**1 目的**

Code Review是一种在开发阶段用于发现与修复源码中的缺陷的手段，意在提高软件的质量与开发人员的技巧，Code Review主要注重几个方面的审查：

* 代码的规范性、易读性和一致性，检查是否满足组织级的编码规范。好处：一致、易懂的代码可以降低后期的维护成本，并能够加速新开发人员的项目融 入速度；
* 代码功能逻辑检查，确认功能是否满足要求，是否存在逻辑上的错误。好处：首先，能够早期的发现BUG，降低软件修复的整体成本；其次，能够帮助 初级的开发人员提升经验；最后，Code Review也是项目中开发人员彼此交流的一种重要形式；
* 代码非功能性检查，主要体现在性能与可扩展性，非功能性的检查主要依赖高级开发人员的经验与能力。

**2 适用范围**

研发所有软件开发项目。

**3 基本原则**

按照目前业界的标准，Code Review需要遵循如下标准：

* 每次Review的总代码行不超过400行，建议在200以内；
* 每次Review时间不要超过90分钟，建议在60分钟内；
* 每小时Review的效率控制在300行以内；
* 对外接口必须Review；
* "维护支持系统"的问题、致命Bug的修改代码必须Review；
* 发现的缺陷不进入BUG系统，统一走代码共享方式解决；

**4 角色职责**

* **Dev Leader**：负责制定CodeReview计划，安排CodeReview活动职责分工，组织进行 CodeReview，确保按CodeReview过程和规范执行。
* **Dev**：依据CodeReview计划和CodeReview过程和规范执行CodeReview活动。

**5 CodeReview流程**

Code Review的执行流程主要包含四个阶段：

* **准备阶段**，也可以称为计划阶段，在这个阶段，主要工作是排定人员计划（例如，哪几个人彼此交叉Review，哪些人负责哪些模块 的Review等）、排定Review会议计划（如果需要）、简要的设计和思路说明等等；
* **审查阶段**，这个阶段主要是代码提交和执行Review的工作，在这个阶段会产生代码的缺陷，并通过不同的途径（视不同 Review方式有所区别）反馈给代码Owner；
* **改写阶段**，代码Owner接收到众多的缺陷，进行代码修改，并将缺陷状态更正，提交代码库；
* **完成阶段**，缺陷提交者确认缺陷修复的正确性后。  
  以上四个阶段会循环出现，循环的频率视不同的Review方式有所不同，从每天、每周、每2周（Base Line：2周必须执行一次）不等。

**6 评审方式**

目前我们主要执行的有三种Code Review方式，他们分别适应于不同的软件开发阶段：

* **交叉审查**：是频率较高的Review，比较适用于代码产生的高峰期（编码阶段）；
* **代码会审**：是以会议形式组织的评审，较适合于关键模块的审查；
* **Email Pass**：是有严格的代码提交控制要求的，比较适合于稳定阶段（系统测试阶段）。

**6.1 交叉审查**

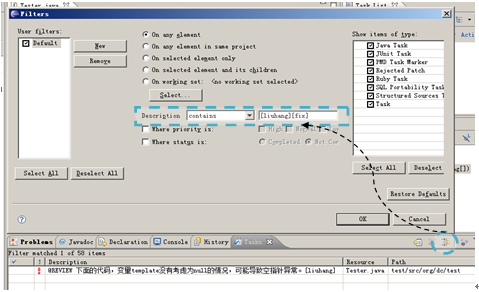
交叉Review是项目开发人员日常的工作之一，视项目不同，每天花30-60分钟（Base Line：不少于30分钟）不等的时间进行开发人员之间的交叉代码审查。交叉审查的流程如下：

