41 | 沟通之痛：如何改变？

2018-11-05 胡峰

沟通问题，一直都是程序员的痛点。

隔壁专栏（左耳听风）的陈皓以前在他的博客上写过一篇文章叫《技术人员的发展之路》，里面提及职业发展到一定阶段，也许你就会碰上一些复杂的人和事，这种情况下他写道：

> 这个时候再也不是 Talk is cheap, show me the code! 而是，Code is cheap, talk is the matter!

这里的 Talk 其实就是沟通，在工作中你要是留心观察，就会发现很多沟通问题，比如，跨团队开会时常发生的一些分歧和争论。沟通，越发成为一件重要的事，至少和写代码同等重要；沟通清楚了，能让我们避免一些无谓的需求，少写不少无效的代码。

然而现实中， 沟通问题，有时却被作为程序员的我们有意或无意地回避与忽略了。面对沟通问题，我们该如何看待和分析这个问题，并做出一些改变呢？

一、木讷与沉默

木讷与沉默，这两个名词似乎已变成了程序员的标签，它们形象地体现了程序员在沟通中的表现。

在程序员的世界里，沟通的主要场景可能包括：与产品经理沟通需求，与测试同学推敲 Bug，与同行交流技术，给外行介绍系统，还有和同事分享工作与生活的趣闻，等等。然而，有些程序员在分享趣闻时，与谈需求或技术时的表现大相径庭，刚才明明还是一个开朗幽默的小伙，突然就

变得沉默不语了。

沉默有时是因为不想说。特别在沟通需求时，有些程序员默默不言，但心里想着：“与其扯那么多，倒不如给我省些时间写代码！”然而，程序员写出的代码本应该是公司的资产，但现实中代码这东西是同时带有资产和负债双属性的。

需求没沟通清楚，写出来的代码，即使没 Bug 将来也可能是负债。因为基于沟通不充分的需求写出来的代码，大部分都是负债大于资产属性的，这最后造成的后果往往是：出来混都是要还的，不是自己还就是别人来还。

有些程序员可能会争辩道，“与人沟通本来就不是我们所擅长的，再说了我们也并不是因为热爱跟别人聊天才做软件开发这一行的。”这个言论很有迷惑性，我早年一度也是这么认为的。

我毕业去找工作那年，外企如日中天，所以我也去了当时心中很牛的 IBM 面试。面试过程中的大部分交谈过程和内容现在我都记不清了，但就有一个问题至今我还记忆犹新。面试经理问我：“你是喜欢多些跟人打交道呢，还是跟电脑打交道？”当时的我毫不犹豫地回答：“喜欢跟电脑打交道，喜欢编程写代码，而且我自觉也不擅长和人打交道。”

然后，我就被淘汰了。后来我才明白了，其实当时的这类外企挂着招工程师的名义，实际需要的更多是具有技术背景和理解的售前技术支持，因而就需要所招之人能更多地和人沟通去解决问题，而不只是写代码解决问题。

结合我自己多年的工作经历和经验来看，即便你仅仅只喜欢写代码，那么和人的沟通能力也依然是你必须跨过去的门槛。《计算机程序的结构与解释》有云：“程序写出来是给人看的，附带能在机器上运行。”

其实，写代码本身也是一种沟通，一种书面沟通。沟通从来都是个问题，书面沟通也同样困难。

二、争论与无奈

程序员最容易产生沟通争论的地方：**沟通需求**和**沟通技术方案**。

在程序员的职业生涯路上，我们不可避免地会碰到和同事关于技术方案的争论。我从中得到的教训是：千万不要让两个都自我感觉很牛的程序员去同时设计一个技术方案。

假如不巧，你已经这么干了并得到了两个不同的方案，那么记住，就别再犯下一个错：让他们拿各自的方案去 PK。因为这样通常是得不到你想要的“一个更好的方案”，但却很可能会得到“两个更恼怒的程序员”。

既然分歧已经产生了，为了避免无谓的争论，该怎么解决呢？

1. 以理服人

首先，把握一个度，**对事不对人**，切勿意气用事。

有些程序员之间的分歧点是非常诡异的，这可能和程序员自身的洁癖、口味和偏好有关。比如：大小写啦、命名规则啦、大括号要不要独立一行啦、驼峰还是下划线啦、Tab 还是空格啦，这些都能产生分歧。

如果你是因为“该怎么做某事或做某事的一些形式问题”与他人产生分歧，那么在很多情况下，你最好先确定分歧点是否值得你去拼命维护。这时，你需要判断一下：技术的“理”在什么地方？这个“理”是你值得拼命坚守的底线不？用这个“理”能否说服对方吗？

我所理解的技术的“理”包括：先进性、可验证性、和团队的匹配性、时效性、成本与收益。另外还有一些不合适的“理”，包括：风格、口味、统一、政治等。

不过有时候，有“理”也不代表就能搞定分歧，达成一致。毕竟林子大了，不讲“理”的人也是有的，那么，就需要换一种方式了。

2. 以德服人

分歧进入用“理”都无法搞定时，那就是应了那句古词：“剪不断，理还乱”。

这时继续“理”下去，不过都是互相耍混罢了。“理”是一个需要双方去客观认可的存在，而越“理”越乱则说明双方至少没有这种客观一致性的基础，那就找一个主观的人来做裁决吧。

