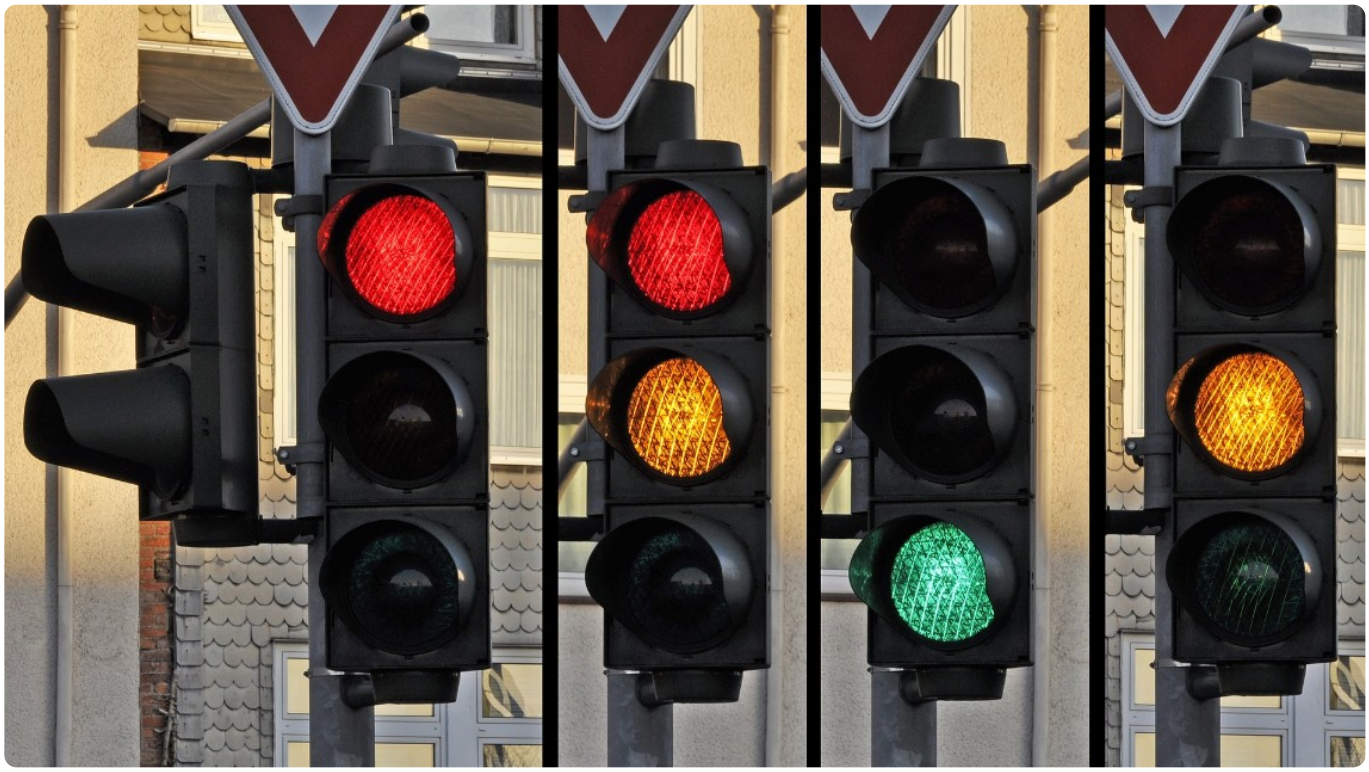
12 | 代码审查：哪种方式更适合我的团队？

2019-09-18 葛俊

研发效率破局之道

进入课程 >



讲述：葛俊

时长 20:10　大小 18.47M

你好，我是葛俊。今天，我们来聊聊都有哪些代码审查方式，以及哪种方式更适合你的团队。

国外互联网行业的很多效能标杆公司都非常重视代码审查（Code Review），比如 Facebook、Google 等就要求每一个提交都必须通过审查。

现在，国内的很多公司也在有意无意地引入代码审查：有的团队直接使用代码仓库管理工具提供的审查功能，比如 GitHub、GitLab 提供的 PR 审查；有的团队则使用专门的审查工具，比如 Phabricator、Gerrit；还有些团队采用面对面检查；甚至有少数公司，尝试使用结对编程的方式进行代码审查。

