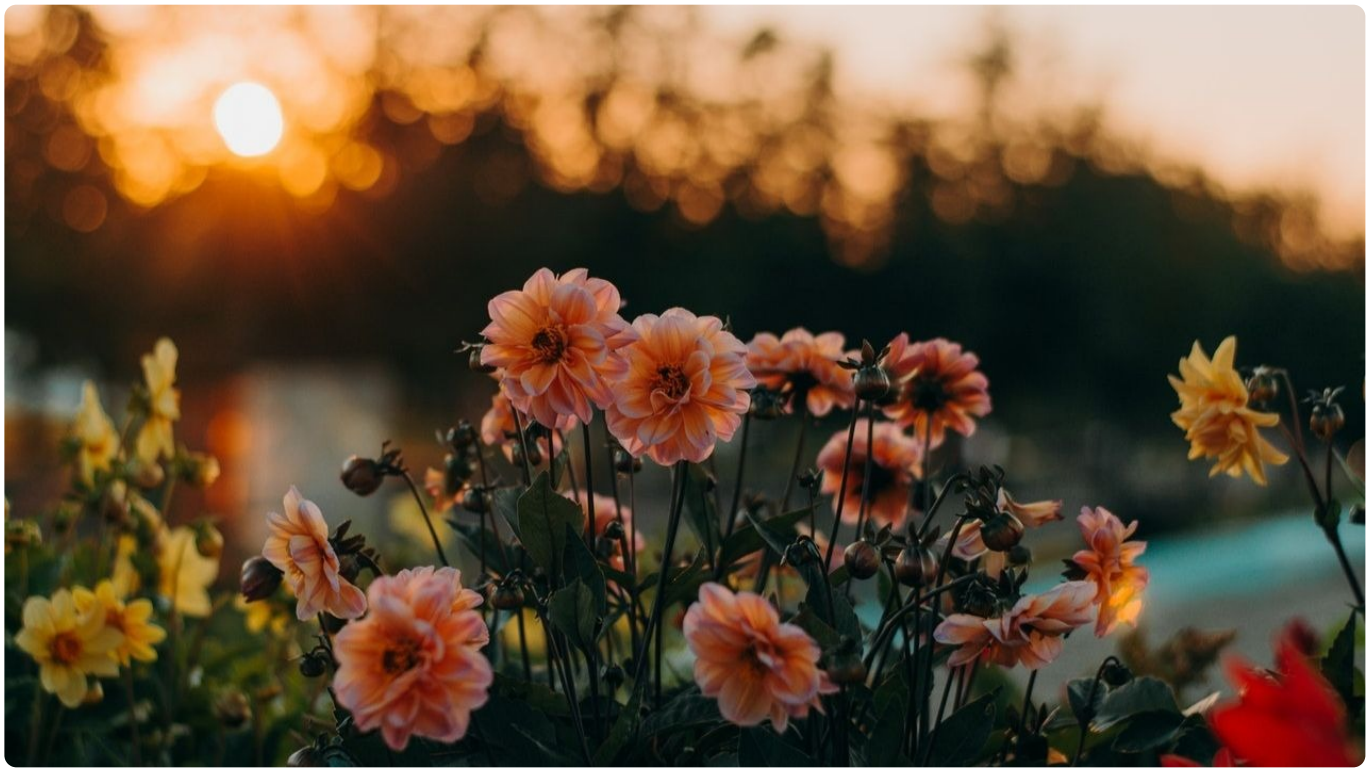
加餐三 | 聊一聊Google是如何做Code Review的

2020-06-24 王争

设计模式之美

进入课程 >



讲述：冯永吉

时长 09:35　大小 8.79M

100 篇的正文已经全部结束了，估计你学得也有点累了吧？时隔这么久，正文终于结束了，从今天起，我们继续加餐内容。

跟正文内容相比，加餐内容我希望尽量轻松有趣，帮你拓展知识面，主要是课后的一些小分享，有的会以讲故事为主，但我也希望它能给你带来收获。如果能够引发你的思考和共鸣那就更好了。所以，我也希望你在留言区，多说说自己的感受和看法，多多与我互动。

话不多说，让我们正式开始今天加餐的内容吧！

为什么国内企业不重视 Code Review?

