27 | 特别放送：聊一聊代码审查

2019-11-11 四火

全栈工程师修炼指南 进入课程 >

讲述：四火
时长 19:41 大小 13.53M

你好，我是四火。

又到了特别放送时间，今天我想聊一聊代码审查（Code Review）。在软件工程师日常的开发工作中，**如果要挑出一项极其重要，却又很容易被忽视的工作，在我看来代码审查几乎是无可争议的第一。**

代码审查是一个低投入、高产出的开发活动，就我个人而言，我从其中学到的习惯、方法和知识，让我获益匪浅。但是，我也在微博、微信上看到程序员朋友们争论代码审查的必要性，甚至包括很多大厂的程序员，还有一些有着许多年经验的程序员。

一开始我觉得有些不可思议，和代价相比，代码审查的好处实在太多了，这有必要费那么大心思去讨论这个必要性吗？后来我意识到，**造成这个争论的原因，既包括缺乏对于代码审查**

好处的认识，也包括一些因果逻辑上的混淆。我想，今天的特别放送，我就来把代码审查这个事儿聊清楚，希望它能成为你日常开发工作当中认真对待的必选项。

那值得一提的是，对于全栈工程师而言，代码审查又有一点特殊性。因为我们经常要写多个层面的代码，包括前端代码 HTML、CSS、JavaScript，后端逻辑，比如 Java 或者 Python，还很可能写很多的脚本代码，比如 Shell，做各种各样的配置，像是和基于 XML、JSON 的配置文件打交道，还很可能使用 SQL 写持久层的逻辑。这些代码中，既包括命令式的代码，也包括声明式的代码。由于涉及到的代码类型比较广泛，代码审查者就自然会和不同的团队或项目中的角色打交道，也需要在不同的思维模式之间来回切换。

代码审查的流程

先来简单介绍一下常见的代码审查的流程。为了开发某个新特性，或者修复某个特定问题，负责的程序员会从代码库的主分支（master branch）上面建立并 check out 一个新的分支，将工作分为一次到若干次的“代码变更”来提交。这每一次的代码变更，都可以组成一次代码审查的单元，有的公司把它叫做 CR（Code Change），有的叫做 PR（Pull Request），还有的叫做 CL（Change List），但无论叫做什么，它一般至少包含这么几项内容：

- **帮助理解代码的描述**，如果有问题单（任务）来跟踪，需要包括相关的问题单号或者问题单链接；
- **实际的代码变更主体**，包括实际的代码和配置；
- **测试和结果**，根据项目的情况，它可以具备不同形式，比如单元测试代码，以及手工测试等其它测试执行的结果说明。

多数情况下，以上这三项都不可或缺，缺少任何一项都会让代码变更失去一定可审查的价值。

进行审查的，一般是一起工作的，对代码涉及变更熟悉的其他程序员。这个“熟悉”，既包括业务，也包括技术，二者当中，有一项不具备，就很难做好审查工作，给出有建设性的审查意见。

接下去的交互就在这个代码变更上面了，审查者会提出其问题和建议，变更的作者会选择性采纳并改进变更的描述、代码主体以及测试。**双方思考、争辩，以及妥协，目的都是寻求一**

个切合实际且可以改进代码质量的平衡。

如果审查的程序员觉得代码没有太多问题，就会盖上一个“Approved”或者“Shipped”戳，表示认可和通过。这根据项目而定，一般代码变更最少要得到 1~3 个这样的认可，才可以将代码变更合并（merge）到主分支。而主分支的代码，会随着 CI/CD 的流程进入自动化的测试程序，并部署上线（关于这部分你可以参阅 [第 30 讲]）。

常见的争议

在介绍代码审查的好处之前，我想先来谈谈争议。因为我观察到**大多数的争议，都不是在否认代码审查的好处，而是聚焦在不进行代码审查的那些“原因”**或者“借口”上，而有些讽刺的是，我认为**这里面大部分的“原因”，所代表着的因果关系并不成立**。

1. 加班要累死了，完成项目都来不及，还做什么代码审查?