虽然国内公司在代码审查上做了不少尝试，也有一些公司做得比较好，比如我了解到七牛云就做得不错，但大多数国内公司还是对代码审查理解得不够深入，对审查方法的认识也不够全面，只能简单地去追随一些最佳实践。结果就是，有些团队的代码审查推行不下去，半途而废；有的则流于形式，花了时间却看不到效果。

那么，怎样才能做好代码审查呢？

我认为，**做好代码审查的一个前提条件就是，要找到适合自己团队的方法。**要做到这一点，你就要对代码审查有一个深入的了解，弄清楚常用方式及适用场景，然后才能做出正确选择。

所以，在今天这篇文章里，我首先会和你分享常用的代码审查方式及其适用场景，然后和你分享几个具体案例。

什么是代码审查，有什么作用？

首先，我们来看看代码审查的定义。代码审查是指代码作者以外的其他人对代码进行检查，以寻找代码的缺陷和可以提高的地方。

这里需要注意的是，按照定义，**人工检查才是代码审查。**机器进行的检查一般叫作代码检查或者代码自动检查。常见的办法是人工审查和机器检查同时进行，我会在后面和你详细讨论这部分内容。

代码审查的作用很多，主要表现在 5 个方面。

作用 1，尽早发现 Bug 和设计中存在的问题。我们都知道，问题发现得越晚，修复的代价越大。代码审查把问题的发现尽量提前，自然会提高效能。

作用 2，提高个人工程能力。不言而喻，别人对你的代码提建议，自然能提高你的工程能力。事实上，仅仅因为知道自己的代码会被同事审查，你就会注意提高代码质量。

作用 3，团队知识共享。一段代码入库之后，就从个人的代码变成了团队的代码。代码审查可以帮助其他开发者了解这些代码的设计思想、实现方式等。另外，代码审查中的讨论记录还可以作为参考文档，帮助他人理解代码、查找问题。

作用 4，针对某个特定方面提高质量。一些比较专业的领域，比如安全、性能、UI 等，可以邀请专家进行专项审查。另外，一些核心代码或者高风险代码，也可以通过团队集体审查的方式来保证质量。

作用 5，统一编码风格。这，也是代码审查的一个常见功能，但最好能通过工具来实现自动化检查。

以上就是代码审查的 5 个主要作用。接下来，我会与你详细介绍具体的审查方法。在这部分内容中，我会针对每一种方法详细讨论其优缺点及适用场景，并根据我的经验给出建议。这是本篇文章的中心内容，希望你可以多花些精力好好理解消化。

根据团队特点选择代码审查方法

代码审查有多种方法，按照不同的维度可以有不同的分类。

按审查方式分类

按照审查方式，代码审查可以分为工具辅助的线下异步审查和面对面审查两类。

工具辅助的线下异步审查，就是代码作者通过工具将代码发送给审查者，审查者再通过工具把反馈传递给作者。比如，GitHub、GitLab、Gerrit、Phabricator 等工具的审查都是这种方式。这是当前最常见的审查方式，应用灵活的话，可以在任何一个团队取得良好的效果。

使用这种方式的优点，主要表现在两个方面：

一是，审查者可以控制审查时间，减少对自己工作的干扰。比如，审查者可以用一两个小时的时间，来集中处理多个代码审查要求。

二是，在工具中的讨论会留下文档，方便以后参考。

相应地，使用这种方式的缺点是实时性不好。但，这个问题很容易解决。比如，你可以在工具上发出审查要求，然后跑到审查者办公桌前进行面对面讨论。

面对面审查，就是审查者和代码作者在一块儿阅读代码，进行实时讨论。这种方式的好处是快，可以高效地审查不方便用文字讨论的代码。比如，架构问题的讨论就很适合用这种方

式。面对面审查的一个极端例子是结对编程，也就是两个人同时写代码，可以说是把实时性发挥到了极致。

但，面对面审查的主要问题，就是对审查者的干扰大，如果大量使用的话会影响审查者的心流。一个降低干扰的方法是，提前约时间。

10 年之前我在微软工作的时候，我们就常常会使用面对面审查的方法。但除了特殊情况以外，大家都需要提前预约才能去找审查者进行面对面审查。总的来说，这种方式还是可以接受的。