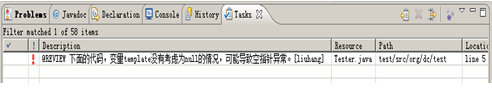
1. **准备阶段**：项目（Role）Dev Lead（或者开发经理）根据项目的模块与开发人员资源情况，排定交叉Review的计划，输出的内容如下所示（示例）：

|  |  |  |
| --- | --- | --- |
| **开发人员** | **负责Review项目/模块** | **源码负责人（可选）** |
| 张三 | eos-common-\* | 李四 |
| 李四 | eos-sca-\* | 赵六 |
| 王五 | bps-mananger-\* | 张三 |
| 赵六 | bps-configuration-\* | 王五 |
| **Review时间：每天早晨9:00-9:30进 行Review** |  |  |

1. **审查阶段**：（Role）开发人员根据计划安排，每天早晨（使用Eclipse或者EOS Studio）同步（PS：不是直接更新，是同步）自己需要Review代码，在Eclipse的同步视图中，查看源代码的变更情况，并同时 Review，提出发现的缺陷（问题）。提出缺陷的方式，是直接在相应的代码中增加注释，注释规格如下所示，包含：**@Review 问题描述[问题发现人] {问题修改人}**
2. //@REVIEW 变量template没有考虑为null的情况，可能导致空指针异常。[zhangsan]{lisi}

填写注释以后，负责审查的开发人员将相应代码提交源码库。

1. **改写阶段**：（Role）开发人员更新自己负责开发的代码，通过Eclipse的Tasks视图，查看属于自己的代码缺陷描述，如 下图所示：  
     
   双击其中Tasks视图中的任何缺陷描述，进行相应的代码修改，修改完成以后，在原来的注释后面增加{问题修改人}[fix]表示，如下所示：
2. //@REVIEW 变量template没有考虑为null的情况，可能导致空指针异常。[zhangsan]{lisi}[fix]

**完成阶段**，(Role)负责Review的开发人员，检查修改，并确认正确。如果不正确，就再次进入改写阶段。  
查找需要自己确认的修改方法为，在Tasks视图列表中，使用过滤功能，如下图所示：  


**6.2 代码会审**

代码会审主要用于开发阶段的重点模块集中Review，它采用的是正式的会议Review。Review的频率视不同的项目有所不同（Base Line：2周至少一次Review）。

1. **准备阶段**：项目Dev Lead（或者开发经理）根据开发计划，制定相应的会审计划，计划包含会审主题、参与资源、主持人等，输出如下计划表，并在后期跟踪执行；

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **时间** | **地点** | **主题** | **组织者** | **记录者** | **目标缺陷数** |
| 1 | 4/17 19:00-21:00 | M2 | XXXX | 张三 | 李四 | 8 |
| 2 | 4/17 19:00-21:00 |  |  |  |  |  |

1. **审查阶段**：具体某一次会审，负责组织的开发人员（也是当次会审代码的Owner），提前一天发出会审通知（会议通知）和当次会审 主题的简要设计思路，在会审过程中，顺着某个主线，给参与会审的其他开发人员讲解代码，其他开发人员可以随时提出问题，确认是缺陷的话，负责记录的开发人 员将缺陷"按照交叉审查方式，向相应的代码中加入注释"，例如，李四组织Review，王五负责记录，如果张三在Review过程中发现一个代码A类中缺 陷，王五随后在A类中直接增加注释，并注明修改人，如下：
2. //@REVIEW 变量template没有考虑为null的情况，可能导致空指针异常。[zhangsan]{lisi}

增加注释以后，王五直接提交代码库。

1. **改写阶段**：会议结束以后，负责组织的开发人员，将源码更新，按照"交叉审查"的方式，修改代码，然后增加{修改人}[fix]注 释，提交源码库；
2. **完成阶段**，与"交叉审查"相同。

**6.3 Email Pass**

Email Pass方式的审查主要用于项目的稳定阶段（系统测试阶段），他主要用于检查进入系统测试阶段以后，BUG或者JIRA问题修复的代码质量。视项目的不 同，需要进行代码审查的范围有所不同，BUG从一般到致命（Base Line：代码封闭以后的严重以上级别BUG必须通过Email Pass的Code Review，JIRA问题可以根据项目具体确认，推荐未P0级问题需要Code Review）。

