

13 | 代码审查：学习Facebook真正发挥代码审查的提效作用

2019-09-20 葛俊

研发效率破局之道

[进入课程 >](#)



讲述：葛俊

时长 22:04 大小 20.21M



你好，我是葛俊。今天我们来聊聊代码审查的落地。

在上一篇文章中，我和你详细讨论了代码审查的作用，也给出了选择适合自己团队审查方式的建议。但是，仅仅知道要做什么还不够，更重要的是落地。我见到很多国内公司也在尝试使用代码审查，但是效果很不好，往往流于形式，最常听到的一个负面反馈就是“代码审查浪费时间”。

代码审查的成功推行的确不是一件容易的事。今天，我们就一起尝试来解决这个问题。我会从三个方面给出一些建议：

第一，在团队内引入代码审查的步骤和方法；

第二，成功推进代码审查的关键操作；

第三，持续做好代码审查的重要原则。

今天的文章较长，我们现在就进入第一个部分，

引入代码审查的步骤和方法

从我的经验来看，要成功引入代码审查，首先要在团队内达成一些重要的共识，然后选择试点团队实行，最后选择合适的工具和流程。

1. 代码审查应该计入工作量

代码审查需要时间，这听起来好像是废话，但很多团队在引入代码审查时，都没有为它预留时间。结果是大家没有时间做审查，效果自然也就不好。而效果不好又导致代码审查得不到管理者重视，开发人员更不可能将代码审查放到自己的工作计划中。于是，形成恶性循环，代码审查要么被逐渐废弃，要么流于形式。

之前在 Facebook 的时候，我们预估工作量的时候就会考虑代码审查的时间。比如，我平均每天会预留 1~2 个小时用于代码审查，大概占写代码总时间的 1/5。同时，代码审查的情况会作为绩效考评的一个重要指标。

另外，平时我们也会给审查者关于审查质量的实时反馈。比如，我刚加入 Facebook 的时候，对代码审查不够重视，做得不够好。我的主管就两次给我反馈，让我提高审查质量。其中一次是让我多给同事做审查，另外一次是让我多给一些结构上的建议，不用太重视语法细节。

经过这两次反馈，我意识到了代码审查是我工作中实实在在的一部分，需要得到足够的重视。也正因为此，我才真切地感受到了代码审查的价值。

总之，管理者要明确代码审查是开发工作的重要组成部分，并记入工作量和绩效考评。这，是成功引入代码审查的前提，再怎么强调都不为过。

形成共识以后，下一步就是选择试点团队推行代码审查。

2. 选择试点团队

引入代码审查的一种方法是，在整个团队从上往下全面推行。在团队规模比较小的时候，比如少于 20 个开发者，这个方法很好用，能够快速推动。但如果团队规模较大，成员对好的

代码审查方法又不是很熟悉的话，往往会造成混乱。

比如，大家会倾向于按照自己的想法进行审查，这就可能会出现审查者只检查格式，或者把个人喜好强加到别人身上的错误做法。

如果在一开始的时候，就出现较多的负面效果的话，会让大家丧失推行代码审查的信心。

所以，我推荐先不要大范围实施，而是应该先选择试点团队，之后再推广。这样做的好处是，一方面，试点团队成员有限，容易推行新的做法；另一方面，有了试点团队的成功案例后，再全面推行就有了可借鉴的经验，会更顺畅。

关于试点团队的选择，最好是找成员经验丰富、对技术有追求，同时所做业务又不是最紧急的小团队。这样的团队成员不仅会有兴趣，而且也有精力去学习好的审查实践，容易出成果。