在专栏第 80 讲中，我列举了 Code Review 的重要性，在项目中执行 Code Review 会带来哪些好处，以及如何克服一些常见的难题，在项目中启动 Code Review 等等。今天，我们想再继续这个话题，和你聊一下 Code Review。不过，我刚才也说了，今天的内容会相对轻松一些，我会主要给你讲讲我在 Google 做 Code Review 的一些经验和心得。

我们都知道，Google 在 Code Review 方面做得非常好，可以说是很多公司学习的榜样。从我个人的经历来说，我的技术成长相当大的一部分得益于当年在 Google 的 Code Review。所以，我也希望更多的同行能意识到 Code Review 的重要性，能够在项目中推行 Code Review，受益于 Code Review。

但据我了解，国内的大部分公司都不怎么接受 Code Review，在开发中，根本没有 Code Review 的流程。所以，我一直思考，到底是什么原因，导致这么优秀的一种开发模式，在国内的技术圈内没有被发扬光大。很多人会认为，主要原因是，项目工期紧，没时间做 Code Review。我觉得这只是表面的原因，最根本的原因还是缺少技术文化传承。

我们知道，普遍而言，越是大公司里的工程师，技术能力会越强，技术影响力会越大。这些公司的工程师，即便跳槽去其他公司，一般都会担任核心成员或者 Leader 的角色。但是，在国内，即便像 BAT 这些输出有影响力工程师最多的一线公司，也没有很好地实践 Code Review，相对应的，这些公司的工程师也就没有一手的 Code Review 的经验和感受，更无法了解到 Code Review 的好处，也更不会在团队、公司，甚至技术圈中去推行 Code Review 了。

打个不恰当的比方，这些一线互联网公司的工程师一直接受着“996”狼性文化价值观的熏陶，即便跳槽去其他公司，作为资深员工或者技术 Leader，他们也会带领新的团队开始 996，最终导致整个 IT 行业的加班氛围都很浓，不加班反倒会显得不正常。

用 996 作类比，如果 BAT 这些比较有技术影响力的公司，内部对 Code Review 很认可，执行得非常好，从这些公司往外输出的工程师，就会像我一样，大力传播 Code Review。星星之火可以燎原，慢慢地，整个技术圈就会接受并且推行 Code Review 了。

实际上，据我所知，不只是我，只要是从 Google 跳槽出来的工程师，到了其他公司之后，都特别热衷于传播 Code Review。而且，只要是被 Google 工程师带领过的团队，在开发流程中严格执行过 Code Review 的团队，对 Code Review 都无比认可。所以，我个

人觉得，很多人不认可、不推行 Code Review，最直接的原因还是没有经历过 Code Review，没有有经验的人来带。

实际上，才开始接触 Code Review 的时候，我也比较反感。我刚毕业就进入了 Google，在此之前，上学的时候，尽管也写了很多代码，也参与过一些垂直课题的研发，但是，那时候的开发只是为了完成功能，从来没有考虑过代码质量问题、代码设计问题，更别提 Code Review 了。现在想想，自己当时对 Code Review 的认知水平，跟现在很多国内工程师的认知其实是差不多的。

所以，在一开始进入 Google 的时候，对于 Code Review 我也是不怎么接受的。我第一次提交的代码不足百行，就被 Leader Review 出了 n 多问题，而且大部分问题都非常细节，比如变量的命名不够达意、注释不够规范、多了一个空行、少了一个空格之类的。对于这些琐碎的细节，我当时心里挺排斥的，心想：我是来“造火箭”的，为什么成天纠结于这些“拧螺丝”的事情呢？

现在回去想想，当时的想法真的挺幼稚的。

如果站在团队协作的角度来看，对于一个长期维护、多人参与、代码比较多的项目来说，代码的可读性、可维护性等与质量相关的问题，是非常重要的。所以，Code Review 作为保证代码质量的最有效手段之一，也就非常有必要了。如此吹毛求疵地执行 Code Review，看似非常极端，但也表明了公司强硬的态度、坚定的立场，就是要把 Code Review 执行彻底。这也是 Code Review 没有在 Google 流于形式的一个很大的原因。

在入职一段时间后，来来回回经过多次 Code Review 之后，我的代码质量整体提高了很多，被 Review 出的问题也越来越少了，我也切身地体会到 Code Review 的好处。因此，慢慢地，对这件事，我从排斥变得认可。与此同时，我也慢慢地开始 Review 别人的代码了。

Google 是如何进行 Code Review 的?