类似的问题还有，“代码审查拖慢了进度”，“代码审查不利于快速上线”。这是最常见的不做代码审查，或者草草进行代码审查的理由了，但是稍稍一细想，就会发现这里的因果逻辑完全不对。

这就像以前国内大兴“敏捷”的时候，有好多程序员，甚至项目经理，觉得因为项目时间紧才要实施敏捷，因为可以少写文档，少做测试，随意变更需求，可这里的因为所以根本是牛头不对马嘴。我记得知乎上有句流行的话叫做，“先问是不是，再问为什么”，这里也可以用，因为项目压力大就让“不做代码审查”来承担后果，这实在是过于牵强了。

项目压力大，时间紧，可以草草做分析，不做设计，直接编码，不做重构、不做测试、不做审查，直接上线，快及一时，可是造成的损失，最后总是要有谁来背锅的。这个锅很可能，就是上线后无尽的问题，就是恶性循环加班加点地改问题，就是代码一个版本比一个版本烂。当这些问题都焦头烂额，就更不要说团队和程序员的成长了。

2. 代码审查太费时间，改来改去无非是一些格式、注释、命名之类不痛不痒的问题。

这也是个逻辑不通的论述，虽然这个还比前面那个稍微好一点。只能提出这些“次要问题”，很可能是代码审查的能力不够，而并非代码审查没有价值；或者是代码审查的力度不够，只能提出一些浅表的问题，这个现象其实更为普遍。

前面已经介绍过了，**一是技术，二是业务，二者缺一都无法做出比较好的审查。**在某些特殊情况下，有时候确实不具备完备的代码审查条件，我们现在来分业务、技术欠缺的情况进行讨论。

如果团队中有业务达人，但是技术能力不足。比如说，新版本使用的是 Scala 来实现的，但是团队中没有精通 Scala 的程序员，这个时候可以寻找其它团队有 Scala 经验的程序员来重点进行技术层面的代码审查，而自己团队则主要关注于业务逻辑层面。当然，既然是自己团队的代码，所用到的技术要慢慢补起来。

如果团队的成员具备技术能力，但是业务不了解。这种情况也可以进行将业务和技术分开审查这样的类似处理，但是如果业务相对复杂，可以先开一个预审查会，就着代码或者设计文档，简单地将业务逻辑介绍和讨论清楚，再进行审查。

3. 团队的习惯和流程就是不做代码审查，大家都是这么过来的。

我觉得这也不是一个论述不应该做代码审查的正当理由，类似的还有“绩效考评又不提代码审查”，以及“我上班、码代码、下班、拿钱，审查代码干什么”。大家都不做，并不代表不做就是正确的，如果你赞同代码审查的好处和必要性，那么你的思考会告诉你，应该做这件事情，大家不做并不是一个理由。

如果你发现这件事很难推动，你可以尝试去和你的项目经理聊一聊，或者结合自己的项目以及下面会讲到的代码审查的好处论一论，看看是不是能说服那些没有意识到代码审查好处的程序员和项目经理。当然，这是另外和人沟通以及表达自己观点的事情，如果大家都是朴素的干活拿钱的观点，没有对于代码质量和个人发展更高的追求，或者价值观和你相距十万八千里，改变很困难，你就应该好好思考是不是应该选择更好的团队了。

4. 代码审查不利于团建，因为经常有程序员因为观点不同在代码审查的时候吵起来。

这依然不是一个正当理由，这就好像说“因为开车容易出交通事故，所以平时不允许开车”这样荒谬的逻辑一样。

首先，如果有偏执的不愿意合作的程序员，那么不只是代码审查，任何需要沟通和协作的活动都可以把争吵的干柴点燃。对于这样的程序员的管理，或者如何和这样的程序员合作，是

另外的一个话题，但这并不能否认代码审查的必要性。当然，在下文讲到实践的部分我会介绍一些小的技巧，帮助你在代码审查中心平气和地说服对方。

其次，有控制的一定强度内的争执，未必是坏事。有句话叫做“理越辩越明”，除了能做出尽可能合理的决定以外，在争论的过程中，你还会得到分析、思考、权衡、归纳、表达，乃至心理这些综合能力的锻炼，本来它们就不是很容易得到的机会，我们为什么还要放过呢？