有了试点团队，我们接下来就需要选择代码审查工具及配置流程了。

3. 选择代码审查工具，并把机器审查和人工审查结合起来

我在前一篇文章中提到过，几乎所有团队都适合使用工具进行异步的一对一审查。

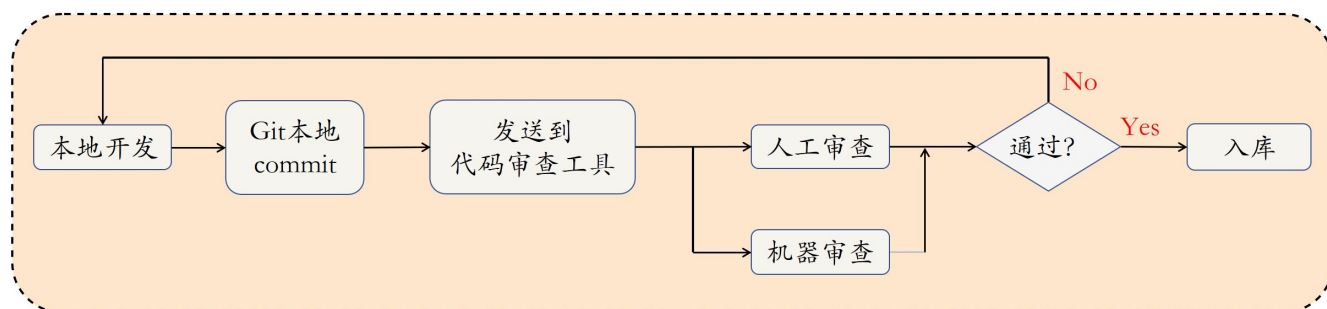
关于工具，如果你的团队本来已经在使用 GitLab、GitHub、Gerrit、Phabricator 管理代码的话，那么很容易上手代码审查。因为，GitHub、GitLab 有基于 PR 的审查。而 Gerrit 和 Phabricator 本身就主打代码审查的功能。

如果你所在团队没有使用这些工具的话，就只能投入资源引入一个新工具，可以是上面提到的几个工具，也可以是其他审查工具，比如[Review Board](#)。

至于代码审查工具的设置，你可以在网上搜索到详细的配置指南。这里我就不一一列举。

我在这里着重讲讲**配置流程中非常重要的一步，就是配置机器审查和人工审查配合的工作流**。代码审查是代码入库前质量保证的重要一环，所以通常是和 CI 工具配合使用。最好能够让机器自动化地完成关于代码风格、静态检查、单元测试等工作，这样可以让审查者把最宝贵的时间投入到逻辑、设计等难以自动化的问题上。这里，我和你分享一种最常见的工作流。

- 第一，将代码提交到本地 Git 仓库或者用于审查的远端 Git 服务器的分支上；
- 第二，把 commit 提交给代码审查工具；
- 第三，代码审查工具开始进行机器审查和人工审查；
- 第四，如果审查通不过就打回重做，开发者修改后重新提交审查，直到审查通过后代码入库。



至于具体的工具集，我这里给出两个例子：

第一个例子是 Facebook 采用的方式。他们使用 Phabricator 作为代码审查工具。同时直接使用原生的 Git Server 作为代码仓服务器。用户将改动提交到 Phabricator，然后进行机器和人工审查。检查通过后，Phabircator 负责将代码推送到 Git Server 上，或者用户手动将本地改动 Push 到 Git Server 上。

这个工作流使用的是原生 Git Server 管理主仓，如果你要使用 GitLab 或者 GitHub 也可以。关于机器审查的配置，Phabricator 提供大量的钩子和插件机制，非常方便。具体细节你可以参考[Phabricator 官方文档](#)。

第二个例子是使用 GitLab、Jenkins 和 SonarQube 进行配置。具体使用 GitLab 管理代码，代码入库后通过钩子触发 Jenkins，Jenkins 从 GitLab 获取代码，运行构建，并通过 Sonar 分析结果。这里有一篇[不错的文章](#)供你参考。

好了，以上就是今天文章的第一部分，引入代码审查的三个步骤：一是，就代码审查的工作量达成共识；二是，选择试点团队；三是，确定审查工具和流程。接下来，我们来看本文的第二大部分：成功推进代码审查的关键操作。

成功推进代码审查的关键操作

代码审查需要注意的点有很多，网上也有很多文章列举了很多最佳实践。从我在 Facebook 的经验来看，其中有两个实践最是直接、明确，而且有效：一是注意审查提交的原子性，二是审查中关注提交说明（Commit Message）。

