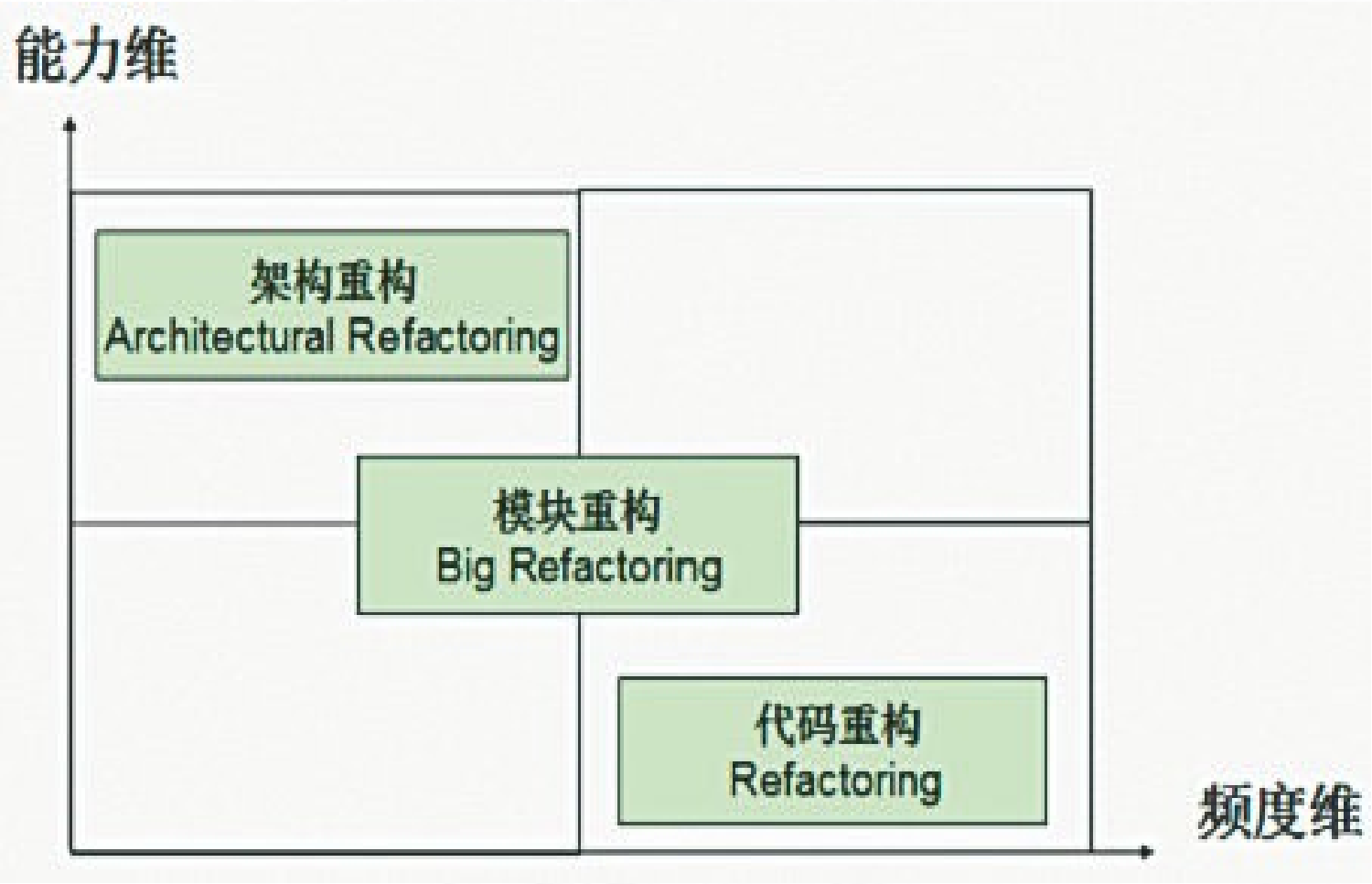
类的提取与分拆

类的扩展

方法重写

原则：高内聚，低藕合，将变化放在一起

重构与设计


能力维
架构重构
Architectural Refactoring
模块重构
Big Refactoring
代码重构
Refactoring
频度维

重构与设计

- 关于建模与模型
  - 抽象与聚焦
  - 模型 3 要素： 属性 , 行为 , 关系
  - Modeling focus is on What, not How
  - Value object VS. Entity Object
- Composition(组合) 与 Aggregation(聚合)
- Delegation(委托)
- 组合 , 聚合 , 委托优先于继承 (Inheritance)

