代码整洁之道-态度、技艺与习惯

软件最核心的资产是代码，而对于一个日积月累的产品而言，代码的可维护性意味着未来的可持续性和成本竞争力。从这种角度来说，整洁的代码不只是开发人员的良好习惯，更是产品的关键质量因素。本课程从开发人员的视角，关注产品的核心代码的质量。通过大量的实例剖析讲解：

- 什么是好的代码：可读、可复用、可维护、可扩展。
- 如何编写好的代码：函数、类、变量、判定条件、模式、接口
- 如何对已有的代码进行质量度量
- 如何测试已有的代码
- 如何重构已有的代码
- 如何管理日益复杂的代码

代码管理经验可以切实帮助软件企业降低企业项目开发成本，大面积提高软件工程师编程能力和代码质量管理能力。

初级工程师能够透过大师的眼睛来看待编程，了解编程的 价值观和原则；

具有丰富经验的设计师和架构师可以通过模式进行反思，探究成功实践背后的意义. 把价值观，原则和开发实践结合；

管理者可以制定自己公司的代码管理策略.

编程是一种态度-------编程价值观

一：代码是债务

- 代码的认识---代码就是债务
- 代码是债务，越少越好

- 你拥有的代码越多，添加新内容所要付出的成本就越高
- 通过案例分析让代码库尽可能小的方法：
    - 尽可能创建通用的工具。
    - 删除不用的代码或者特性。
    - 确保项目模块化，并分割成相互没有关联的子项目。
    - 熟悉你经常使用的代码库。
    - 对代码库的规模时刻保持警惕，保持它是小而敏捷的。
- 通过国际研发中心电信计费系统演示代码是债务的思想，10 多年国外研发团队设计与研发第一版本，目前几百人在维护
  通过项目演示通过重构如何减少了一半的代码，维护的人员的减少

项目的失败可能归咎于各种各样的原因。一些项目因糟糕的需求而失败，另一些则由于钱和时间超支了，还有少数单纯是因为糟糕的管理所致。如果我们探究其根本原因，是否会发现所有项目失败的罪魁祸首是糟糕的代码呢？

Bob 大叔坚信糟糕的代码所带来的成本之大足够让一个项目失败。

第二篇： 编程是一种技艺-------编程实践篇

高质量函数

一：高质量函数/过程

- 为什么需要函数
- 函数复杂度度量
- 函数圈复杂度以及度量
- 函数抽象层次-单一抽象层次原则 SLAP(Single Level of Abstrction Principle)
- 函数实现模式之一组合函数(Composed Method)
- 万恶之源一函数过长
- 函数第一原则:是要短小,函数第二原则:是还要短小,函数第三原则:是必须短小
- 函数重构之道一抽取方法(Extract Method)和抽取对象函数
- 函数命名一怎样取好的函数名
- 通过大量项目代码分析,函数的遇到的各种问题,如何编程高质量函数

二：函数代码重复

- 重复的危害
- 强加的重复/无意的重复/无耐心的重复/开发者之间的重复
- 不要重复自己 DRY—Don't Repeat Yourself Principle

- Make It Easy to Reuse（让复用变得容易）
- 魔法数(Magic number)
- 重复性代码(Duplicated Code)
- 接口不同的相似类(Alternative Classes with Different Interfaces)
- 系统分离关注点
- 系统架构的基础通用服务组件
- 通过某项目代码是介绍重复编码问题
- 演示研发过程之中的常见重复问题,以及如何解决

三：函数参数

- 函数参数过长
- 最理想的参数数量是零,其次是一,再次是二,有足够的理由才能使用三个以上参数.
- 函数参数重构之道-引入参数对象(introduce parameter object
- 函数参数的顺序.
- 不要把程序参数当做工作变量/临时变量
- 函数参数模式-collecting parameter
- 函数返回值
- 通过大量项目代码是函数参数问题
- 演示函参数的重构

四：变量

- 变量定义常见的错误
- 变量的数据类型
- 变量的初始化原则
- 变量的作用域
- 变量的持续性
- 变量的绑定时间
- 数据类型和控制结构之间的关系
- 变量的命名
- 全局变量
- 通过大量项目代码演示变量相关问题

