20 | 答疑篇：如何平衡短期收益和长期收益？

2019-10-09 葛俊

研发效率破局之道

进入课程 >

讲述：葛俊

时长 12:32　大小 11.50M

你好，我是葛俊。今天，我来针对“工程方法”模块的留言问题，做一次集中解答与扩展吧。

针对“工程方法”模块的文章，很多同学留下了精彩的留言，阐述了自己对研发效能的认识、遇到的问题以及解决方案。

虽说我已经一一回复过你们的留言了，但有些问题在我看来非常有价值，值得再扩展一下。所以，我今天挑选了 3 个典型问题进行详细回答，并对每一个问题延展出一个话题和你讨论，希望能帮助你加深对这些问题的理解，切实帮助你提高你或者你团队的研发效能。

问题一：因为赶进度就慢慢不做代码审查了

@john_zhang 同学在第 13 篇文章中，提出了这样一个问题：

> 我们推行过一段时间的代码审查，因为团队只有三五个人，所以采用的是团队审查，每天半个小时左右，可惜后来赶开发进度，慢慢就没做了。现在开发同事总是以赶进度为由，不太认同代码审查，怎么破？

这条留言下有好几位同学都点赞了，看来这是一个常见的情况。所以，我再和你分享下针对这个问题的具体解决办法吧。

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. 减少团队审查，更多地使用工具进行一对一的代码审查。因为团队审查很难做到效率高，所以应该只是针对一些重点的提交才采用这种方式。
3. 培训团队，统一认识，让大家看到代码审查的长期收益，比如考虑代码的可维护性以及后续添加新功能的速度，而不只是盯住当前的开发进度这个短期收益。甚至，我们可以把代码审查这个操作，作为团队的制度要求大家执行。

说到这里，又涉及了长期收益和短期收益的冲突要怎么权衡，这也是我们普遍关注的一个问题。接下来，我再和你扩展下这个问题吧。

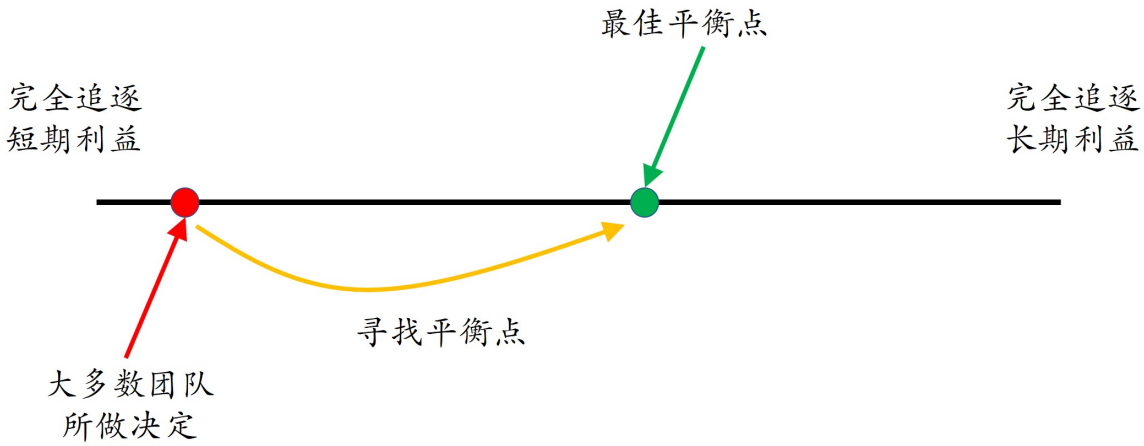
比如，不止一个同学提到“公司连给研发人员配置好一点的电脑的意识都没有，何谈研发效能？” 的确，使用配置低的电脑能够节省当前的开支，但是会损害长期的开发效率。这里的短期利益便是当前开支，而长期利益就是公司的长期产出。

又比如，我在第 14 篇文章中与你讨论的技术债问题，借债是考虑短期利益，而还债考虑的就是长期利益。

从这些例子可以看到，**短期利益和长期利益往往会存在冲突，我们必须做好取舍，才能最大程度地提高研发效能。**在我看来，短期利益和长期利益的权衡，可以重点考虑以下两个方面。