操作一：提高提交的原子性

代码提交的原子性，是指一个提交包含一个不可分割的特性、修复或者优化，同时这个提交要尽可能小。

原子性提交的优点是，结构清晰，容易定位问题。一般来说，代码提交的原子性做得越好，代码质量越好。同时，原子性提交因为小而聚焦，也是做好代码审查的基础。

我曾经和一个 10x 开发者合作一个大功能。他将其拆为了 15 个原子性提交，每一个提交的代码量都控制在 500 行以下。这 15 个提交都是我审查的，在 3 天的协作过程中，这个同事不断发出审核 PR，我不断反馈修改意见，然后他不断对旧的提交进行修改，同时又发出新的 PR 请求。

大多数时候，都是三四个提交在被同时审查，而这三四个提交是实现同一个功能。整个审查过程非常流畅，效率很高。试想一下，如果不是代码提交的原子性做得很好，绝对难以做到这一点。

这段经历给我的触动非常大，让我深刻意识到了原子性提交对代码审查的重要性。

所以在 Facebook，代码提交的原子性是代码审查非常重要的指标，如果提交的原子性不好，常常会被直接打回。我后来在 Stand 公司就采用了这个办法，让大家在审查中首先看原子性。如果一个提交做了几件事，不看细节直接打回。因为团队小，很快就形成习惯，效果很棒。我推荐你在你们团队也尝试类似办法。如果能接受一开始的阵痛并执行下来的话，很快就会看到效果。

另外，这里还有一个**实现原子性提交的技巧**：在对一个大功能进行原子性拆分的时候，**功能开关是一个很好的工具**。这里有一个马丁·福勒（Martin Fowler）的关于[功能开关](#)的链接，可供你参考。

操作二：提高提交说明的质量

提交说明是提高代码审查的利器，好的格式应该包含以下几个方面：


标题，简明扼要地描述这个提交。这部分最好在 70 个字符之内，以确保在单行显示的时候能够显示完整。比如，在命令行常用的 `git log --oneline` 输出结果要能显示完全。

详细描述，包括提交的目的、选择这个方法的原因，以及实现细节的总结性描述。这三个方面的内容最能帮助审查者阅读代码。

测试情况，描述的是你对这个提交做了什么样的测试验证，具体包括正常情况的输出、错误情况的输出，以及性能、安全等方面的专项测试结果。这部分内容，可以增加审查者对提交代码的了解程度以及信心。

与其他工具和系统相关的信息，比如相关任务 ID、相关的冲刺（sprint，也可翻译为“迭代”）链接。这些信息对工具的网状互联提供基础信息，非常重要。

这里还有一个 Git 的技巧是，**你可以使用 Git 的提交说明模板（Commit Message Template），来帮助团队使用统一的格式**。比如，可以像下面这个例子一样，使用 `git config --global commit.template` 命令来设置模板。

 复制代码

```
1 # 配置文件：提交说明模板文件 ~/.git_commit_msg.txt
2 > cat .git_commit_msg.txt
3
4
5
6
7 Summary:
8
9
10 Test:
11
12
13 Task ID:
14
15
16 # 设置上述文件为提交说明模板。
17 > git config --global commit.template ~/.git_commit_msg.txt
18
19
20
21
22 # 使用实例：之后 git commit 命令自动使用上述模板
23 > git add app.js
24 > git commit
25
26
27 Summary:
28
29
30 Test:
31
32
```



```
33 Task ID:
34
35
36 # Please enter the commit message for your changes. Lines starting
37 # with '#' will be ignored, and an empty message aborts the commit.
38 #
39 # On branch master
40 # Your branch is up to date with 'origin/master'.
41 #
42 # Changes to be committed:
43 #       modified
```

通过制定严格的提交说明格式来规范其质量，可以方便审查者查理解被审查代码的意图、实现思路，并通过测试情况，加快对代码的理解，提高对代码质量的信心，从而大大提高审查者的效率。同时，严格的提交说明格式及好的说明质量也可以督促开发者提高代码质量。所以说，它是一个简单、直观且有效的代码审查落地实践。

