硅谷人如何做 Code Review

2018-01-24 朱赟

硅谷人如何做 Code Review

朱赟

- 00:04 / 13:20

今天技术管理课的主题是： Code Review，也就是我们常说的代码评审。Code Review 主要是在软件开发的过程中，对源代码进行同级评审，其目的是找出并修正软件开发过程中出现的错误，保证软件质量，提高开发者自身水平。

和国内的工程师聊天，发现国内公司做代码评审的比例并不算高，这可能和各公司的工程师文化有关系。不过硅谷稍具规模的公司，代码评审的流程都是比较规范的，模式也差不多。

首先是两个概念

在进入正题之前，先介绍两个概念，一个是 Commit，一个 PR。硅谷大部分公司都使用 GitHub 企业版管理自己的代码仓库，GitHub 里的 Commit 和 PR 的概念在代码审核中非常重要。

1 Commit。就是 GitHub 上的一次“ Commit ”行为，这是可以单独保存的源代码的最小改动单位。

2 PR。也就是 Pull Request，是一次代码提交请求。一个 PR 可以包含一次 Commit，也可以是多个。提交请求后 GitHub 会在相关页面上显示这次提交请求的代码和原代码的所有不同之处，这就是本次 PR 的所有改动。

请求提交后，其他工程师可以在 PR 的页面上提出意见和建议，也可以针对某一些代码的改动进行讨论，也可以给整体评价。代码的作者也可以回复这些意见和建议，或者按照建议进行改动，新的改动将为本次 PR 中提交新 Commit（也可以覆盖之前的 Commit）。

关于 GitHub 和 Pull Request，池老师最近在他的公众号里写了一篇“GitHub 为编程世界带来了什么改变 ”，这篇文章中有比较详细的描述，你可以参考学习。

其次，我来谈谈代码合并规则

一般情况下，所有的 PR 都必须有至少一个人认可，才能进行合并。如果改动的内容涉及多个项目，则需要每个项目都有相关人员认可才能合并。还有一些特别关键的代码，比如支付相关的，通常也会需要支付组的人确认才行。

在代码合并之前，进行 Code Reivew 的工程师们会在 GitHub 的相关页上给出各种评论，页面是共享的，这些信息大家都能看到。

有些评论是询问，代码的作者直接回复或解释就行，有些是指出代码的问题，代码作者可能会据此改动，也可能不会同意，那就需要回复评论，阐述观点，你来我往。有时候一个实现细节，讨论的主题可以多达十几条或几十条，最终需要达成一致才能进行合并。

再次，帮助别人成长，而不是帮他写代码

以前有朋友会说：“看到代码写得太差，觉得来回评论效率太低，干脆自己冲上去搞定”。曾经我也有过类似的想法，不过我慢慢意识到这并不是一个合适的想法。

首先，从对方的角度来说，代码写不好，可能是对业务不熟悉，对编程语言不熟悉，也可能是对公司代码的整体架构不熟悉。如果你帮他“写”，而不是耐心指出哪里有问题，那么下一次他可能还是不知道怎么做。这样不仅无益于别人成长，有的时候甚至会让别人有挫败感。

并且，帮别人写代码的方式可扩展性很差。即使 Code Review 会花掉十倍于你自己写的时间和精力，但它会让人明白代码应该怎么写，从长远来看，这其实是在一定程度上“复制”你的生产力。

你不能什么都自己写，尤其是开始带项目、带新人以后。每天 Review 五个人的代码和写五个人的代码，长期而言哪个更合算呢？答案显然是前者。

写代码是一个学习过程，怎么做一个好的代码审核人更是一个学习和成长的过程。自己绕过一个坑不难，难的是看到别人那么走，远远地你就能告诉他那里有个坑，而他在你的指导下，以后也会帮忙指出别人的类似问题。

我在这一点上最近感触尤为深刻。随着团队越来越大，新人也越来越多，有一段时间我每天工作的一半时间都在 Review 别人的代码，写评论。

这样做的初期确实比较辛苦，不过效果也随之慢慢显现，大部分时间工程师们已经可以进行相互 Reivew 代码了，于是我可以腾出很多时间来做别的工作，代码质量也得到了保障。

提交代码的类型

在进行 Code Review 之前，要搞清楚提交的代码到底是干嘛的，然后进行针对性的审核。我们一般把提交的代码分成四类。

1. Bug 修复。一般公司都有独立的 Bug 追踪和管理系统，每个 Bug 都有一个票据。代码提交的 PR，一般和票据是关联的。
2. 代码优化。比如文件的移动和拆分，部分函数的重构等。
3. 系统迁移。包括代码库拆分，用另一种语言重写等。
4. 新系统和新功能。新功能在实现之前都需要进行设计审核，最终版本的设计文档会包括数据库的 Schema、API 的签名（ Signature ）、代码的流程和模块等内容；相关代码的提交，也就是 PR，一般是和设计文档挂钩的。