第一，寻找两者最佳的平衡点。比如，在一条线段上，最左边的端点是只关注短期利益，最右边则是只关注长期利益，两个极端情况都不好。但是，我见到的大部分情况是，团队做的选择会特别靠近最左边。虽然说有些选择是出于业务压力情非得已，但有相当一部分情况，是因为短视造成的。**由于太追逐眼前利益，而造成长期的生产力下降、研发人员积极性下降、产出降低的例子比比皆是。**

为了避免后劲不足的情况，我们必须往中间移动，找到最佳平衡点。



第二，随着时间推移，这个最佳平衡点也会有变动。比如，在第 14 篇文章末尾，我给你举的 A、B、C 三个公司对待技术债的案例。C 公司在不同的时间分别选择了最合适的平衡点，很好地平衡了借债与还债，从而获得了竞争优势。这里，我建议你结合长期利益与短期利益的权衡，再返回第 14 篇文章，回顾下这个案例。

问题二：截屏上传工具链的实现

在第 11 篇文章中，我与你介绍了一个截屏工作链，具体步骤包括截屏、上传图片、获取 URL、缩短 URL、保存到系统剪贴板。这样一来，我们使用快捷键触发截屏流程之后，就可以直接用 Cmd+V 把图片的短 URL 粘贴到 Commit Message 或者聊天工具中了。

有许多同学对这个工具很感兴趣，询问我如何实现，以及是否有对应的开源产品。不过遗憾的是，我没有找到对应的开源产品和方法。所以，我再和你说说，我之前在 Stand 公司时是如何实现的吧：

图像存储用的是 Google Drive；

截屏上传工具是Share Bucket。它支持上传到指定的 Google Drive，同时还支持存储 URL 到剪贴板。还有一个类似的工具是MonoSnap。

至于 URL 缩短服务，我们当时并没有使用。如果你考虑使用的话，可以看一看开源的 YOURLS。

这里，感谢 @日拱一卒同学，和我们分享了他们公司的实现方法：

> 我们的解决方案和你说的差不多，涉及到不同的工具，处理方式不太一样。
>
> 1. 如果工具本身支持图片存储，例如 ZenHub 或者 JIRA，我们用工具本身来存储图片。
> 2. 如果工具本身不支持图片存储，就用公司提供的网盘来存储图片，在工具中引用相关的链接。

在我看来，第二种方法会更好一点。因为有一个统一存储图片文件的地方，不同的工具都可以指向同一张图片。

其实，在 Facebook，还有另外一个类似的工具链，只不过它针对的不是截屏，而是拷贝文本。

我们在工作中常常会针对一段文本进行讨论，但如果这段文本特别长的话，把它拷贝到讨论工具中，就会占用很大的空间。所以，我们就用了一个工具链来解决这个问题。使用步骤也类似：

1. 拷贝要讨论的文本。
2. 用快捷键触发工具链。这些工具会自动把文本上传到一个服务器，然后把 URL 缩短，并把缩短后的 URL 保存到系统剪贴板里。
3. 使用快捷键 Cmd+v 把短 URL 粘贴到讨论工具（比如 Commit Message）中，其他人点击这个 URL 就可以到文本服务器上查看具体的文本内容。

这个工具链涉及了文本存储服务，PasteBin是目前最流行的工具，不过它并没有开源，而且只提供 SaaS 服务。而我们日常工作中拷贝的文本，通常是不能放到公网上的。所以，PasteBin 很可能不适合你。这里，我推荐一个类似的开源工具HasteBin，你可以安装到公司内网使用。

关于这个截屏工具流，**我想扩展的一个话题是，研发效能的提高带来的量变会产生质变**，也就是说研发效能的提高，可能看起来只是做事情快了一些，产出高了一些，甚至有人会认为这并没有什么太大的作用。但实际上，效能提高累积到一定阈值之后，就可以带来质变。

以截屏工具链为例。在没有类似工具链的情况下，虽说可以手动实现上面的工作，但因为操作步骤太繁琐，所以我们在 Commit Message 里很少使用截屏。而这个工具链将这些繁琐的操作简单化了，由此带来的效率提升让绝大多数开发者都乐于在 Commit Message 中使

用截屏去讨论问题。也就是说，这个工作流带来的效能提升累积到了一定的阈值之后，引发了质变。

提升研发效能的量变带来的质变，另一个表现是，效能的提升增强了公司整体的竞争力。当效能提高到一定程度的时候，就可以在公司竞争中起到重要作用甚至是决定性作用。

在我看来，这并不是夸大之词，尤其是在软件开发行业逐步成熟，需要比拼研发效能的今天。

问题三：Facebook 的 SEV 系统是做什么用的?

