**代码评审方案文档**

## 引言

代码评审（Code Review）已经被广泛的认可为一种非常好的做法，可以使缺陷发现前移，很多团队也都在进行Code Review工作。但是，投入多少高级别资源进行Code Review比较合适？Code Review采用何种方式的投入产出比最高？不同的场景和对象采用何种Code Review方法和策略？这些问题都没有很好的应对方法。

## 编写目的

Code Review主要目的是能够确保代码库整体的健康程度随着时间的推移而不断改善，提高代码和产品的整体质量。

本文档的愿景也是这样，但这并不是编写此文档的目的，当然也不是为了说明Code Review的重要性和怎么做会更好，而是希望通过适当的规范和标准，建立评审和协作流程以提升当前工作的质量，让Code Review成为开发流程的一部分。并通过实践，根据团队的实际情况进行调整，不断的完善评审和协作流程。

## 代码评审（Code Review）

### 评审标准

适当的规范和标准绝不是消灭代码内容的创造性、优雅性，而是限制过度个性化，以一种普遍认可的统一方式一起做事，提升协作效率，降低沟通成本。希望后续能不断完善该标准：

1. 代码不能损害系统的整体质量；
2. 技术和数据高于意见和个人偏好；
3. 权衡项目进度和建议的重要性，适当放宽要求；
4. 紧急情况下，应该更加注重评审速度和代码的正确性(能否解决当前的紧急情况)；
5. 代码只要能明显改进系统和改善系统的整体运行状况（功能完成性、可理解性、可维护性、可测试性···），即便不够完美，也应该批准；
6. 系统里不想要的功能，即便代码设计的很好，也可以拒绝掉；

### 评审关注点

1. **设计：**设计是否合理，是否适合当前系统
2. **功能性：**代码功能和业务功能是否一致，对用户和维护者是否友好
3. **复杂性：**是否过度设计
4. **测试：**测试是否正确、合理和有用的
5. **命名：**变量、类名等命名是否清晰明了
6. **注释：**注释的可读性，是否清晰有用？（不要写干了什么，而是写为什么这么写）
7. **代码风格**：确保遵循编码规范，但不强制
8. **文档：**如果变更改变了用户构建、测试、交互或者发布代码相关的逻辑，是否更新了相关文档
9. **安全：**接口是否安全

### 评审速度

# 内部开源