Code Review最佳实践

宝玉 朱小厮的博客 昨天

我一直认为Code Review（代码审查）是软件开发中的最佳实践之一，可以有效提高整体代码质量，及时发现代码中可能存在的问题。包括像Google、微软这些公司，Code Review都是基本要求，代码合并之前必须要有人审查通过才行。

然而对于我观察到的大部分软件开发团队来说，认真做Code Review的很少，有的流于形式，有的可能根本就没有Code Review的环节，代码质量只依赖于事后的测试。也有些团队想做好代码审查，但不知道怎么做比较好。

网上关于如何做Code Review的文章已经有很多了，这里我结合自己的一些经验，也总结整理了一下Code Review的最佳实践，希望能对大家做好Code Review有所帮助。

Code Review有什么好处？

很多团队或个人不做Code Review，根源还是不觉得这是一件有意义的事情，不觉得有什么好处。这个问题要从几个角度来看。

- 首先是团队知识共享的角度

一个开发团队中，水平有高有低，每个人侧重的领域也有不同。怎么让高水平的帮助新人成长？怎么让大家都对自己侧重领域之外的知识保持了解？怎么能有人离职后其他人能快速接手？这些都是团队管理者关心的问题。

而代码审查，就是一个很好的知识共享的方式。通过代码审查，高手可以直接指出新手代码中的问题，新手可以马上从高手的反馈中学习到好的实践，得到更快的成长；通过代码审查，前端也可以去学习后端的代码，做功能模块A的可以去了解功能模块B的。

可能有些高手觉得给新手代码审查浪费时间，自己也没收获。其实不然，新人成长了，就可以更多的帮高手分担繁重的任务；代码审查中花时间，就少一些帮新人填坑擦屁股的时间；良好的沟通能力、发现问题的能力、帮助其他人成长，都是技术转管理或技术上更上一层楼必不可少的能力，而通过代码审查可以有效的去练习这些方面的能力。

- 然后是代码质量的角度

现实中的项目总是人手缺进度紧，所以被压缩的往往就是自动化测试和代码审查，结果影响代码质量，欠下技术债务，最后还是要加倍偿还。

也有人寄希望于开发后的人工测试，然而对于代码质量来说，很多问题通过测试是测试不出来的，只能通过代码审查。比如说代码的可读性可维护性，比如代码的结构，比如一些特定条件才触发的死循环、逻辑算法错误，还有一些安全上的漏洞也更容易通过代码审查发现和预防。

也有人觉得自己水平高就不需要代码审查了。对于高手来说，让别人审查自己的代码，可以让其他人学习到好的实践；在让其他人审查的同时，在给别人说明自己代码的时候，也等于自己对自己的代码进行了一次审查。这其实就跟我们上学时做数学题一样，真正能拿高分的往往是那些做完后还会认真检查的。

•还有团队规范的角度

每个团队都有自己的代码规范，有自己的基于架构设计的开发规范，然而时间一长，就会发现代码中出现很多不遵守代码规范的情况，有很多绕过架构设计的代码。比如难以理解和不规范的命名，比如三层架构里面UI层绕过业务逻辑层直接调用数据访问层代码。

如果这些违反规范的代码被纠正的晚了，后面再要修改就成本很高了，而且团队的规范也会慢慢的形同虚设。

通过代码审查，就可以及时的去发现和纠正这些问题，保证团队规范的执行。

关于代码审查的好处，还有很多，也不一一列举。还是希望能认识到**Code Review和写自动化测试一样，都是属于磨刀不误砍柴工的工作，在上面投入一点点时间，未来会收获代码质量，会节约整体的开发时间。**