按审查人数分类

按审查人数，代码审查可以分为一对一审查和团队审查两类。

一对一审查，就是审查者单独进行审查。除了面对面的一对一审查，基于工具的异步审查是最常见的一对一审查。

这种方式的好处是，安排起来容易，对审查者的干扰小。不过，代码作者一般不会只把代码发给一个人审查，而是会同时发给几个人，避免出现因为单个审查者最近没空而耽误审查速度的情况。

需要注意的是，这种方式，可能会出现几个审查者同时审查一段代码的现象，从而造成重复劳动。常见的解决办法有两个：

一种是，代码作者明确邀请审查者检查代码的不同部分或者不同方面；

另一种是，审查者在开始审查时通知其他审查者，确保不会重复审查。

团队审查，就是团队成员聚在一起讨论一些代码。这种方式适合专项讨论，比如团队成员集体讨论关键代码。具体方式常常是面对面在会议室进行，当然有的团队也会使用电话会议。

团队审查的好处是，大家一起讨论可以检查出更多问题，但缺点也比较明显，因为要开会，所以很容易出现会议效率不高的通病。解决这个问题，至少做到以下四点：

一是，会前做好准备。也就是，组织者提前发出审查要求和要点，要求每个参会者会前阅读。

二是，增加一个协调人员，以确保会议讨论能聚焦和按部就班地进行。

三是，要有会后跟进，最好能够把文字记录存档到可以搜索到的地方，供以后参考。

四是，尽量减少参会人数。

当然了，这四点适用于所有会议。

按审查范围分类

按照审查范围，代码审查可以分为增量审查和全量审查两类。

代码增量审查，是只针对改动部分进行审查。在一个团队形成代码审查的机制以后，一般会只审查新增代码。

这种方式的好处是，能把有限的时间花在最容易出问题的地方。不过需要注意的是，在审查新的代码改动时，一定要一同考虑相关代码，看看是否会因为新代码的引入造成问题。另外，也需要注意是否会有旧的代码，可以被重构掉。

目前来看，这种方法没有什么明显的缺点。

代码全量审查，是对现有代码的全量，比如说整个文件、某几个函数进行审查。

这种方式常见的适用场景有两个：

一是，专项检查；

二是，在刚开始引入代码审查时，对遗产代码进行一次审查。

这种方式的优点是关注整体质量。但，缺点是工作量大，不能常常进行。

按审查时机分类

按照审查时机，代码审查可以分为代码入库前门禁检查、设计时检查、代码入库后检查三类。

代码入库前门禁检查，是把代码审查作为门禁的一部分，要求代码在入库前必须通过人工审查。这也是最常见的审查方式。比如，GitHub 等工具里面进行的 PR 审查就属于这一类；

在 Gerrit 工具里，代码入库前必须通过打分才能入库，也是典型的门禁场景。

这种审查方式的优势很明显，如果没有流于形式的话，可以在代码入库前的最后一步拦截问题，从而避免入库后昂贵的缺陷修复成本。

但这种方法可能导致一个问题，即太过死板从而降低代码入库效率。比如，如果一个公司严格要求入库前门禁检查，即使是修改一个错别字也必须要完整检查才行，这在进行紧急热修复的时候可能就不合适。

解决这个问题的办法就是，引入灵活的机制。比如，Facebook 通过 Phabricator 审查，有一个办法可以让代码作者绕过审核者直接将代码入库，但是需要通知审核者这样操作的理由。一个常见的合理理由，就是“这个修复很简单，我的把握很大。但又很紧急，需要马上通过热修复上线。另外，现在是凌晨一点半，不想把你叫起来帮我审查。”

代码入库前的设计时检查，就是在设计阶段进行代码审查。这种方式主要是讨论代码的架构设计是否有问题。因为代码还没有成型，修改成本非常小，代码作者的抵触情绪也很小。相反，如果代码写好后再去审查架构设计，一旦发现问题就会改动较大，甚至推倒重来，非常浪费时间。我见到的大多数情况是，为了保证项目进度而不得不放弃修改。

事实上，有调查显示，代码审查更容易发现架构问题而不是 Bug。所以**我的建议是，尽量使用代码设计时审查。**具体的方法是，可以使用与门禁代码检查相同的工具，只不过这个时候发出去的 PR 目的是为了讨论，而不是为了入库。讨论结束后删除这个 PR 即可。

