# 重构 改善既有代码的设计

重构的节奏: 测试、小修改、测试、小修改、测试、小修改…… 以让重构快速而安全地前进.

* 重构是什么?
  + 使用一系列重构手法,对软件内部结构的一种调整,在不改变软件可观察行为的前提下,提高其可理解性,降低修改成本.
* 为什么要重构?
  + 重构改进软件设计
  + 重构使软件更容易理解
  + 重构帮助找到bug
  + 重构提高编程速度
* 什么时候需要重构?
  + 添加功能时重构
  + 修补错误时重构
  + 复审代码时重构
* 什么地方需要重构?
  + 重复代码
  + 过长函数
    - 如何确定该提炼哪一段代码?
      * 寻找注释
      * 条件表达式
      * 循环
  + 过大的类
  + 过长参数列 -> 传入对象
  + 发散式变化(一个类受多种变化的影响) -> 类与变化一一对应
  + 霰弹式修改(一种变化引发多个类相应修改) -> 类与变化一一对应
  + 依恋情结(函数对某个类的兴趣高于对自己所处类的兴趣) -> 移到感兴趣的类中
  + 数据泥团 -> 总是绑在一起出现的数据应该拥有属于它们自己的对象
  + 基本类型偏执
  + Switch惊悚现身 -> 多态
  + 平行继承体系(每当为某个类增加一个子类,必须也为另一个类相应增加一个子类) -> 让一个继承体系实例引用另一个继承体系的实例
  + 冗赘类(所得不值其身价) -> 让它消失
  + 夸夸其谈未来性(现在用不上的) -> 删掉
  + 令人迷惑的暂时字段(为类里某个复杂算法而定义的字段) -> 提炼这个复杂算法以及相应的字段到一个单独的类
  + 过度耦合的消息链
  + 中间人(某个类过半函数都委托给其他类,这样的过度运用委托)
  + 两个类过于亲密 -> 把两者共同点提取到一个新类
  + 异曲同工的类 -> 提取超类
  + 不完美的类库(库的现有实现不能完成我们希望的工作)
  + 单纯的数据类 -> 检查数据访问权限是否做了合理的设置; 外部访问函数提取到这个数据类里
  + 被拒绝的遗赠(子类不想继承超类的所有函数和数据) -> 给子类建一个兄弟类,把用不到的函数推给兄弟类
  + 过多的注释(因为糟糕的代码而存在的注释) -> 重构代码,删除注释
* 测试(找出现在或未来可能出现的错误)
  + 单元测试(提高程序员的生产率)
  + 功能测试(保证软件能够正常运作)
* 重构手法
  + Extract Method(提炼函数) – 过长的函数或者一段需要注释才能让人理解用途的代码
  + Inline Method(内联函数) – 内部代码和函数名称同样清晰易读, 去掉这些不必要的间接层函数, 直接使用内部代码
  + Inline Temp(内联临时变量) – 找到该临时变量的所有引用点，将它们替换为“为临时变量赋值”的表达式
  + Replace Temp with Query(以查询取代临时变量) – Query为返回该临时变量值的一个方法
  + Introduce Explaining Variable(引入解释性变量) – 比如条件语句中使用简单的变量来代替复杂的判断语句
  + Split Temporary Variable(分解临时变量) – 为多次赋值的临时变量创建相应的临时变量, 保证每个变量只承担一个责任
  + Remove Assignments to Parameters(移除对参数的赋值) – 不要修改方法传入的参数, 可以用一个临时变量取代; 对于引用传递的出参数可以不遵循这个规则, 但尽量用return返回一个值, 如果需要返回多个值, 可以考虑返回一个对象
  + Replace Method with Method Object(以函数对象取代函数) – 将一个大型函数放到一个单独对象中，这样局部变量就成了对象内的字段，然后就可以在该对象中将该大型函数分解为多个小型函数
  + Substitute Algorithm(替换算法) ­– 随着对问题的更多理解，往往会发现更简单的解决方案
  + Move Method(搬移函数) –

## 要了解更多信息并获取 OneNote，请访问 [www.onenote.com](http://go.microsoft.com/fwlink/?LinkID=523891)。