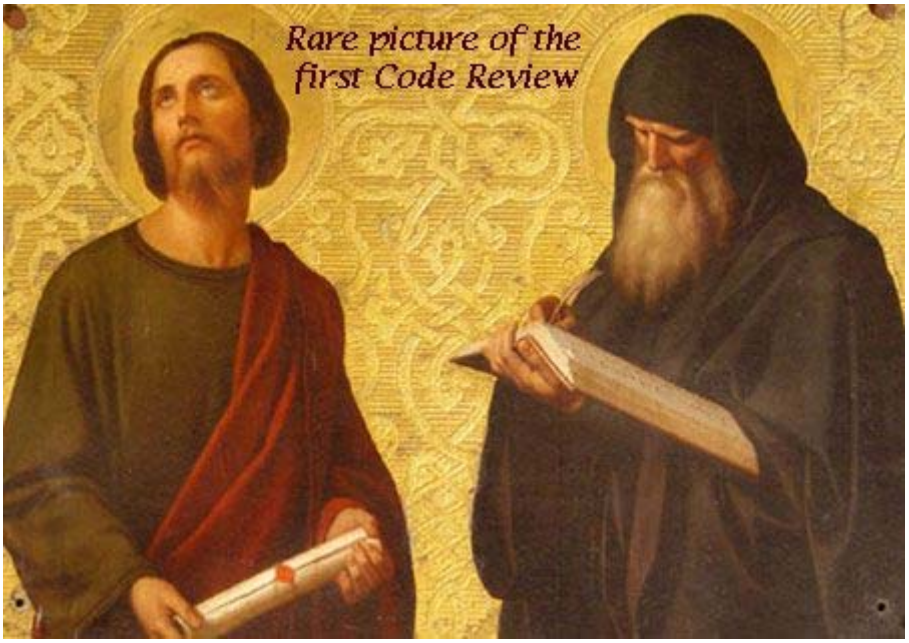
代码论千言：如何评价别人的代码

引子

> 遇到一个陌生人，只看到他沾满泥土的鞋子。
>
> 夏日地铁里，闻到了周围人的汗臭。
>
> 朋友聚会，似乎有些人眼屎没擦干净。
>
> 初次相亲见面，就一直在说对方不好看。
>
> 你看到别人的种种不足，当面指出来，为什么没有人感谢你？
>
> 不光没有人感谢，似乎还有人恶言相向，某些人还因此要打你。
>
> 你做错了吗？
>
> ————《帐前卒的寓言》



Code Review作为编码社会活动，一直被人诟病。每个人似乎都知道Code Review好，但是推广起来却有种种困难。而“用IDE编程”这事似乎根本就不用推广。这到底是为什么？因为Code Review中不可避免要指出别人的代码不足。我们也经常会遇到：

- 评价/建议后，编码者却不修改；

- 总是同一种错误，一犯再犯；
- 抵制你的评价意见，寻求他人再次Review。

我们review别人的代码的代码，不是专门为代码点赞，那是“鼓励师”要做的事情。我们主要是指出别人写的代码哪里不好。我想，即使是最和善、最理智的程序员也肯定不会喜欢那些评论。即便如此，我们还是要评论，要让别人接受我们的观点。不过为了让别人乐于接受，肯定要有恰当的鼓励。另外，在评论别人的代码之前，我们最好知道是谁写的代码。如果是匿名的，当然可以肆无忌惮的评论。但Code Review是一种非匿名的社交活动：我们都知道编码者是谁，编码者也知道评价者是谁。与说话的技巧一样：“对什么人说什么话”。所以我们首先需要了解编码者。

读代码识人心

> 比鬼神更可怕的，是人心。
>
> ————《盗墓笔记》

读懂人心，实在是太难了。通过代码去读懂人心，那就更难了。如果和编码者经常在一起，性格喜好自然能略知一二。如果你从来没有接触过编码者，只看代码能了解什么呢？我们无法从代码中看出那人的血型、星座、爱好；也难看出勇敢、乐观、顽强、抑郁或悲伤；更看不出他是表现型、分析性还是实干型。当然以上那些性格特点对我们评价代码也没有什么卵用。从代码中读懂人心，只需关注三个方面：

- 是否认真编码
- 是否对技术有激情
- 编码水平怎么样

为什么只关注这三个方面？认真的编码者对细节格外关心，指明他代码细节的缺陷，他的技术水平一定会有提升。有激情的编码者一定会迫不及待的使用新的技术、新的语法、新的组件。应该告诉他们这些新的技术、语法、组件在代码里用的是否恰当。高水平的编码者乐于提高自己的编程技巧，也喜欢破坏现有的规则，他们看重的只有逻辑错误和运行时问题。Review时，投其所好，才能让编码者信任你，然后才可能接受你的建议。如果编码者编码不认真、技术没激情、水平也不咋样，请问该怎么办？这我们后面会说。

认真的编码者

编码者是否想到你所想到的细节，是否还考虑了你没有想到的点？是否遵循了code style规范中每一条细节？是否考虑了边界情况？在特殊处理的地方，有没有写详细的注释？当你评论的他的代码，不管同意与否，他是否每一条都认真回复了。

有激情的编码者

编码者是否尝试了新的技术：新的组件、新的语法糖？是否愿意解决一些别人不碰的难题？不那么美观的代码是否充满了不断尝试的味道？是否规避了存在问题的第三方组件？是否尝试写过一些公用组件？是否对旧的问题有新的思路?