需要强调的是，设计时代码审查的用处巨大，但在实践中却常常因为大家对它不够了解而被忽略。

代码入库后检查，就是检查已经入库的代码。有些工具专门支持事后审查，比如 Phabricator 的审计功能（Audit）。

这种方式的好处是，既不阻塞代码入库，又可以对提交的代码进行审查，只要入库代码没有马上上线，风险就很小。实际上，这就是我们开发审计功能的意图。如果你所使用的工具里没有这个功能，可以在代码历史浏览工具里，用讨论的方式进行。虽然不够方便，但也够用了。

入库后检查的另一个作用是，提高遗产代码的质量。

以上就是代码审查的基本方式、特点及适用场景。掌握了这些，你就可以根据自己团队的特点，选择最合适的方式。最后，我再和你分享三个成功案例，希望给你更多启发。

代码审查方式选择的三个成功案例

下面的这三个案例，团队规模、引入代码审查的时机，以及适用的代码审查工具、方式都有所区别，你可以边看案例，边思考适合自己团队的代码审查方法。

案例一：5 个开发者组成的初创团队的代码审查实践

虽然这个团队只有 5 名开发者，但灵活地做到了高性价比的代码审查。总结他们的经验来看，主要体现在以下几个方面。

从审查方式来看，他们采用的是基于 GitHub 的线下异步审查。这样做的考量以及好处是：

- 他们的代码仓库是 GitHub，使用 GitHub 的 PR 功能直接进行审查。虽然 GitHub 不如专业的代码审查工具方便，但够用了。
- 使用 GitHub 做基本的代码审查所需的配置很少，所以引入工具的工作量也非常小。
- 有的情况使用面对面审查，方法是在代码 PR 发出去之后基于 GitHub 来进行面对面讨论，同时会把几个讨论放到一起，而不是无节制地使用面对面审查，以提高工作效率。

从审查范围来看，从开发第一个 MVP 开始，他们就引入了代码审查，所以后续一直采用代码增量的一对一审查。只是在 App 上线之后，针对安全，对登录模块做了一次全量代码入库后检查。

从审查时机来看，他们并没有强制使用代码入库前门禁检查。但是，他们仍然把代码审查作为一个高优先级的任务来做，要求没有特殊原因都要做代码审查。

除此之外，他们做了力所能及的机器检查，比如通过 GitHub 的钩子运行各种 Linter 以及单元测试和一些集成测试，与人工检查互为补充。

结果就是，这个初创公司从一开始就形成了灵活使用代码审查的文化，提高代码质量的同时，并没有减缓代码入库的速度。

案例二：30 人团队的代码审查实践

这个团队的代码管理工具是 GitLab，没有使用代码审查。因为业务发展太快，代码质量问题越来越严重，所以他们决定引入代码审查。具体来说，做法如下。

他们放弃了 GitLab，直接用 Gerrit 进行代码仓库管理和代码审查。这是因为 GitLab 的代码审查功能不如 Gerrit 强大。而 Gerrit 同时也具备代码仓管理功能，所以就没必要同时维护两个系统了。

从审查方式和审查人数来看，他们采用的是工具辅助的线下一对一审查，对于团队审查非常慎重，只是偶尔使用。

从审查范围来看，他们只是在开始阶段，集中团队核心成员对现有遗产代码进行了几次多对一、面对面的代码全量审查。

从审查时机来看，他们将代码审查作为门禁的一部分，严格执行，以保证业务快速发展时期，上线代码的基本质量。

在机器检查方面，他们使用 Gerrit、Jenkins、SonarQube 三个工具互相集成，自动化了较多的机器检查。

针对之前出现过多次架构问题，但因为发现较晚而来不及修复的情况，团队逐渐引入设计时审查这一实践，尽早发现问题并修复，效果非常不错。

这里，我需要再强调下这个团队引入代码审查的步骤。

他们通过规定严格的提交说明（Commit Message）规范以及 PR 描述规范，来逐步提高代码提交的原子性。也就是说，每个提交只做一件事，同时这个提交又不会太小。这种方法很有效，我会在下一篇文章中和你做进一步介绍。

案例三：百人以上团队的代码审查实践

这个团队原本使用 GitLab 作为 Git 服务器，不过没有做代码审查。在引入代码审查的过程中，为了提高代码审查的效率和体验，他们没有使用 GitLab 做代码审查，而是引入了 Phabricatgor。具体做法是：

