45 | 代码评审：寄望与哀伤

2018-11-14 胡峰

我们都知道代码评审（Code Review）很有用、很重要，但现实中我所经历的和看到的团队，很少有能把代码评审落地得很好，并发挥出其应有作用的。这个问题困扰我已久。

感性认识

代码评审的作用，有一定经验的程序员们想必都会有感性认识。

它是很多软件工程理论和方法学中的重要一环，对于提升代码质量和找出一些潜在隐患很有帮助，如果你有一些正式的代码评审经历过程，想必也能感性认知到其正面作用。但在我过去工作的这些年里，经历了几家公司，数个不同的团队，却几乎没有哪一个会把代码评审作为必要的一环去执行的。

过去，我们总是在线上出现一些奇怪的疑难问题后，一群相关程序员才围坐在一起，打开相关代码来逐行分析，根据线上现场的“尸检”来做事后分析和推导。这样的事后代码分析实际上根本不是代码评审，也完全违背了代码评审的初衷。

代码评审的初衷是提高代码质量，在代码进入生产环境前经过同行评审来发现缺陷，降低损失概率。这一点程序员都好理解，提前的代码评审就像雷达扫描我们重点关注的代码领地，以期发现或明显或隐藏的危险因素。

漫画《火影忍者》里有一种忍术技能：白眼，这种技能有近 360° 的观察范围。程序员在写程序时力求思考全面，不留死角或盲点，但实际死角或盲点总是存在的。随着我们经历和经验的成

长，思考和认识得越发全面（越发接近 360°），拥有了近乎“白眼”的能力，但即使是像“白眼”一样，也依然会存在盲点。

正如世界上没有两片完全一样的树叶，也许也不会有两个认知视角完全重叠的人，这样相互进行代码评审也就弥补了个人单一视角和认知思考的盲点问题。除此之外，代码评审还有一个社会性功用，如果你在编程，而且知道一定会有同事将检查你的代码，那么你编程的姿势和心态就会完全不同。这之间的微妙差异正是在于会不会有人将对你的代码做出反馈与评价。

代码评审的编程实践正是基于这样的感性认知，影响你的编码心理，并试图通过群体视角来找出个体认知盲点区域的隐患或 Bug，但到底这样的做法能降低多少出现 Bug 的概率呢？

理性分析

有人对代码评审的作用进行了更理性且量化的分析，结论如下（来自维基百科）：

> 卡珀斯·琼斯（Capers Jones）分析了超过 12,000 个软件开发项目，其中使用正式代码审查的项目，发现潜在缺陷率约在 60%～65% 之间；若是非正式的代码审查，发现潜在缺陷率不到 50%；而大部分的测试，发现的潜在缺陷率会在 30% 左右。

> 一般的代码审查速度约是一小时 150 行，对于一些关键的软件，一小时数百行代码的审查速度太快，可能无法找到程序中的问题。对于产品生命周期很长的软件公司而言，代码审查是很有效的工具。

从如上的实验分析结论来看，代码评审对于发现潜在缺陷很有用，相比测试能发现的缺陷率高一倍，但也需要投入巨大的时间成本 —— 一小时审查 150 行代码，再快就不利于发现潜在缺陷了，而且更适用于长生命周期的产品。

所以，下面这个现象就容易理解了。我发现在同一家公司做代码评审较多的都是研发通用底层技术产品或中间件的团队，而做业务开发的团队则较少做代码评审。两者对比，底层技术产品或中间件的需求较稳定，且生命周期长；而业务项目，特别是尝试性的创新业务，需求不稳定，时间要求紧迫，并且其生命周期还可能是昙花一现。

多种困境

从感性和理性两个角度认知和分析了代码评审的好处，但其适用的场景和花费的成本代价也需要去平衡。除了这点，如果把代码评审作为一个必要环节引入到研发流程中，也许还有一些关于如何实施代码评审的困境。

困境一，项目期限 Deadline 已定，时间紧迫，天天加班忙成狗了，谁还愿意搞代码评审？这是一个最常见的客观阻碍因素，因为 Deadline 很多时候都不是我们自己能确定和改变的。

困境二，即使强制推行下去，如何保障其效果？团队出于应付，每次走个过场，那么也就失去了评审的初衷和意义。

困境三，团队人员结构搭配不合理，新人没经验的多，有经验的少。新人交叉评审可能效果不好，而老是安排经验多的少数人帮助 Review 多数新人的代码，新人或有收获，但对高级或资深程序员又有多大裨益？一个好的规则或制度，总是需要既符合多方参与者的个体利益又能满足组织或团队的共同利益，这样的规则或制度才能更顺畅、有效地实施和运转。