高水平的编码者

编码水平有优劣，但这个优劣大多数只存在于人的内心。有时候你看了一眼别人的代码，你心里想：“这段代码能写的更简单，更容易。”所以优劣大多是和自己比较的：技术比自己好，还是比自己差。如果你熟悉整个团队，那么你就能知道这个人的编码水平大概处于整个团队的什么位置。

但是编码者提交的代码量太少，或者大多是log/print/bean，可能的确难以了解他的这三方面，那就用comments多交流几次。

书写评论

初识，最好多点赞，并建议如何编码。如果对方是新入职的人员，最好尽可能的一次性的指出程序要修改的地方。一个是他有时间修改；二是新人需要了解公司里的编码规范；三是提升他的编码水平。

在review的过程中不要否定新技术，也不要轻易肯定新技术，**适用才是评判依据**。为什么使用这些新技术？是旧的写起来麻烦，还是旧的无法满足需求，还是只为尝鲜？对于

调研团队，我们鼓励使用各种新的技术。不管是挖坑还是填坑。需要通过他们的使用情况了解多种新技术的功能特性、成熟度、性能优劣和适用条件。对于功能交付团队，不管是上线还是开发软件给客户使用，能满足需求的同时，一定要为了稳定不折腾。否则后期就会被bug所累，疲于奔命。

表达相同意思的一段文字可以有不同的写法。工作要有乐趣，不要评论千篇一律、枯燥乏味。下面我要说几种评论的风格:

- 吐槽。针对熟识的同事或者针对技术水平很高的人。幽默风趣，又让人容易接受。
- 委婉建议。针对对技术有激情的人，不应该抹杀别人的技术追求。多以疑问句为主，让其自己反思。
- 批评。针对多次犯同一错误的新人。这样的错误一般都是非常容易改正的。例如多写常量定义、使用linux换行符、打印不用System.out等等。有时用感叹句让批评的语气再严厉一点，否则下次新人还会犯同样的错误。
- 直白陈述。可能大多数采用的是这句式。另外针对编码得过且过的**老码农**，你可能需要多写点：除了点明问题所在，还应该建议他怎么去编写代码。这样可以减少他返工次数，消减他的抵触情绪。
- 讥讽。除非是你的好基友，否则不要用，不要用，不要用，说三遍，不要树敌。

既然当面交流，不妨听听编码者怎么看待你的评论。不要一言不合就暴跳如雷，要问问他想怎么修改，或者为什么不想修改。

出发点还是**“利”**字。但是受益者可能不同的：是编码者得利，还是团队得利，还是公司得利。这就要看你的三寸不烂之舌，如何既讨人欢心又让人听你指示。程序员这群理智的动物，有时最看不惯的就是阿谀奉承。大家都是靠技术混饭吃的，虽然糖衣炮弹固然重要，但更重要的是言明利弊。

有些人即使你心平气和的交流，仍然不愿意修改他们的代码，这时候就要关注问题的边界。如果在容忍的边界里，或者对建议置之不理，或者允许开发者以后再修改。如果是边界之外，就应该据理力争。那边界究竟是怎样划定的呢？我写在下面：

- 代码中存在bug。bug会导致各种错误，有些会导致数据错乱、文件错乱等难以修复的问题。

- 代码难以理解，日后难维护。当然如果写这代码的人永不离职，自己维护尚可。可人要是离职了，还要看后继者是否愿意担风险去重构这些代码。

- 长时间运行有问题。那多久会出问题呢？看多久再次部署，如果每天都部署一次，两天就算长时间了。但是为了考虑团队每个人都休息好、有自己的假期、以及不在休假时被打扰。还是尽量考虑长远一点。如果你最长假期有10天，那至少要考虑软件运行大半个月没问题。

- 数据量大、访问量大时代码运行缓慢或者宕机。如果估算运行部署几天后数据/访问量就会到达这个规模，那一定要尽早修改代码。

- 改动后将大量节省他人的时间。如果将代码抽取为模块或者组件后，有更大的收益。特别是他人即将开始这部分工作时，一定要尽快的抽取出相关代码。不抽取的话，一是别人也要写相关代码；二是一旦有修改，相同功能的代码都会修改；三是即使后期再抽取了组件，也没人愿意重构。

我们都是执着的技术人，我们心中自有不可逾越的底线，我们也希望所有的编码者不要越过这条底线。我们把这底线告诉给编码者，这也是Code Review中评审者的义务。但建议也写了，见面也谈过了，编码者和审阅者两方就是各不相让。那没有其他办法，只能通过公司里的人事关系去解决：

- 如果是你的下属，都已经讲明白道理，还是不执行，那是这个下属不称职。

- 如果是你的上级，你已经告知了他风险，他觉得你的建议没有道理，那就请他说出自己的道理。当然你觉得是他是一个傻X领导，拍拍屁股走人就是；但如果你**越职言事**，那就是大忌了。

- 如果是你的平级，告知他上级代码的风险，之后就交给他的上级处理吧。

当然你需要知道，为了权衡：有时为了紧急上线，有时为了更大的利。即使逾越了你的底线，仍会对建议置之不理————先把代码merge再说。

Code Review是为了什么？最终是为了获利。在公司，什么都不考虑，只为把代码写的完美无缺，那才是真正的傻。