### **1.Code Review的意识**

* 作为一个 Developer，不仅要提交可工作的代码(Deliver working code)，更要提交可维护的代码(Deliver maintainable code)
* 必要时进行重构，随着项目的迭代，在计划新增功能的同时，开发要主动计划重构的工作项
* 开放的心态，虚心接受大家的评审建议(Review Comments)

### **2.代码评审的方式**

* 开 Code Review 会议
* 团队内部会整理 Check List
* 团队内部成员交换代码
* 找出可优化方案
* 多问问题，例如：“这块儿是怎么工作的？”、“如果有XXX 情况，你这个怎么处理？”
* 区分重点，优先抓住设计，可读性，健壮性等重点问题
* 整理好的编码实践，用来作为 Code Review 的参考

### **3.评审内容**

##### **1.架构/设计**

* 单一职责原则
* 这是经常被违背的原则。一个类只能干一个事情，一个方法最好也只干一件事情。比较常见的违背是一个类既干UI的事情，又干逻辑的事情，这个在低质量的客户端代码里很常见
* 行为是否统一，例如：
  + 缓存是否统一
  + 错误处理是否统一
  + 错误提示是否统一
  + 弹出框是否统一  
    ……
* 代码污染
  + 代码有没有对其他模块强耦合
* 重复代码
* 开闭原则
* 面向接口编程
* 健壮性
  + 是否考虑线程安全
  + 数据访问是否一致性
  + 边界处理是否完整
  + 逻辑是否健壮
  + 是否有内存泄漏
  + 有没有循环依赖
  + 有没有野指针  
    ……
* 错误处理
* 改动是不是对代码的提升
  + 新的改动是打补丁，让代码质量继续恶化，还是对代码质量做了修复
* 效率/性能
  + 关键算法的时间复杂度多少？有没有可能有潜在的性能瓶颈
  + 客户端程序对频繁消息和较大数据等耗时操作是否处理得当

##### **2.代码风格**

* + 可读性
  + 衡量可读性的可以有很好实践的标准，就是 Reviewer 能否非常容易的理解这个代码。如果不是，那意味着代码的可读性要进行改进
  + 命名
  + 命名对可读性非常重要
  + 英语用词尽量准确一点，必要时可以查字典
* 函数长度/类长度
  + 函数太长的不好阅读
  + 类太长了，检查是否违反的 单一职责 原则
* 注释
  + 恰到好处的注释
* 参数个数
  + 不要太多，一般不要超过 3 个

### **Review Your Own Code First**

* 每次提交前整体把自己的代码过一遍非常有帮助，尤其是看看有没有犯低级错误