@Geek_f0179a 同学在第 4 篇文章中问到：Facebook 有一个 SEV 复盘系统，这个系统的主要功能是什么，会记录哪些关键信息呢?

SEV 系统是一个事故响应及根因分析系统，SEV 是严重性“Severity”的前三个字母。这个系统会根据事故的严重性为其分配一个数字，大概是 1~4：SEV1 最严重，SEV4 最轻。每次发生事故，除了记录严重性之外，还要按时间顺序记录事故发生、发现、定位、解决、确认解决过程的每一步的时长和具体操作细节。

每周公司都会定时举行 SEV 讨论会，基本只讨论严重的问题，比如 SEV1 和 SEV2 的事故。会议由高层管理者主持，事故责任人轮流进入会议室对事故进行描述和讨论。

这个会议的主要目的是，对事故进行回溯，分析根因以及如何避免再次出现类似问题。也就是说，SEV 系统的目的，重在总结提高，不要多次重复同一个错误，如果重复犯错要惩罚。

SEV 系统的效果很好，在避免问题重复出现的同时，我们也比较敢于试错，至少从我的经验看是这样的。

针对这个事故复盘系统，我想扩展的话题是：出现问题之后，到底应不应该追责，应该怎样追责?

在我看来，考虑这个问题应该从最终目的出发。复盘的目的是最大程度地减少以后再出现同一个问题的概率。那我们应该怎样处理呢?

如果每个错误都要惩罚，就会有以下几个问题：

因为害怕惩罚，尽量去掩盖问题而不是暴露问题、分析问题，或者尽量撇清责任甩锅。

惧怕闯祸，团队成员开始信奉“多做不如少做”。

因为担心指出别人的问题，会让他受罚，结果就是不愿意得罪人，导致问题被隐藏。

而另一个极端，如果犯错没有任何惩罚的话，又会造成以下问题：

同样的错误一犯再犯。

没有奖惩造成不公平。这就会让原本认真工作的员工失去认真做事的动力，降低对代码质量、产品质量的关注，提高质量的工程措施（比如，单元测试、开发自测等）更是难以推广。

所以，在我看来，针对错误更有效的处理方法应该是，以保持团队的主动性、责任感和执行力为原则，具体措施包括：

追究责任，但不是惩罚。知其然并知其所以然，搞清楚前因后果，避免重复犯同一个错误。

重复犯错一定要惩罚。

反复问为什么，找到根本原因。

小结

好了，以上就是今天的主要内容了。如果有哪些你希望深入了解还未涉及的话题，那就直接给我留言吧。

接下来，我们就会开启新的模块了，进入个人效能的讨论了，希望这个模块能够帮助你提升个人效能，持续成长，成为 10x 开发者，提升个人竞争力。

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　19 | 不再掉队，研发流程、工程方法趋势解读和展望

下一篇　21 | 高效工作：Facebook的10x程序员效率心法

精选留言 (3)

写留言

兴国
2019-10-09

代码审查是必须要做的，作为团队leader要有这个意识，更要起到带头作用。
出现bug，解决bug。解决后反复讨论找到根本问题，然后优化流程，加强执行力。防止类似事情再次出现，降低bug率。

展开

作者回复: 是的。这些都是值得投入的，有长期收益的事！

3

日拱一卒
2019-10-12

针对产品环境报出的问题，如何去处理，是一个需要把握度的问题，之前有其他课程也提过，大体思路是相通的。
1. 鼓励快速试错，错误次数不应该直接和KPI挂钩。
2. 出错后要找原因，多问为什么，找到根本原因和解决方案，并维护好相关的知识库，方便后来的新组员快速熟悉，避免再次踩坑。...
展开

李双
2019-10-10

学习
展开

作者回复: 👍👍👍