该怎么做？

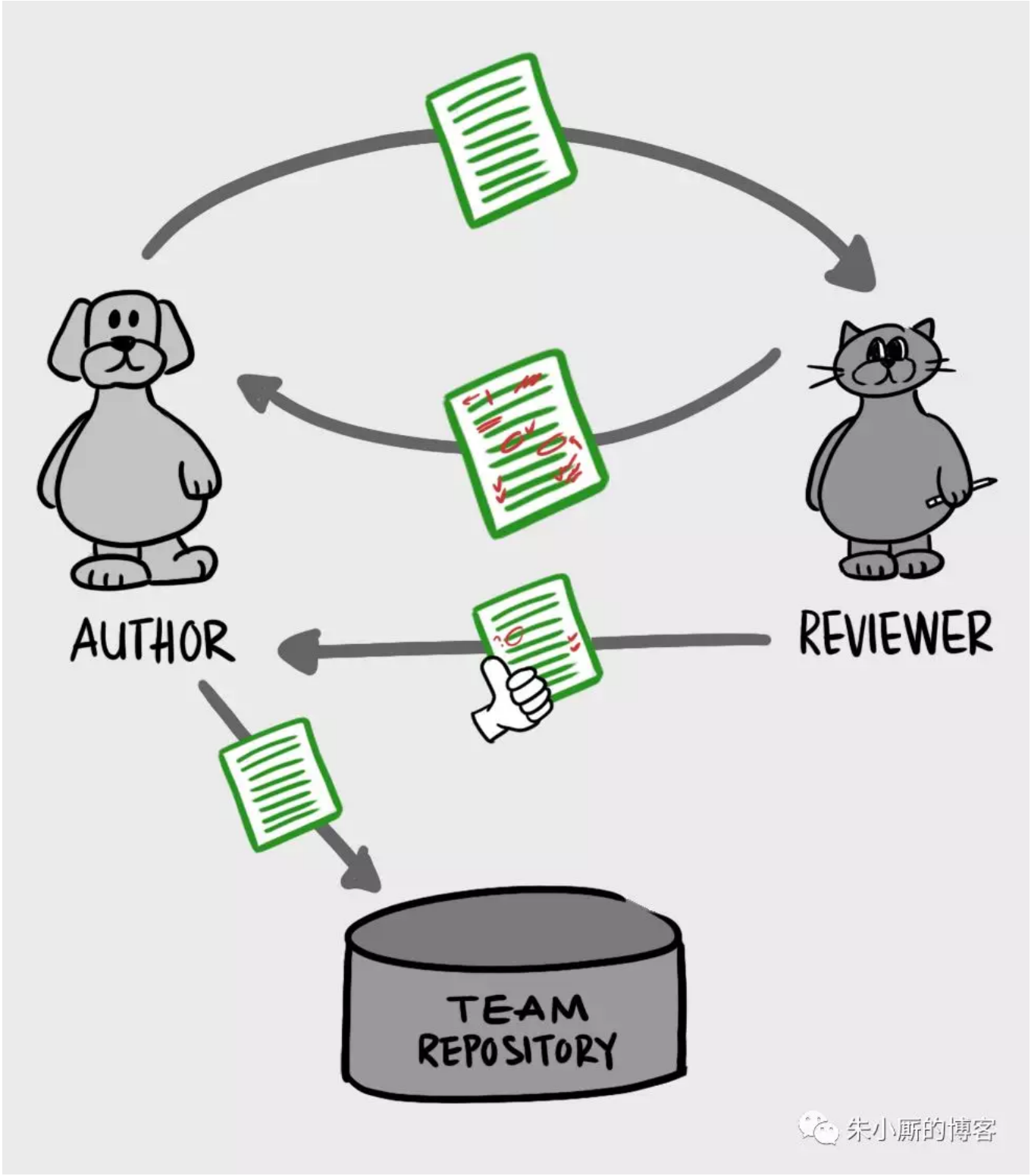
现在很多人都已经有意识到Code Review的重要性了，只是苦于不知道如何去实践，不知道怎么样算是好的Code Review实践。

把Code Review作为开发流程的必选项而不是可选项

在很早以前，我就尝试过将代码审查作为代码流程的一部分，但只是一个可选项，没有Code Review也可以把代码合并到master。这样的结果就是想起来才会去做Code Review，去检查的时候已经有了太多的代码变更，审查起来非常困难，另外就算审查出问题，也很难得以修改。

我们现在对代码的审查则是作为开发流程的一个必选项，每次开发新功能或者修复Bug，开一个新的分支，分支要合并到master有两个必要条件：

•所有的自动化测试通过•有至少一个人Code Review通过，如果是新手的PR，还必须有资深程序员Code Review通过



图片来源：How to Do Code Reviews Like a Human

这样把Code Review作为开发流程的一个必选项后，就很好的保证了代码在合并之前有过Code Review。而且这样合并前要求代码审查的流程，好处也很明显：

- 由于每一次合并前都要做代码审查，这样一般一次审查的代码量也不会太大，对于审查者来说压力也不会太大
- 如果在Code Review时发现问题，被审查者希望代码能尽快合并，也会积极的对审查出来的问题进行修改，不至于对审查结果太过抵触