下面，我就给出一个之前我在一个公司逐步提高提交说明要求的一个具体例子，供你参考。

当时的情况是，大家的提交说明非常简短，很多只包含一句话。比如，功能提交常常写个“实现 A、B 功能”；Bug 修复提交则更简单，通常就三个字母“Fix”。针对这个情况，我采取了以下三个步骤：

第一步，规定提交说明一定要包括标题、描述和测试情况三部分，但暂时还不具体要求必须写多少字。比如，测试部分可以简单写一句“没有做测试”，但一定要写。如果格式不符合要求，审查者就直接打回。这个格式要求工作量很小，比较容易做到，两个星期后整个团队就习惯了。虽然只是在提交说明里增加了简单描述，也已经为审查者和后续工作中进行问题排查提供一些必要信息，所以大家也比较认可这个操作。

第二步，要求提交说明必须详细写明测试情况。如果没有做测试一定要写出具体理由，否则会被直接打回。这样做，不但为审查者提供了方便，还促进了开发人员的自测。整个团队在一个多月后，也养成了详细描述测试情况的习惯。

第三步，逐步要求提交的原子性。我要求每一个提交要在详细描述部分描述具体实现了哪些功能。如果一个提交同时实现了多个功能，那就必须解释为什么不能拆开提交；如果解释不合理的话，提交直接被打回。

这一步实施起来比较困难，原因包括大家对功能拆分不习惯或者不熟悉、对 Git 操作不熟悉。针对这些问题，我通过内部培训来提高团队的 Git 能力。同时，我先在一个小团队内集中精力实践提交的原子性，通过我的直接参与来快速提高这个小团队的能力。然后，再让这个团队帮助其他团队提高。

大概三个月以后，整个团队在提交原子性方面提高了很多，代码审查也就真正有效地做起来了。

可以看到，在这个过程中，提交说明起到了抓手的作用，有效地帮助团队推进开发自测，以及提高代码原子性。

好了，以上内容就是成功推进代码审查的两个关键操作，即提高提交的原子性，以及使用提交说明做抓手。最后，我们再来看看成功推行代码审查在文化方面的两个原则。

成功推行代码审查的两个关键原则

代码审查原则一：互相尊重

代码审查是两个开发者之间的技术交流，双方都要谨记互相尊重的原则。

从代码作者的角度来看，审查者要花费时间去阅读他并不熟悉的代码，来帮助你提高，应该尽量为审查者提供方便。

比如，提高提交说明的质量，就是对审查者最基本的尊重。还有，如果你的代码都没有进行自测就提交审查，你觉得审查者心里会怎么想呢？又比如，如果你提交的一个审查有一万行代码，让审查者怎么看呢？所以，代码作者一定要替审查者着想，帮助审查者能够比较轻松地完成审查。

这里还有一些细节性的问题：

1. 注意描述文字的格式。比如，使用 Markdown 格式书写，在 GitHub、GitLab 等工具上就会比较美观。这些格式方面的问题，可能有的开发者会觉得麻烦而不屑于去做。但实际上，这样的细节，会让审查者更愿意也更容易去阅读你的提交说明，提高代码审查的效率。这也是对审查者的尊重。
2. 在描述测试情况的时候，尽量提供真实的输出，如果是 UI 改动的话，最好能够提供截图。提交说明只支持文字，但你可以把图片上传到其他地方，然后提供链接。这样审查

者可以更直观的看到修改效果，对审查效率的提高有非常大的帮助。

3. 如果需要审查者特别注意某一方面，要明确指出。如果有些代码过于复杂，可以主动找审查者当面讨论。

从审查者的角度来看，在提出建议的时候，一定要考虑代码作者的感受。最重要的一点是，不要用一些主观标准来评判别人的代码。

在 Facebook 的时候，我团队里有一个同事对一些技术的细节特别坚持己见。本来两个实现方式的效果差不多，设计也各有优劣，但他要求作者一定要按照他的思路来实现。同时，他的语言能力特别强，常常在讨论里面长篇大论地写他的理由，让代码作者非常头痛，降低了大家的研发效能。最后还是大家都在绩效考评时给他提意见，他才改了一些。