重构与设计

- Architecture(架构)
  - The Most important parts
  - Design Structure
  - The relationships between those parts
- Risks in Software Development
  - Forgetting an important requirement
  - Building the wrong thing
  - Reducing risk is in the requirements phase

重构与设计

- What's Common?
- What's Variable?
- If a design isn't flexible, then CHANGE IT!
- Never settle on bad design

重构与设计

- 开发者需要了解设计原则，学习设计模式
- 设计是不断演化并浮现出来的
- 重构不断，原始设计会腐化
- 设计与性能无法兼顾时
  - 重设计：良好的程序结构与可扩展性
  - 重性能：好的用户访问体验与速度
  - How should we do?

重构与设计

- 开发者不能只会写代码
  - 做计划，对工作的管理与设定目标
  - 学会设计与架构知识
  - 理解需求
  - 学会降低风险
  - 整体的配合与协作

重构的正确时机

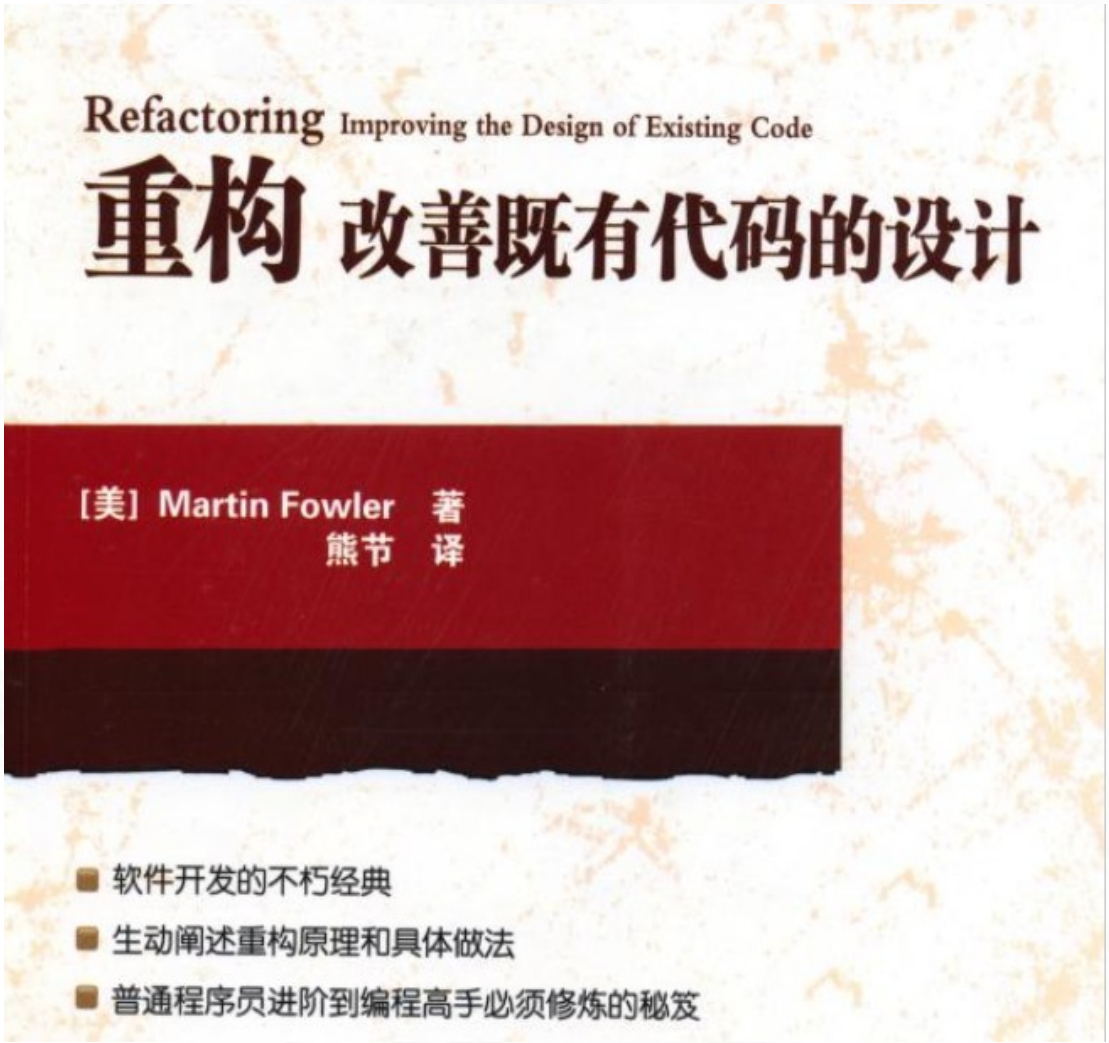
- 前提：足够的时间，人力，财力支持，并对代码进行回归验证
- 添加新特性时现有代码结构让你不方便进行
- 理解不熟悉代码时
- 重复第二次做类似事情
- 修补错误时
- 评审代码时
- 为了改进设计