如果你觉得Code Review难以推行，不妨先尝试着把Code Review变成你开发流程的一个必选项。

把Code Review变成一种开发文化而不仅仅是一种制度

把Code Review 作为开发流程的必选项后，不代表Code Review这件事就可以执行的很好，因为Code Review 的执行，很大部分程度上依赖于审查者的认真审查，以及被审查者的积极配合，两者缺一不可！

如果仅仅只是当作一个流程制度，那么就可能会流于形式。最终结果就是看起来有Code Review，但没有人认真审查，随便看下就通过了，或者发现问题也不愿意修改。

真要把Code Review这件事做好，必须让Code Review变成团队的一种文化，开发人员从心底接受这件事，并认真执行这件事。

要形成这样的文化，不那么容易，也没有想象的那么难，比如这些方面可以参考：

- 首先，得让开发人员认识到Code Review这件事为自己、为团队带来的好处•然后，得要有几个人做好表率作用，榜样的力量很重要•还有，对于管理者来说，你激励什么，往往就会得到什么•最后，像写自动化测试一样，**把Code Review要作为开发任务的一部分，给审查者和被审查者都留出专门的时间去做这件事**，不能光想着马儿跑得快又舍不得给马儿吃草

如何形成这样的文化，有心的话，还有很多方法可以尝试。只有真正让大家都认同和践行，才可能去做好Code Review这件事。

一些Code Review的经验技巧

在做好Code Review这件事上，还有一些经验技巧可以参考。

选什么工具辅助做CODE REVIEW？

现在很多源代码管理工具都自带Code Review工具，典型的像Github、Gitlab、微软的Azure DevOps，尤其是像Gitlab，还可以自己在本地搭建环境，根据自己的需要灵活配置。

配合什么样的开发流程比较好？

像Github Flow[1]这样基于分支开发的流程是特别适合搭配Code Review的。其实不管什么样的开发流程，关键点在于代码合并到master（主干）之前，要先做Code Review。

真遇到紧急情况，来不及代码审查怎么办？

虽然原则上，必须要Code Review才能合并，但有时候确实会存在一些紧急情况，比如说线上故障补丁，而又没有其他人在线，那么这种情况下，最好是在任务管理系统中，创建一个Ticket，用来后续跟踪，确保后续补上Code Review，并对Code Review结果有后续的代码更新。

先设计再编码

有些新人发现自己的代码提交PR（Pull Request）后，会收到一堆的Code Review意见，必须要做大量的改动。这多半是因为在开始做之前，没有做好设计，做出来后才发现问题很多。

建议在做一个新功能之前，写一个简单的设计文档，表达清楚自己的设计思路，找资深的先帮你做一下设计的审查，发现设计上的问题。设计上没问题了，再着手开发，那么到Review的时候，相对问题就会少很多。

代码在提交CODE REVIEW之前，作者要自己先REVIEW和测试一遍

我在做代码审查的时候，有时候会发现一些非常明显的问题，有些甚至自己都没有测试过，就等着别人Code Review和测试帮助发现问题。这种依赖心理无论是对自己还是对团队都是很不负责任的。