使用 Phabricator 的镜像方式进行代码审查。也就是说，代码仓库仍然是 GitLab，Phabricator 上只有一个用来做代码审查的克隆，这样既实现了代码审查，又把对原有

Git 流程的影响降到了最小。

因为团队较大，又分散在多个地区，所以大量使用了线下的异步审查流程。为了保证开发人员在等待一个提交审查的同时，还可以做其他开发工作，他们使用了 Git 的提交链和多分支的方法。关于 Git 的这个使用技巧，我会在后面的文章里与你详细讨论。

基于 Phabricator 进行代码设计时审查，解决因为代码仓库规模大，导致添加新功能时设计复杂也容易有所疏漏的问题。开发人员使用伪代码来表明自己的设计计划，并发出代码审查需求，然后跟审查者进行面对面讨论或者视频会议讨论。

使用代码审查对新人进行培训，通过严格审查新人提交的代码，来传达团队的代码规则、质量基准等。

另外，值得一提的是，在使用 Phabricator 进行代码审查之前，这个团队最常用的是少量的团队集中审查，并且有审查权的只是团队的几个核心成员。这种方式导致了两个问题：一是审查效率低下；二是这几个核心成员本身都是技术骨干，但因为需要花费大量时间做审查，导致无法贡献足够多的新代码。

采用新的代码审查方式之后，降低了团队开会进行代码审查的频率，并且逐步放开了审查权限，解决了代码审查成为瓶颈的问题。

以上就是三个成功案例，相信根据你们公司的实际情况具体分析，你一定能找到合适的方法。

小结

代码审查对团队产出质量、个人技术成长有很多好处。代码审查方式多种多样，根据不同维度可以分为工具辅助线下审查、面对面审查、多对一审查、一对一审查、代码增量审查、代码全量审查、设计时审查、入库前检查、入库后检查等。

我将这些审查方法的优缺点整理为了一张表格，你可以以此为参考，根据团队实际情况去挑选合适的方式。

分类方式	审查方法	优点	缺点
审查方式	线下异步审查	1. 时间灵活 2. 干扰小 3. 易于存档	实时性差
	面对面审查	实时性好	对审查者干扰大
审查人数	一对一审查	1. 安排起来容易 2. 干扰小	多人同时线下审查容易出现重复工作
	团队集体审查	讨论深入，审查细致	人数多时，容易效率低下
审查范围	增量审查	聚焦重点效率高	无明显缺点
	全量审查	1. 系统性 2. 专项集中检查	工作量大，不能常常进行
审查时机	代码入库前门禁检查	对于把关入库代码的质量，效果很好	太过死板的话，会降低代码入库效率
	代码入库前的设计时检查	1. 最早发现问题，从而大大降低问题修复成本 2. 代码作者抵触情绪小 有效的架构讨论工具	无明显缺点
	代码入库后检查	既不阻塞代码入库，又可以对提交的代码进行审查	有问题代码错过检查而上线的风险

总体来说，绝大部分团队都适合引入工具进行异步的一对一审查，在互联网上也有比较多的关于这种审查方式的最佳实践推荐。比如，最近 Google 发表了他们的代码审查指南（Google’ s Code Review Documentation），你可以参考。