代码审查的好处

下面我们来谈谈代码审查的好处。你可能会想，这有什么可谈的，这好处难道不是发现软件bug，提高代码质量吗？

别急，代码审查的好处可远远不止这一个，我觉得它还至少包括下面这些好处。

1．代码审查是个人和团队提升的最佳途径之一。

这里的学习，既包括技术学习，也包括业务学习。和英语学习一样，如果只听 BBC 或者 VOA 的纯正口音，没有任何语法错误，英文反而不容易学好，学英文就要接触生活英语，各种口音，各种不合标准的习惯用法。阅读代码也一样，要学习不同的代码风格和实现。

在做代码审查的时候，如果不理解代码，是无法给出最佳审查的。因此自己会被迫去仔细阅读代码，弄懂每一行每一个变量，而不是给一个 LGTM（“Looks Good To Me”）了事。

2．代码审查是团队关系建设和扩大双方影响力的有效方式。

争论是这个过程中必不可少的一环，争论除了能加深对于问题和解决方法的理解，在不断的反驳和妥协中，也能树立影响力，建立良好的关系。另外值得一提的是，**代码审查可不是说非得给别人挑刺儿，对于做得特别漂亮的地方，要赞扬，这也是建立良好关系的一种途径。**从团队合作和交流的角度来说，程序员往往缺乏沟通，每个人不能只专注于自己的那一份代码默默耕耘，而是需要建立自己的影响力的，代码审查过程中的交互，就是一个不可多得的方式。

3．识别出设计的缺陷，找到安全、性能、依赖和兼容性等测试不易发现的问题。

代码审查在整个软件工程流程中还算早、中期，尽早发现问题就能够尽可能地减少修复问题的成本。而且，代码审查能够发现的问题，往往是其它途径不易发现的。因此，从这个角度来讲，代码审查要有方向性，比如主流程和某些重要用例，在审查的时候可以要求代码变更的程序员提供单元测试，或者是手工覆盖的测试结果，这样就可以认定这些分支覆盖到的逻辑是正确的，不需要在审查时额外关注。

4. 设立团队质量标杆的最佳实践方式。

在我经历的团队中，基本上代码审查做得好的，代码质量都高。这不见得是程序员的能力特别出色，而是通过代码审查把这个质量的 bar 顶起来了。你可以想象，一个对别人的代码颇为“挑剔”的人，他会对自己的代码截然相反地糊弄了事，睁一只眼闭一只眼吗？特别对于刚踏入职场的程序员来说，这点尤为重要，要知道一个人刚工作的两三年，对性格、习惯这些关乎职业生涯因素的影响是巨大的，一个好的标杆比任何口号都有效。

一些小技巧

最后我们来谈一些小技巧，来帮助这个代码审查的过程顺利进行。

1. 每次变更所包含的代码量一定要小。

这一点很重要，代码变更是要给人看的，因此确保变更足够小，能够让它容易理解，审查代码的人，也不会觉得疲劳和有压力。代码清楚了，审查也就可以有效地进行，也更容易得到通过和认可。如果预计代码量大怎么办？可以尝试将其分解成若干个小的变更，一个一个提交。

2. 让团队中的“牛人”在代码审查中发挥作用。

团队中的核心成员，可以相对来说少做一点实现，多在设计上做一点参与和决策，多把握代码审查这一环节。以前我在某一个团队中，总代码超过了六十万行，我们实施过这样一种管理方式，将代码划分为几个大的模块，每一模块都指定一个技术责任人，他会对该层代码全面负责，所有的代码变更都要经过他的审查。

3. 变更代码的质量要超过当前代码库的平均水准。

代码的审查意见有“建设性意见”和“次要意见”之分，那么那些“次要意见”，例如格式、注释、命名，到底做到什么层次，就会成为一个争论的话题，要求低了代码质量接受不了，而要求高了又会拖慢开发进度。我觉得，这种情况下，可以遵循这样的判断标准：看**新提交的代码会让当前的代码库代码质量更高了，还是更低了**，只有高于当前项目平均质量的代码才能合并入主分支。

