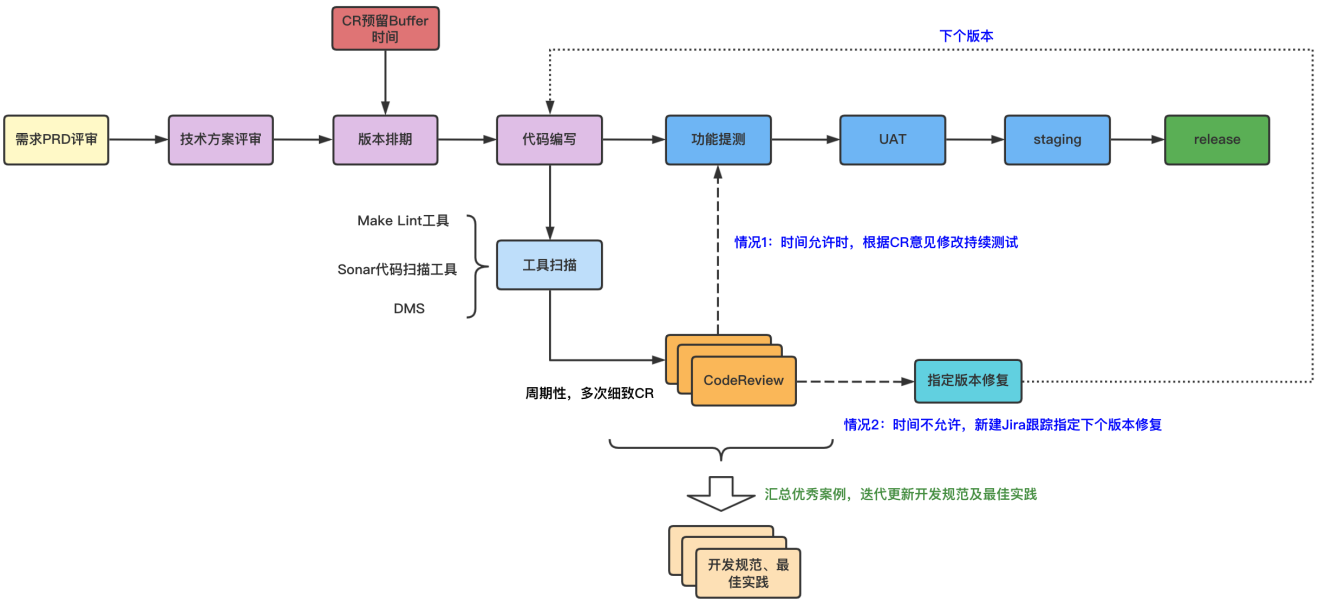
CodeReview最佳实践

1.CodeReview执行流程



我们在执行CR前后需要注意的点：

1. 是否在排期时，预留好了CR的Buffer时间去保证CR的质量
2. 执行CR前，是否已经完成相关工具的扫描
3. CR执行过程中，是否有记录相关问题及修复意见，并安排修复责任人和修复计划

2. 工具使用

2.1 make lint工具

新建本地Makefile文件，创建下面命令

```
lint:
        if command -v golangci-lint >/dev/null 2>&1;then \
                echo "exists golangci-lint"; \
        else \
                echo "not exists golangci-lint"; \
                brew install golangci-lint; \
        fi

        golangci-lint run -v    --skip-dirs="vendor"  --exclude unused,sa1012,asmdecl,S1004,SA5008,SA1029  ./...
```

每次在提交代码前，可以本地执行一下 `make lint` 命令，扫描一下常规错误。

2.2 Sonar扫描支持

2.3 DMS CICD支持

3. CodeReview检查项

Code Review主要检查代码中是否存在以下方面问题：代码的一致性、编码风格、代码的安全问题、代码冗余、是否正确设计以满足需求（功能、性能）等等

3.1 完整性检查

- 代码是否完全实现了设计文档中提出的功能需求
- 代码是否已按照设计文档进行了集成和Debug
- 代码是否已创建了需要的数据库，包括正确的初始化数据
- 代码中是否存在任何没有定义或没有引用到的变量、常数或数据类型 (可在编译中发现，再辅助make lint工具等)