1. **准备阶段**：项目（Role）Dev Lead（或者开发经理）根据项目的开发人员资源情况，排定交叉Review的计划，输出的内容如下所示（示例）：

|  |  |
| --- | --- |
| **开发人员** | **BUG/JIRA修改人** |
| 张三 | 李四 |
| 李四 | 赵六 |
| 王五 | 张三 |
| 赵六 | 王五 |

1. PS：在Email Pass的方式中，只需要分配开发人员Review的配对关系，不需要针对某个具体的模块或者物理项目。
2. **审查阶段**：（Role-角色）负责BUG或者JIRA的开发人员，修改完代码以后，将修改的代码通过Eclipse导出为 xxx.patch文件（Eclipse中选源码树->右键->Team->Create Patch或者EOS Studio中项目->右键->小组->创建补丁），并通过如下邮件格式发送给配对Review的开发人员以及Dev Lead（例如，上表中的张三应该发送给王五）  
     
   负责Patch文件Review的开发人员，收到邮件，通过Eclipse导入Patch文件，在变更视图中进行代码的Review，如果发现问题，通过 邮件回复给提供Patch的开发人员，告知Review发现问题，否则，回复邮件告知"Review Pass"。
3. **改写阶段**：（Role-角色）负责提供Patch文件的开发人员收到Review人员回复的邮件，如果是Review Pass，则直接提交源码库；否则，根据回复邮件描述，修改代码，重新导出Patch文件，发给配对开发人员进行Review，直到Review Pass。
4. **完成阶段**：(Role)负责Review的开发人员，检查修改，并确认正确。如果不正确，就再次进入改写阶段。

**7 相关模板**

**7.1 Email Pass的邮件发送模板**

* **发件人**：修改代码的开发人员
* **收件人**：开发Leader，负责Reveview自己代码的开发人员
* **邮件标题**：[PlsRevew][BUG]XXX / [JIRA]XXX / [PROBLEM]XXX：问题简要描述
* **邮件内容**：

|  |  |
| --- | --- |
| **BUG/JIRA号** | 本次修改内容的BUG号或JIRA号 |
| **问题描述** | 要更详细描述BUG的情况或JIRA问题的需求 |
| **问题原因** | 是BUG产生的具体原因，对于复杂的bug需要说明 |
| **解决方案** | 是指开发人员解决bug和jira问题的具体措施和方案简介 |
| **提交的文件/Patch** | 指需要提交的文件或patch的说明，对于单个java的Patch可以省略不写；对于类似EOS项目中的一些文件，  *如jsp*，\_ image\_，\_ js等文件，不能使用Java Patch方式的，则需要重点说明，\_ 这些文件需要放在哪些目录下等等。\_ |
| **可能影响** | 是指bug/jira修改后，可能会对哪些功能产生影响，这些功能需要测试人员回归测试，以保证产品的稳定性 |
| **其他说明** | 其他需要说明的地方 |
| **作者** | 修改代码的开发人员 |

**7.2 Email Pass的邮件回复模板**

**发件人**：Review代码的人员  
**收件人**：修改代码的开发人员   
**邮件标题**：[Review Pass]/[Review Fail]//[PlsRevew][BUG]XXX / [JIRA]XXX / [PROBLEM]XXX：问题简要描述  
**邮件内容**：

* 若Reveiw Pass，则可不需要邮件内容；
* 若Review Fail，则说明：
  + 问题描述：
  + 建议修改：

**8 度量分析**

* CodeReview发现的问题数
* CodeReview所花费的工作量
* CodeReview的代码行数
* CodeReview的效率（缺陷数/人时)
* CodeReview发现的缺陷率（缺陷数/KLOC）