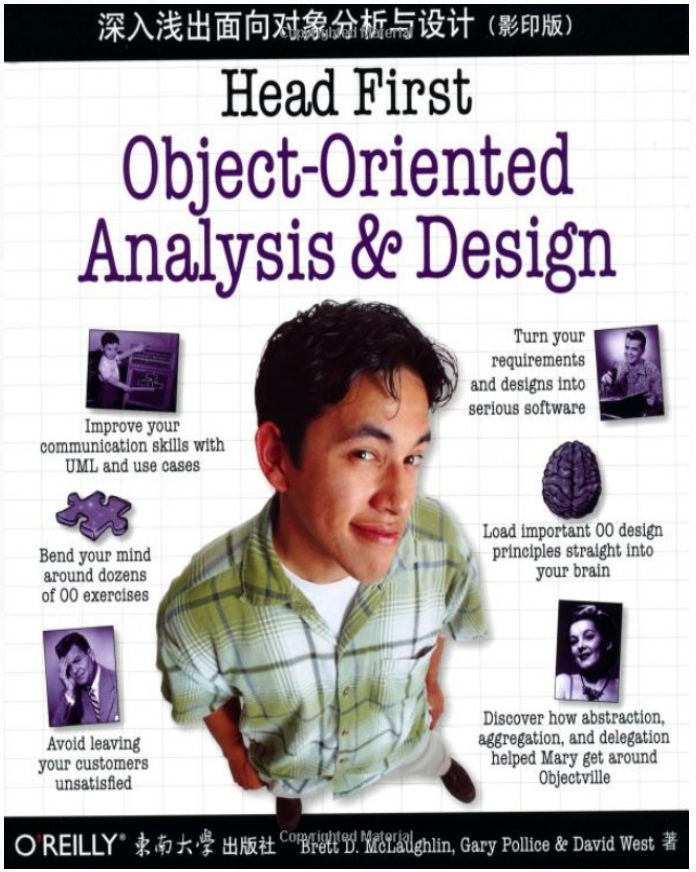
Kent Beck 的两顶帽子

- 添加新特性时
  - 不应该优先考虑重构
  - 迅速正确实现功能
- 重构时
  - 不要添加新功能

不该重构的时机

- 项目已近最后期限
- 逻辑复杂，你没有花时间去分析
- 你不理解前任开发者为什么这样编写
- 团队里的新成员，或新接触项目


Refactoring Improving the Design of Existing Code
重构 改善既有代码的设计
[美] Martin Fowler 著
熊节 译
软件开发的不朽经典
生动阐述重构原理和具体做法
普通程序员进阶到编程高手必须修炼的秘笈



Summary

- 重构三要素
- 重构的意义
- Smell Code 与重构方法
- 重构与设计
- 重构的正确时机

Thanks!

Q & A