尊重代码作者的做法，还有：

1. 在打回提交的时候，一定要礼貌地描述原因。
2. 审查要尽量及时，如果不能及时审查要告知作者。

这些都只是互相尊重的一些具体实践。在做代码审查时，我建议你随时记得要互相尊重，多为对方考虑。只有这样，代码审查才能顺畅。

代码审查原则二：基于讨论

代码审查常常出现问题的一个地方是，在审查过程中因为意见不同而产生争执甚至争吵，所以一定记住**代码审查的目的是讨论，而不是评判**，作为管理者一定要在团队中强调这个原则。

讨论的心态，有助于放下不必要的自尊心，从而顺利地进行技术交流，提高审查效率。另外，讨论的心态也能促进大家提早发出审查，从而尽早发现结构设计方面的问题。

在 Facebook 时，我们常常会发出一些目的只是讨论的代码审查，讨论之后会抛弃这个提交，然后重新发出新的代码，效果非常不错。

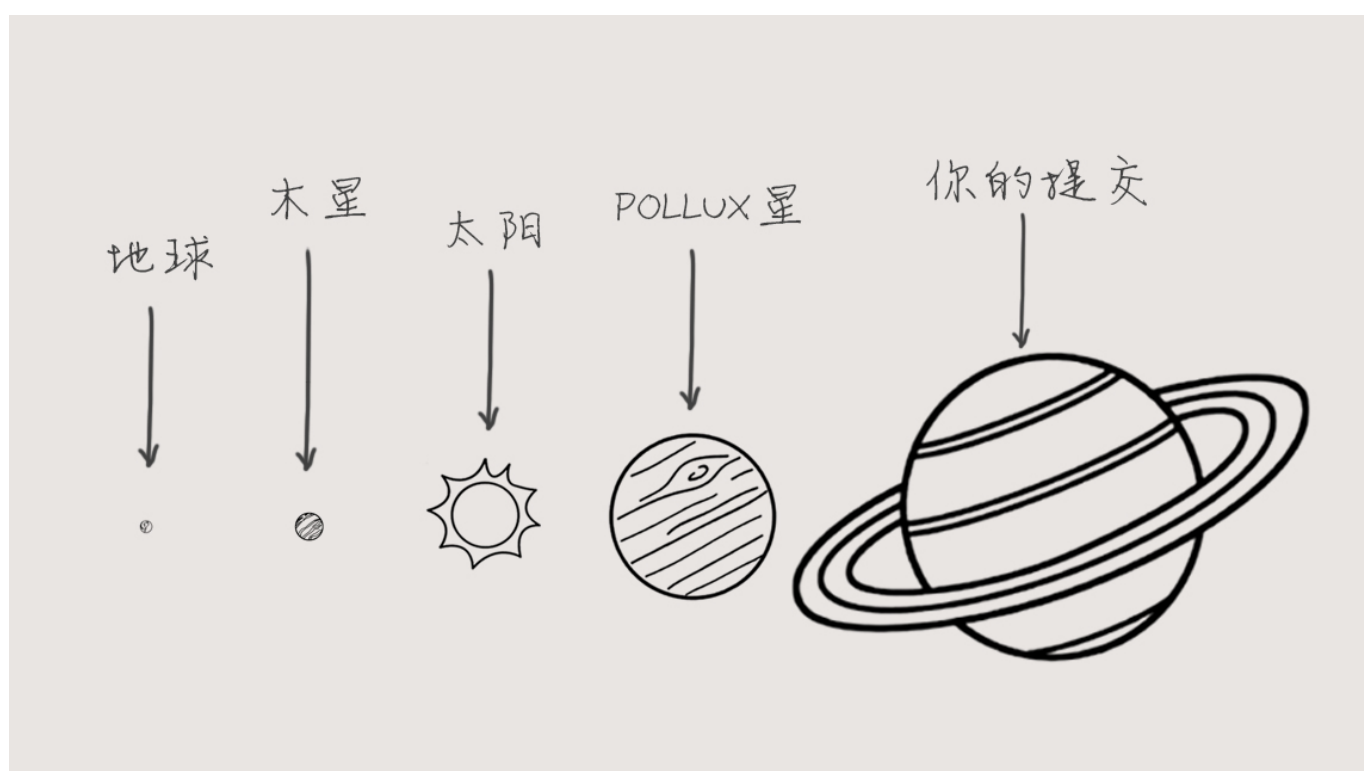
另外，我还有一些关于讨论的建议：

审查者切记不要说教，说教容易让人反感，不是讨论的好方法。

审查者提意见即可，不一定要提供解决方法。我曾经见过一个团队要求提出问题必须给出对应的答案，结果是大家都不愿提问题了。

想办法增加讨论的有趣性。在 Facebook 做代码审查的时候，我们常使用图片进行讨论，用有趣的方式表达自己的意见。这样做有两个好处，一是容易被对方接受，二是开发工作比较枯燥，我们应该主动找点乐趣，你说对不对？

比如说，如果觉得代码提交太大，审查者就可能会贴一张有很多星球的图片。星球按由小到大的顺序排成一行，最左边的是地球，右边是木星，再右边是太阳。天体越来越大，最右边一个超级大，但是标签上写的不是星球的名字，而是“你的代码提交”。大概是这样一幅图：



以上就是两个文化相关的原则。互相尊重和基于讨论。只有这样，才能实现代码作者和审查者的双赢：代码作者的代码质量得到提高，代码审查者能顺畅审查代码。

小结

几乎所有的开发团队都适合使用工具进行代码审查。无论团队大小如何，都可以通过合适的代码审查进行高性价比的讨论。

在今天的这篇文章中，我针对代码审查的引入、推进，以及文化原则三个方面，给出了一些建议。

在引入阶段，我有三个建议：

1. 团队统一思想，代码审查是有效工作的一部分，应该计算到工作量里面；