在 Google，我们把每次提交的代码片段叫做一个 CL，全称是 Change List。它就相当于 GitHub 中的 PR（Pull Request）。每个 CL 都要至少一个 Owner 和一个具有 Readability 的同事 Approve，才能提交到代码仓库中。其中，Owner 一般都是技术 Leader 或者项目负责人，而 Readability 是一个证书，表示你具有了写出可读代码、符合

编码规范代码的能力。Readability 会细化到每种编程语言，比如 Java Readability、C++ Readability 等。

如果你想申请某种语言的 Readability，你就要提交一段至少包含 100 行代码、并且稍微有点复杂的 CL 给 Readablity 评审委员会。评审委员会会指派一个资深工程师 Review 你的代码，给你一些修改建议，然后，你需要根据修改建议对代码进行修改，再提交 Review。这样来来回回几次之后，他觉得没问题了，就会给你颁发 Readability。有了 Readability 之后，你的 Review 才真的能起到 Approve 的作用。当然，即便没有 Readability，你对同事代码的 Review 本身也是有价值的。所以，并非只有 Readability 的人才能 Review 别人的代码。

在 Google，每种编程语言都有对应的编码规范。但是，Code Review 本身并没有统一的 Check list。在 Code Review 的时候，除了编码规范可以参考之外，大部分都是靠工程师自身的经验来 Review。不过，Review 考虑的也无外乎这样几个常见的方面：代码结构是否合理、代码是否容易理解、业务是否正确、异常考虑是否全面、是否有隐藏的 bug、线程是否安全、性能是否满足业务需求、是否符合编码规范等等。

Code Review 听起来很复杂，要考虑的点很多，但实际上，等到你做熟练了之后，并不会花费太长的时间。一个 CL 从提交 Review 到最终合并到代码仓库，一般也就需要一天的时间。当然，对于一些比较大的 CL、比较复杂的 CL、有比较多争议的 CL，以及一些新手的 CL，可能会花费比较多的时间。

但是，大部分情况下，我们都不提倡太大的 CL。太大的 CL 对代码审查者来说是很大的负担，Review 过程会很慢，会导致代码迟迟提交不上去。

对于比较复杂的 CL，我们一般建议要写好文档，或者通过类似 Jira 这样的项目工具，详细描述 CL 的前因后果、上下文背景。这样，代码审查者就能一眼看懂代码包含的设计意图。对于争议比较多的 CL，我们建议直接当面沟通，这样也更加有效率。对于一些新手的 CL，因为他们对编码规范等不熟练，可能来来回回要改好几次，才能满足要求，但这个过程是每个新人都要经历的，多改几次就好了。

实际上，Code Review 并不神秘，如果你想了解更多关于 Code Review 的事情，可以去读一读 Google 官方公布的 Code Review 最佳实践。当然，如果有什么疑问，你也可以在留言区问我。

让国内大部分 IT 从业人士认识到 Code Review 的重要性，形成 Code Review 的技术文化，可能还需要一个漫长的时间。不过，我特别希望，你在学完专栏之后，能够意识到 Code Review 的重要性。有朝一日，当你成了领导，有了话语权、影响力之后，能够推动在团队、公司内进行 Code Review，甚至为 Code Review 在整个国内技术圈中发扬光大贡献一份力量。

课堂讨论

你觉得为什么国内的大部分公司都不重视 Code Review，在开发中都没有 Code Review 流程呢？你觉得如何把 Code Review 在国内技术圈中发扬光大呢？有什么好的建议吗？

欢迎留言和我分享你的想法，如果有收获，也欢迎你把这篇文章分享给你的朋友。

上一篇　春节特别加餐 | 王争：如何学习《设计模式之美》专栏?

精选留言 (1)

wkq2786130

2020-06-24

我刚开始也不理解code review，直到有一天发现自己写的代码自己读不懂，然后开始优化，开始写注释，理清主要逻辑，开始分层，开始使用通俗易懂的命名，后来逐渐意识到code review的好处