困境四，有人就是不喜欢别人 Review 他的代码，他会感觉是在找茬。比如，团队中存在一些自信超强大的程序员，觉得自己写的代码绝对没问题，不需要别人来给他Review。

以上种种，仅仅是我过去经历的一些执行代码评审时面临的困境与障碍，我们需要找到一条路径来绕过或破除这样的障碍与困境。

参考路径

在国内，我并没有看到或听闻哪家把代码评审作为一项研发制度或规则强制要求，并落地得很好的公司。

而对于一些硅谷的互联网公司，倒是听闻过一些关于代码评审的优秀实践。比如，在一篇介绍 Google 代码评审实践的文章中说道：在 Google，任何产品，任何工程的代码，在被进行严格或者明确的代码评审之前，是不允许提交的。这一点，Google 是通过工具自动就控制了未经评审的代码就没机会进入代码库。

Google 以一种强硬的姿态来制定了关于代码评审的规则，规则适用范围是全公司，对任何人都不例外。即使面对团队中超自信且强大的程序员也无例外，要么遵守规则，要么离开组织。这一点从 C 语言和 Unix 的发明者、图灵奖得主——肯·汤普森（Ken Thompson）在 Google 的趣事中可以一窥其规则的强硬性，作为 C 语言发明者之一的他因为没有参加 Google 的编程语言能力测试所以无法在 Google 提交 C 代码。

所以，像 Google 这样的公司对于代码评审属于高度认可且有公司制度和规则的强硬支持，再辅助自动检测和控制工具的严格执行，一举就破解了以上四类困境。但要实践类似 Google 这样严格的代码评审制度和规则，似乎对于大部分公司而言都有不小的挑战，这需要公司制度、团队文化和技术工具三方面都能支持到位，而且还要让各方对实施此项制度的收益和代价取得一致认可，岂是易事？

所以，现实的情况是，大部分公司都是在各自的小团队中进行着各种各样不同形式的代码评审，或者完全没有代码评审。

现实选择

以前尝试过要在团队内部做代码评审，听说兄弟团队搞得不错，然后就一起交流经验。交流开始不久就跑偏了，重心就落在了应该选个什么好用的代码评审工具来做，如今想来这完全是舍本逐末了。

这就像以为有了好的编辑器（或 IDE）就能写出好的代码一样，而事实就是有很多好用的代码评审工具我们依然做不好代码评审。这让我联想起了古龙小说《陆小凤传奇》中的一段描述，记忆尤深：

> 西门吹雪：此剑乃天下利器，剑锋三尺七寸，净重七斤十三两。
>
> 叶孤城：好剑。
>
> 西门吹雪：的确是好剑。
>
> 叶孤城：此剑乃海外寒剑精英，吹毛断发，剑锋三尺三，净重六斤四两。
>
> 西门吹雪：好剑。
>
> 叶孤城：本就是好剑。

剑是好剑，但还需要配合好剑客与好剑法。

即使在最差的环境下，完全没有人关心代码评审这件事，一个有追求的程序员依然可以做到一件事，自己给自己 Review。就像写文章，我写完一篇文章不会立刻发布，而是从头脑中放下（Unload），过上一段时间，也许是几天后，再自己重新细读一遍，改掉其中必然会出现的错别字或文句不通畅之处，甚或论据不充分或逻辑不准确的地方，因为我知道不管我写了多少文字，总还会有这些 Bug，这就是给自己的 Review。

给自己 Review 是一种自省，自我的成长总是从自省开始的。

代码评审，能提升质量，降低缺陷；代码评审，也能传播知识，促进沟通；代码评审，甚至还能影响心理，端正姿势。代码评审，好处多多，让人寄予希望，执行起来却又不免哀伤，也许正是因为每一次评审的收益是不确定的、模糊的，但付出的代价却是固定的，包括固定的评审时间、可能延期的发布等。

哀伤过后，我们提交了一段代码，也许没人给我们 Review，稍后我们自己给自己 Review 了，也可以得到了一段更好的代码和一个更好的自己。

最后，我曾在前文《思维：科学与系统》中就用代码评审作为例子说明了这是一个系统问题，每个团队面临类似的系统问题都会有具体的情况。关于代码评审，不妨谈谈你所在环境所面临的情况和你的理解？

极客时间

程序员进阶攻略

每个程序员都应该知道的成长法则

胡峰 京东成都研究院 技术专家