2. 选择合适的试点团队；
3. 让机器审查和人工审查结合，使得人工审查更聚焦。

在推进实施的阶段，我推荐提高提交的原子性，以及重视使用提交说明两个关键操作。

最后，我建议通过互相尊重和基于讨论这两个原则，从文化的角度固化团队的代码审查实践。

这三个方面的建议，是我基于 Facebook 的经验总结得出的，是 Facebook 高效代码审查的重要原则和方法，可以帮助你在团队中推动高效代码审查。

同时，这三个方面的措施也分别对应了在一个团队引入、推进、深化代码审查的步骤。希望对你在团队中引入代码审查的具体过程有一定的借鉴作用。

思考题

1. 你见过或者经历过推行代码审查的成功或者失败案例吗？你觉得成功或失败的原因是什么呢？
2. 上面提到 GitHub、GitLab、Gerrit 都是用 Git 分支来存储被审核的代码。只有 Phabricator 使用数据库存储。你知道为什么吗？（提示：Phabricator 里没有“Git”这三个字母。）

感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再见！

研发效率破局之道

Facebook 研发效率工作法

葛俊

前 Facebook 内部工具团队 Tech Lead



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 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 12 | 代码审查：哪种方式更适合我的团队？

下一篇 14 | 质量与速度的均衡：让“唯快不破”快得更持久

精选留言 (2)

写留言



john_zhang

2019-09-20

我们推行过一段时间代码审查，因为三五个人，所以采用的是团体审查，每天半个小时左右，可惜后来开发进度赶，慢慢就没做了，现在开发同事总是以进度赶为由，不太认同代码审查，怎么破？

1

5



Alick

2019-09-22

葛老师，能否介绍下 Facebook 代码评审的绩效考评方法？

基于评审意见数，觉得纬度单一；基于问题严重性，有易起纷争的担忧；

采取何种代码评审绩效评价方式，能够起到正向积极导向效果？

展开 ∨