了解了提交代码的作用，审核就会更有针对性和效率，也更容易从作者的角度阅读代码。

最后说一下 Code Review 的注意事项

从代码提交者的角度，在代码审核中需要注意哪些问题呢？

第一，为什么要进行 PR ？原因一定要在提交的时候写得非常清楚，才能帮助审核者理解这个改动是不是合理。上面说的四种提交代码的类型，具体是哪一种，应该写到 PR 的小结中，写得越详细越好。

这在以后需要进行回溯或追踪系统变化时，也是很有益的。如果改的是前端代码，最好贴一个改动前和改动后的截屏，让改动效果一目了然。

第二，除非是极其明显的单词拼写问题，尽量不要引入不是这个 PR 目的的改动。PR 要尽可能保持目标的单一性。每次遇到有人把一些代码结构的优化合并到功能相关的改动时，我都有一种肝火上升的感觉。

这种行为不仅会增加审核者的困难，降低效率，还会掩盖一些简单的错误。并且，如果因为功能的修改导致线上出了问题，一般需要退回到之前的版本，也就是反转PR ，这时候，针对优化相关的改动也就必须被反转。总之是弊远远大于利。

第三，找谁审核？除了本组的人外，有时候代码还会和其他项目组的代码相关，需要找该组的成员审核，这时具体找谁呢？

一般有两个机制来解决这个问题。一是在 GitHub 里 @ 一个组，比如 Payment 组，Risk 组等等，这些组会通知组里所有的人，相关的人看到了就回去审核；二是有一些组的代码，不希望其他组的人在自己不知道的情况下进行改动，就会设置规则，如果有人动了这些代码，也会通知到整个组。

最后，也是最重要的，一定确保所有的改动都是测试过的，无一例外。

从代码审核者的角度，又需要注意哪些问题呢？

审核的粒度要多细？是不是每次审核都要花很多时间？当然，如果时间足够，自然是看得越细越好。如果特别忙的时候，可以做一些筛选。

比如，你可以看一下算法或者编程思路，然后加一个评论 “算法部分看来没有问题”；也可以只看你关心的部分，然后加评论 “支付部分没问题”，或者 “API 部分没问题”。还可以再 @ 一些你觉得可以对其他部分加评论的人。

另外，如果是新人的代码，尽可能地在代码风格、性能等各方面都加以审查。如果是一个老员工，这些方面可以给予更多信任。

具体哪些地方需要审核呢？总结一下。

1 代码格式方面。很多公司都有编程语言风格指南（ Coding Style Guideline ），这是大家的约定俗成，避免公司的代码风格不一致，也避免了一些 “要不要把闭括号另起一行” 的无谓争论。老员工除非不小心，通常大家都不会弄错；新员工在这方面不太熟悉，就有可能出问题。这一类问题是比较容易指出的。

2 代码可读性方面。比如函数不要太长，太长就进行拆分。所有的变量名要能说明它的用意和类型（比如 hosting_address_hash，一看就知道是房东地址，而且是个哈希类型）。

不要有嵌套太多层的条件语句或者循环语句。不要有一个太长的布尔类型（ Boolean ）判断语句。如果一个函数别人需要看你的长篇注释才能明白，那这个函数就一定有重构的空间。另外，如果不可避免有一些注释，则一定要保证注释准确且与代码完全一致。

3 业务边界和逻辑死角问题。你可以帮助代码作者想想，他有没有漏掉任何业务边界和逻辑死角问题。很多时候这是业务逻辑相关的，尤其需要资深一点的工程师帮助指出需要处理的所有情况。

4 错误处理（Error Handling）。这是最常见的问题，也是代码审核最容易帮别人指出来的问题。

我在文稿中举出了一个例子，一段简单到不能再简单的代码就至少有三个潜在的问题。这些都是需要审核者注意的。

```
def update_user_name(params)
    user = User.find(params[:user_id])
    user.name = params[:new_name]
    user.save!
end
```

> 1. params 里面需要 validate 是不是有 user_id 和 new_name 这两个 key
> 2. 能不能找到这个 user_id 对应的 user
> 3. save 的时候会不会有 DB level 的 exception，应该怎么处理

5 确保测试用例覆盖到了所有的功能路径。严格来说，每段代码都应该有测试用例。如果开发者能够预见到其他人的代码改动会引发自己的代码问题，一定要增加额外的测试用例防止这种情况的发生。

6 代码质量和规范。遵循公司制定的编程规范，比如，如果有重复的代码段，就应该提取出来公用，不要在代码里随意设常数，所有的常数都应该文件顶部统一定义，哪些变量应该是私有的，哪些应该是公有的，等等。

7 代码架构。包括代码文件的组织方式，函数是不是抽象到 lib 或者 helper 文件里；是不是应该使用继承类；是不是和整个代码库的风格一致；API 的定义是不是 RESTful 的等等。

公司层面的支持

从公司层面应该有哪些措施帮助员工有效地进行代码审核呢？

1. 统一的代码提交和审核流程与工具，并确保大家使用同样的工具，遵循相同的流程。
2. 鼓励员工帮助别人审核代码，甚至可以做到效绩评估中。
3. 制定统一的编程规范和代码风格，尤其是在有争议的地方，这样可以解决一些因为程序员偏好带来的矛盾。