演示变量的注意事项

复杂表达式与循环语句

一：条件表达式

- IF/ELSE 语句应该如何编写
- Switch/Case 语句应该如何编写

- 复杂条件表示式的危害
- 过分深层的缩进，或者“嵌套”，已经困扰了计算机界达 25 年之久，并且至今仍然是产生混乱代码的罪魁祸首之一
- 复杂表达式重构之道—引入解释变量/分解条件/抽取方法计算条件
- 表驱动法-多级嵌套 IF 语句的必然之道
- 表驱动法使用总则
- 某保险项目表驱动法应用案例分析
- 通过大量项目代码演示条件表达式编码问题
- 复杂表达式的注意事项,如何解决

二：利用多态解决复杂表达式

- 面向对象多态技术的新认识
- 减少使用 if 语句,重构到多态
- 以 State/Strategy 取代类型代码
- 引入 Null Object
- 以 Command 替换条件调度程序
- 转移聚集操作到 Visitor
- 转移装饰功能到 Decorator
- 通过大量项目代码演示多态可以解决的编程问题

三：防止变异

- 防止变异-如何设计函数,对象,子系.使其内部的变化或不稳定性对其他元素元素产生不良影响
- 接口,多态,数据封装、间接性和标准都是源于 PV
- 开闭原则
- 数据驱动(Data-Driven Design)编程
- 元数据或反射驱动(Meta-data or Reflective)编程
- 复杂业务逻辑-解释器驱动编程
- 案例—通过电信项目介绍如何设计应对变化
- 通过大量项目代码演示多态可以解决的编程问题

四：循环控制

- 选择循环的种类
- 循环控制
- 循环的创建-有内向外
- 循环与数组
- 递归
- Goto 语句
- 控制结构与复杂度
- 案例—通过电信项目介绍如何设计应对变化

通过大量项目代码演示多态可以解决的编程问题

高质量类设计与编码

一：类基础-数据抽象类型(Abstract Data Types)

- 类的基础:抽象数据类
- 需要用到 ADT 的场景
- 使用 ADT 的益处
- 基本类型依赖坏味道
- 数据泥团坏味道
- 案例—通过电信项目介绍数据的抽象
- 通过大量项目代码演示数据抽象类型解决的问题

二：数据封装

- 数据封装
- 数据的访问
- 类的封装
- 通过大量项目代码演示数据封装

三：面向对象设计与编程核心----职责分配

- 单一职责原则
- RDD-职责驱动的面向对象设计方法

四：面向对象的编程

- 上帝类/过大的类--违反单一职责
- 依恋情结-一个方法视乎过于强调处理其他类的数据，而不是处理自己的数据
- 发散式改变
- 散弹式修改
- 消息链
- 中间人
- 不当的紧密性

第三篇： 编程是一种习惯-------管理实践篇

代码改善

一：代码重构

- 重构必然性

- 实际重构遇到的 4 大问题
  - 如何发现重构点
  - 如何去重构(重构方式)
  - 如何知道重构何止截止
  - 如何保证重构的正确性
- 介绍常见的重构技术
- 重构到模式的目录

二：代码测试

- 代码测试
- 代码测试的技巧集锦
- 代码测试的典型错误
- 改善测试过程

三：代码调试

- 代码调试
- 寻找代码缺陷
- 调试的心里因素
- 调试工具

四：修改遗留项目代码的艺术

- 必须修改遗留的代码起因
- 遗留代码修改危险事项
- 如何对依赖代码做测试
- 依赖代码的感知与分离
- 依赖代码修改的接缝技术
- 修改依赖代码的工具
- 降低风险的措施

代码管理

一：代码质量度量

- 代码质量的度量
- 业界其他度量标准
- 通过分析多个实际项目,分别度量相关是否标准

二：代码静态分析工具

- 代码静态分析工具概述
- 以 Java 语言代码静态分析工具为例介绍
- CheckStyle：用于编码标准

- PMD 的 CPD：帮助发现代码重复
- Coverlipse：测量代码覆盖率
- JDepend：提供依赖项分析
- Metric：有效地查出复杂度
- 其他语言相关代码静态分析工具
- 通过案例演示工具在项目之中的应用

三：代码评审

- 代码评审前期准备
- 代码评审的代码量
- 代码评审的检查表
- 代码评审的总结与学习

四：代码管理

- 结合国内多家研发中心的代码管理思路分享

代码质量体系的建立