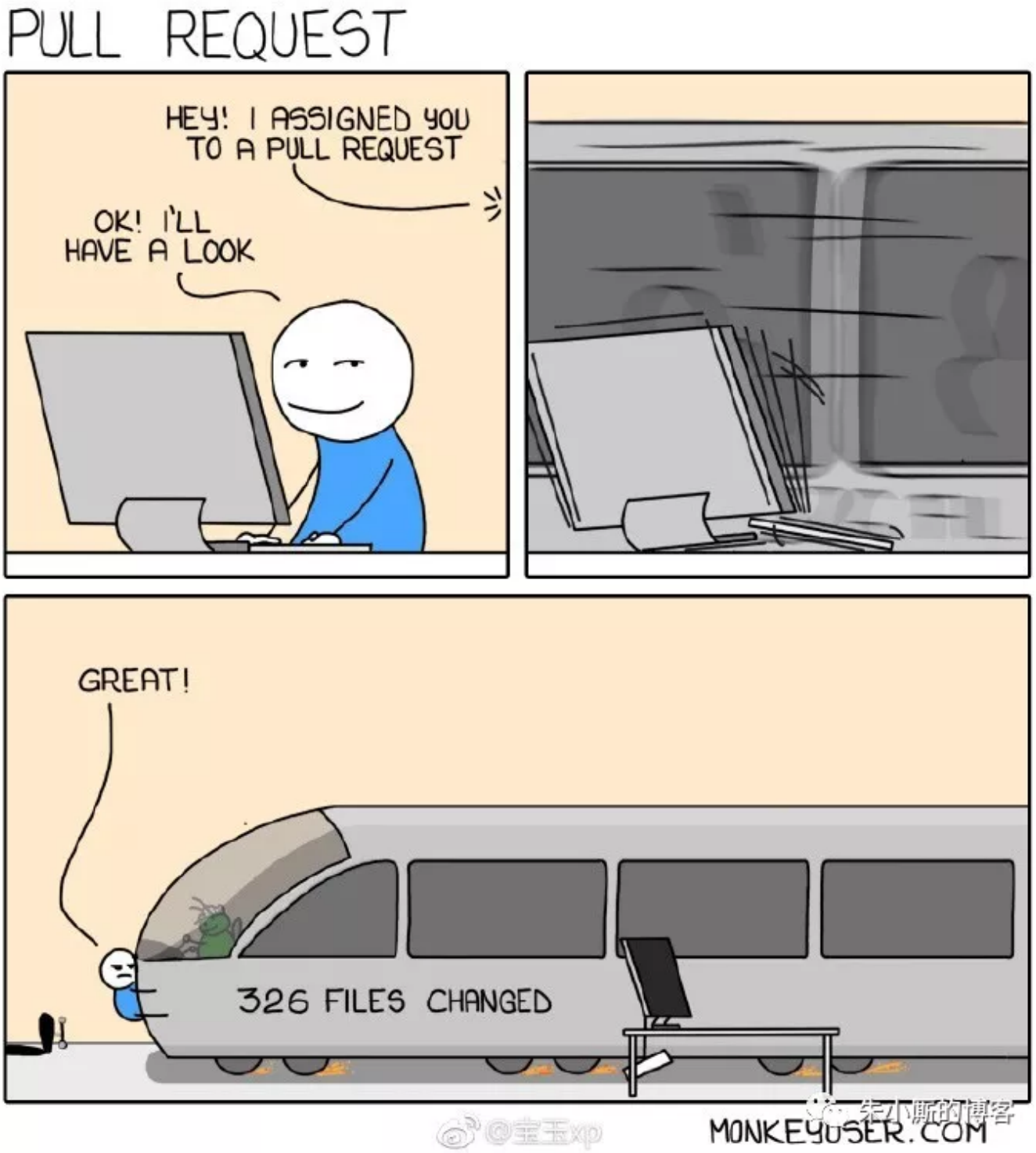
一个好的开发人员，代码在提交Code Review之前，肯定是要自己先Review一遍，把该写的自动化测试代码写上，自己把基本的测试用例跑一遍的。

我对于团队提交的PR，有个要求就是要在PR的描述中增加截图或者录屏，就是为了通过截图或者录屏，确保提交PR的人自己是先测试过的。这也是一个有效的辅助手段。

PR要小

在做Code Review的时候，如果有大量的文件修改，那么Review起来是很困难的，但如果PR比较小，相对就比较容易Review，也容易发现代码中可能存在的问题。

所以在提交PR时，PR要小，如果是比较大的改动，那么最好分批提交，以减轻审查者的压力。



对评论进行分级

在做Code Review时，需要针对审查出有问题的代码行添加评论，如果只是评论，有时候对于被审查者比较难甄别评论所代表的含义，是不是必须要修改。

- 建议可以对Review的评论进行分级，不同级别的结果可以打上不同的Tag，比如说：
- [blocker]: 在评论前面加上一个blocker标记，表示这个代码行的问题必须要修改[optional]：在评论前面加上一个[optional]标记，表示这个代码行的问题可改可不改

- [question]：在评论前面加上一个[question]标记，表示对这个代码行不理解，有问题需要问，被审查者需要针对问题进行回复澄清

类似这样的分级可以帮助被审查者直观了解Review结果，提高Review效率。

评论要友好，避免负面词汇；有说不清楚的问题当面沟通

虽然评论是主要的Code Review沟通方式，但也不要过于依赖，有时候面对面的沟通效率更高，也容易消除误解。

另外文明用语，不要用一些负面的词汇。

总结

Code Review是一种非常好的开发实践，如果你还没开始，不妨逐步实践起来；如果已经做了效果不好，不妨对照一下，看有没有把Code Review作为开发流程的必选项而不是可选项？有没有把Code Review变成一种开发文化而不仅仅是一种制度？

> 作者：宝玉
> 原文：http://t.cn/AiIlWVbO

References

[1] Github Flow: *https://guides.github.com/introduction/flow/*