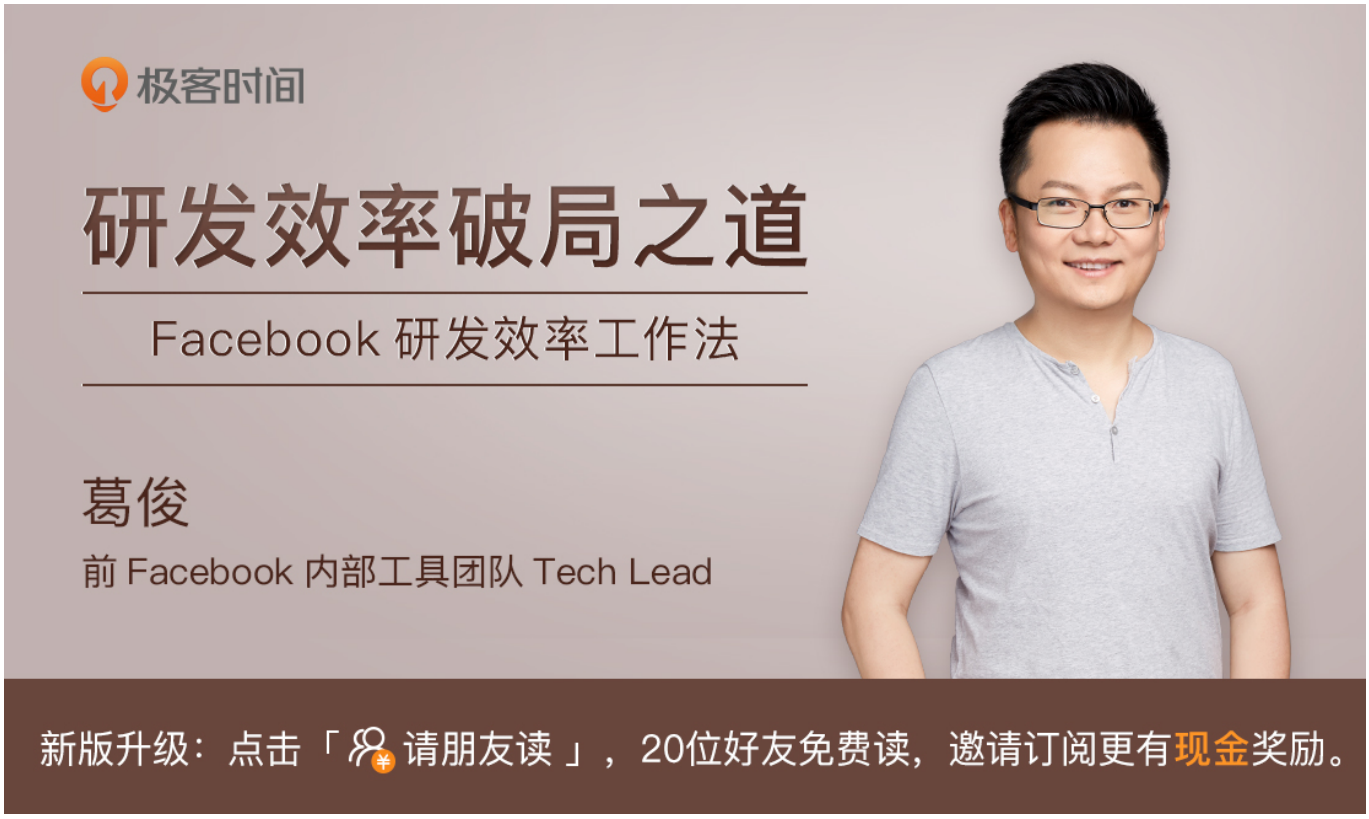
另外，多使用一些设计时审查尽早进行讨论一般也都效果不错。

知道了应该使用哪个方式，下一步就该具体的引入了。在下一篇文章，我会和你介绍成功引入代码审查的几个具体实践。

思考题

1. 你们团队使用了代码审查吗？具体使用了哪几种审查方式呢？
2. 设计时检查除了可以避免后期对代码的大规模调整外，对顺利引入代码审查还有一些其他作用。你能想到还有哪些作用吗？

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！



上一篇　11 | 研发环境：Facebook怎样让开发人员不再操心环境?

下一篇　13 | 代码审查：学习Facebook真正发挥代码审查的提效作用

精选留言 (5)

写留言

李双
2019-09-18

代码审查，很全面！非常赞同设计时审查！架构对了，细节容易调整！

1

日拱一卒
2019-09-22

1. 你们团队使用了代码审查吗？具体使用了哪几种审查方式呢？
我们一般有设计审查和代码审查。设计审查需要全部人员参加，主要是统一大家的认识。代码审查采用离线代码审查的方式，每个team的team lead需要负责审查组员提交的所有代码，组员之间的互相审查比较少，因为组员的技术栈不太一样，有时不太容易给出比较专业的审查结果。这种方式的不好的地方1. 要求team lead有全栈经验，即使有些方面精...

展开

吕哲
2019-09-19

还是一个很好的交流和学习设计模式及方法的机会😉

展开

作者回复: 👍👍👍

li3huo
2019-09-18

linkin 的由作者发起的 code review 方式也很有特色，要求作者组织材料和会议，从而激励期责任感，提升审查效率

展开

Geek_1988
2019-09-18

思考题2：设计时审查可以帮助审查者熟悉代码架构，提高审查者的审查效率。

作者回复: 👍👍👍