这个人通常就是公司所谓的经验丰富、德高望重的“老司机”了，并且双方也都是认可的，比如架构师之类的。但是这类主观裁决也不一定能保证让双方都满意，有时实力相当的技术人也容易产生类似文人相轻的状况。不过看在“老司机”的“德”面上，也能勉强达成一致。

“老司机”裁决最好站在他所认同的“理”这个客观存在上，这是最好的，不过这也取决于“老司机”的工作素养和价值观了。

3. 以力服人

最差的状况就会走到这一步，特别在跨大部门的沟通中。

技术方案无法达成一致，也没有一个跨两个部门的有德之人可以转圜化解，就会升级到这个地步。最后就是靠粗暴的权力来裁决，看双方部门老大或老大的老大，谁更有力或给力。一般来说，非关键利益之争实在没有必要走到这一步了。

三、认识与改变

做出改变的第一步是要能认识到，否则改变不可能发生。

程序员会认识到沟通很重要，有时甚至会比写代码更重要吗？著名的技术型问答网站——Stack Overflow的两位创始人杰夫·阿特伍德（Jeff Atwood）和乔尔·斯波尔斯基（Joel Spolsky）都对此有正面的认识和见解。

杰夫说：

> 成为一名杰出的程序员其实跟写代码没有太大关系。
>
> 做程序员确实需要一些技术能力，当然还要有坚韧不拔的精神。
>
> 但除此之外，更重要的还是要有良好的沟通技巧。

乔尔的观点是：

> 勉强过得去的程序员跟杰出程序员的不同之处，不在于他们掌握了多少种编程语言，也不在于他们谁更擅长 Python 或 Java。
>
> 真正关键的是，他们能不能把他们的想法表达清楚，杰出的程序员通过说服别人来达成协作。
>
> 通过清晰的注释和技术文档，他们让其他程序员能够读懂他们的代码，这也意味着其他程序员能够重用他们的代码，而不必重新写过。
>
> 要不然，他们代码的价值就大打折扣了。

按照程序员解决技术问题的习惯，就是把一个大问题拆解为多个部分的小问题，那这里我们也对沟通做下拆解，它包括三个方面：

- 内容
- 形式
- 风格

从内容上看，虽说你想沟通的本质是同一样东西或事情，但针对不同的人，你就需要准备不同的内容。比如，同内行与外行谈同一个技术方案，内容就是不同的。这里就需要你发挥同理心和换位思考的能力。保罗·格雷厄姆（Paul Graham）曾在他的书《黑客与画家》中写道：

> 判断一个程序员是否具备“换位思考”的能力有一个好方法，那就是看他怎样向没有技术背景的人解释技术问题。

换位思考本质上就是沟通技巧中的一种。

从形式上看，沟通其实不局限于面对面的谈话。面对面交谈是一种形式，书面写作又是另外一种形式，连写代码本身都是在和未来的自己或某个你尚未谋面的程序员沟通。

程序员确实有很多都不擅长面对面的沟通形式。面对面沟通的场景是很复杂的，因为这种沟通中交流传递的载体不仅仅是言语本身，眼神、姿态、行为、语气、语调高低，甚至一种很虚幻的所谓“气场”，都在传递着各种不同的信息。而大部分人都不具备这种同时控制好这么多传递渠道的能力，也即我们通常说的“缺乏控场能力”，这里面隐含着对你其他能力的要求，比如：临场应变、思维的活跃度与变化等。

从风格上看，不同方式和场景的沟通可以有不同的风格。比如面对面沟通，有一对一的私下沟通，风格就可以更随性柔和些；也有一对多的场景，比如演讲、汇报和会议，风格就要正式一些，语言的风格可能就需要更清晰、准确和锐利一些。

沟通之难就在于清晰地传递内容和观点。当你要向其他人详细解释某样东西的时候，你经常会惊讶地发现你有多无知，于是，你不得不开始一个全新的探索过程。这一点可以从两个方面来体会：

1. 你只需要尝试写点你自认为已经熟悉掌握的技术，并交给读者去阅读与评价。
2. 每过一段时间（比如，一个季度或半年）尝试去总结，然后给同事分享下你工作的核心内容，再观察同事们的反应和听取他们的反馈，你就能体会到这一点了。

所以，沟通改变的第一步就是从考虑接收方开始的，看看接收方能吸收和理解多少，而非发送了多少。而沟通问题的三个方面——内容、方式与风格——的考虑，都是为了让接收更方便和容易。

江山易改，本性难易，有时候我们做不到就在于这一点。但现实并不要求程序员成为所谓的沟通达人，只是需要主动认识到身边的沟通问题，去进行理性和逻辑地分析、拆解并做出适当的调整。

从认识我们的本性开始，控制情绪，从而去规避无奈的争论；认识清楚沟通问题的本质是要方便接收，达成共识，保持换位思考和同理心，改变自会发生。

关于沟通的各种形式，每个人可能都有自己擅长和偏好的方面，比如我就更擅长文字沟通，而不擅长一对一当面沟通，那么你呢？