3.2 一致性检查

- 代码的逻辑是否符合设计文档
- 代码中使用的格式、符号、结构等风格是否保持一致

3.3 正确性检查

- 代码是否符合制定的标准
- 所有的变量都被正确定义和使用
- 所有的注释都是准确的
- 所有的程序调用都使用了正确的参数个数

3.4 可修改性检查

- 代码涉及到的常量是否易于修改(如使用配置、定义为类常量、使用专门的常量类等)
- 代码中是否包含了交叉说明或数据字典，以描述程序是如何对变量和常量进行访问的
- 代码是否只有一个出口和一个入口（严重的异常处理除外）

3.5 可预测性检查

- 代码所用的开发语言是否具有定义良好的语法和语义
- 是否代码避免了依赖于开发语言缺省提供的功能
- 代码是否无意中陷入了死循环
- 代码是否是否避免了无穷递归

3.6 健壮性检查

- 异常处理和清理（释放）资源
- 代码是否采取措施避免运行时错误（如数组边界溢出、被零除、值越界、堆栈溢出等）

3.7 结构性检查

- 程序的每个功能是否都作为一个可辩识的代码块存在
- 循环是否只有一个入口

3.8 可追溯性检查

- 代码是否对每个程序进行了唯一标识
- 是否有一个交叉引用的框架可以用来在代码和开发文档之间相互对应
- 代码是否包括一个修订历史记录，记录中对代码的修改和原因都有记录
- 是否所有的安全功能都有标识

3.9 可理解性检查

- 注释是否足够清晰的描述每个子程序
- 是否使用到不明确或不必要的复杂代码，它们是否被清楚的注释
- 使用一些统一的格式化技巧（如缩进、空白等）用来增强代码的清晰度
- 是否在定义命名规则时采用了便于记忆，反映类型等方法
- 每个变量都定义了合法的取值范围
- 代码中的算法是否符合开发文档中描述的数学模型

3.10 可验证性检查

- 代码中的实现技术是否便于测试

3.11 可重用性

- DRY（Do not Repeat Yourself）原则：同一代码不应该重复两次以上
- 考虑可重用的服务，功能和组件
- 考虑通用函数和类

3.12 可扩展性

- 轻松添加功能，对现有代码进行最小的更改。一个组件可以被更好的组件替换

3.13 安全性

- 进行身份验证，授权，输入数据验证，避免诸如SQL注入和跨站脚本（XSS）等安全威胁，加密敏感数据（密码，账号信息等）

3.14 性能

- 使用合适的数据类型，
- 懒加载，异步和并行处理
- 缓存和会话/应用程序数据
- 海量数据处理的场景是否考虑妥当
- 避免for循环操作db
- 高QPS接口避免使用反射（包括deep copy）

注：以上检查项可作为团队CodeReview检查项的一个参考项，各业务线同学可酌情增减。

4.CodeReview步骤

1. CodeReviewer组织会议，邀请相关人员参会；
2. 由代码Owner 依次讲解自己负责的代码和相关逻辑，可以从Web层 → Service层 → DAO层；
3. 代CodeReviewer在此过程中可以随时提出自己的疑问，同时积极发现隐藏的bug；由记录员对这些bug记录在在案， 可参考 CR模板， 需要描述清楚问题，有截图说明，有修复意见，修复责任人和修复版本计划；
4. 务必关联Task中的Code Review模块信息，粘贴好CR链接，后续技术委员会评审会以此作为统计；

5.事后跟踪

1. 问题点的难易程度以及影响的范围；
2. 明确解决问题的责任者和问题点修正结果的确认；
3. 明确解决问题点的版本和时限；
4. Leader在MR时，double check改动是否到位；

6.正向闭环

按月定期执行脚本，过滤各个组的Task中关联的CodeReview连接进行汇总统计，评审出优秀的案例，补充到开发规范和最佳实践中，并以版本的方式，有节奏的邮件周知各各个业务线团队。