代码审核和编程一样，都是日常工作，不要情绪化。我曾经做过一件事，一个外组同事的代码，别人已经认可合并了，可是我觉得有问题，于是反转了流程，在评论里写了原因和建议的改法；当时心里还觉得会不会得罪人。可是后来发现同事反而很客气地接受并道谢了。

另外，评论里经常会出现不同意见，其实是两个人在编程习惯和约定俗成上没有达成共识。如果在不违背公司风格指南的情况下，没必要一定让对方和你有一样的习惯。

如果你真的觉得这样做更好，可以委婉地提出来，比如“我个人是更偏向于A类型的编程风格，不过这不是一个硬性规定。”或者“嗯，改成A类型的编程风格如何，不过这不是强制的。”

今天我和你讨论了 Code Review，主要内容包括 Code Review 中的重要概念，代码合并的规则，帮助别人评审代码而不是写代码，提交代码的类型和做 Code Review 时的注意事项。

事实上工程师们在编程的时候很难保证万无一失，多几双眼睛一起帮你看一遍，就可以很大程度地减少代码里的错误。另外，相互审核的过程中还能从彼此那里学到很多编程的小技巧和好习惯。细想来，很多 Java 和 Ruby 的特别好用的技巧，我都是在做代码审查的过程中学到的。

你的团队有没有做 Code Review 呢？有好的经验可以在留言里告诉我。感谢你的收听，我们下期再见。

songyy

2018-01-25

写蛮棒的，我比较喜欢文中的观点：1) 代码审查可以让工作可扩展；2) 审查边界情况；3) 代码回滚的例子；4) 那段Ruby代码的考虑点问题；5)有ui改动相关的，要给改动前和改动后的对比图(这点我一直在做，也在要求别人做 ��)

此外，从我经验上，还有这么几点：
1) 持续集成环境可以跑测试，测试结果可以自动加到PR上面(我觉得你们肯定是有在这么做的 ��)。这样子让review的人有个更好的信心
2) 代码在build的过程中，可以加上linter，进行code style的检查(我觉得这个你们肯定也在做 ��)。持续集成环境可以把检查结果加在PR上面。我们公司在用code climate的Chrome plugin，这样子做code review的人就不需要提style相关的改动了：能让机器做的事，就没必要让人来做
3) 如果希望强制互相的代码审查，可以在github里面设置，比如master branch，是protected branch，只有在ci通过，有人approve了这个pull request之后才允许merge
4) 在pull request本身要给足context信息，从而让进行review的人很容易搞清楚这个pr在做什么；在之后希望溯源什么改动的时候，在定位到这个pr时，也可以更好地了解当时做这个改动的原因。

作者回复

2018-01-31

这回复写的真是专业，赞

aaaa

2018-01-25

我们也有代码审核，但是我认为很失败，主要表现在形同虚设。
原因如下：
1.领导分配任务时，没有给评审分配时间。在相同的时间内，在原有工作量之上在加入代码评审，评审就是负担，应付了事；再说代码评审有没有奖惩。不看代码，接先approval然后merge就成了最常见的做法。
2.pr看不懂
a.pr说明描述不清或者没有描述。
b.pr里面包含很多东西。
3.pr不给留时间，或者留时间了也没有人看。

杨少侠

2018-01-24

工作时间没时间做review

木头疙瘩

2018-04-24

真的挺希望能加入一家有这样环境的公司，然而我呆过的大小厂都没有这个

二杆子

2018-02-23

很庆幸之前的团队就是这么做的，开始觉得太苛刻，后面慢慢觉得这样对大家(特别是对自己)的成长挺大的。后面我在review别人的代码的时候也“苛刻”了些��

bluze

2018-01-24

好吧 你们的代码提交真复杂

self-discipline

2018-07-30

首先是招聘到合适的人；其次如果人不行，怎么推进点东西都费劲的；我们这我推进的代码审核，好似把他们打劫了一番，好比老牛不喝水推上刑场一般

anchor

2018-06-29

有什么好的review工具推荐吗？之前用fisheye git有好的实践方式吗

anchor

2018-06-29

大家有什么好的review工具，之前都用fisheye

---

anchor

2018-06-29

大家有什么好的review工具，之前都用fisheye

---

gevin

2018-06-19

GitHub 推荐开发时才有GitHub Flow进行协作，原来不是很理解，仅以为PR主要用于向他人的开源项目做贡献。看了安姐这篇文章，才意识到，PR和GitHub Flow，能很好的落实code review，这应该是推荐使用GitHub Flow的一个更重要原因

---

xiao

2018-05-21

请问在各自具体团队上是有专门的code review人员？如果是团队里不是专门评审，那工作量如何平衡

Lament

2018-02-04

高校里面，菜鸡互啄其实问题不大，只要言之有物，至少能带来观点的碰撞，不要演变成相互人身攻击或者某些无意义的争论就好，比如Php是最好的语言……很烦。

---

ajodfaj

2018-01-25

在高校，都是菜鸡互啄，就不review了吧

nigel

2018-01-24

我想有人来review我的代码，但大哥们都忙。