4. 新员工代码，骨架代码的代码审查要更为严格。

对于新员工的代码审查可以稍微严格一些，这有助于培养良好的质量意识和习惯，前面已经提到了，这对于职业生涯都是有益的。“骨架代码”指的是那些与项目业务无关的架构代码，这部分代码从技术的层面来说更加重要，往往也很考验代码功底，代码审查可以更严格一些。

5. 及时表达肯定，委婉表达意见。

只针对代码，不针对人。这听起来很简单，都知道对事不对人的重要性，但是要非常小心不能违背。审查并不是只提反面意见的，在遇到好的实现，不错的想法的时候，可以表示肯定，当然这个数量不宜多，要不然适得其反。至于表达意见方面，我来举几个例子：

对于一些次要问题，我都会标注这个问题是一个 picky 或者 nit 的问题（“挑剔的问题”）。这样的好处在于，明确告知对方，**我虽然提出了这个问题，但是它没有什么大不了的，如果你坚持不改，我也不打算说服你。**或者说，我对这个问题持有不同的看法，但是我也并不坚信我的提议更好。

使用也许、或许、可能、似乎这样表示不确定的语气词（英文中有时可以使用虚拟语气）。这样的好处是，缓和自己表达观点的语气。比如说：“这个地方重构一下，去掉这个循环，也许会更好。”

间接地表达否定。比如说，你看到对方配置了周期为 60 秒，但是你觉得不对，但又不很确定，你可以这样说：“我有一个疑问，为什么这里要使用 60 秒而不是其他值呢？”对方可能会反应过来这个值选取得不够恰当。你看，这个方式就是使用疑问，而非直接的否定，这就委婉得多。

放上例子、讨论的链接，以及其它一些辅助材料证明自己的观点，但是不要直接表述观点，让对方来确认这个观点。比如说：“下面的讨论是关于这个逻辑的另一种实现方式，不知你觉得如何？”

先肯定，再否定。这个我想很多人一直都在用，先摆事实诚恳地说一些同意和正面的话，然后用“不过”、“但是”和“然而”之类的将话锋一转，说出相反的情况，这样也就在言论中比较了优劣，意味着这是经过权衡得出的结论。

6. 审查时，代码要过两遍，第一遍抓主要问题，第二遍看次要问题。

代码过两遍的好处在于，可以把代码中的问题有层次地提出来。第一遍的时候，搞清楚代码大致的机制、原理、结构，这样有大的建设性问题可以提出来，等待修复或达成一致。根据第一遍的情况来决定需不需要过第二遍，如果没有大的分歧，可以过第二遍。这第二遍就可以非常仔细了，包括可以提出一些细节问题，也包括格式和命名之类的次要问题。总结一下就是，这种方式的最大好处就在于可以让大的问题被单独提出来，优先解决，让问题的讨论和解决有了层次。

总结思考

今天的特别放送就到这里，在今天的内容中，我结合自己的经历，向你介绍了代码审查的方方面面，主要涉及了“为什么要做代码审查”以及“怎么样做代码审查”这两个方面。

最后留一个小问题吧，欢迎在留言区一起讨论。

你所在的技术团队代码审查是怎么做的，你有没有什么代码审查上的小技巧愿意分享一下呢？

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　26 | 设计数据持久层（下）：案例介绍

精选留言 (2)

写留言

leslie
2019-11-11

提出点个人的理解吧:记得池老师的专利第一季有篇文章<技术leader是否应该写代码>，其实目前技术经理大概有3成的时间在做Code Review。这块曾经和一些同行聊过：就像老师课中所说的，不做基本上上线可能就挂了。

目前Code Review应当有两张方式吧:

一种是强行做规则-违反规则就不让上线，提交不通过，这块可以用工具去实现；...

展开

丁丁历险记
2019-11-11

笔记

争议项

1 加班累死，代码审查脱慢进度。

2 代码审查做的事无聊。 （改改注释）

3 习惯如此。...

展开